Foundation Framework Reference

Cocoa > Objective-C Language

2008-06-27

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

Simultaneously published in the United States and Canada.

Contents

Figures and Tables

The Foundation Framework

Framework	/System/Library/Frameworks/Foundation.framework
Header file directories	/System/Library/Frameworks/Foundation.framework/Headers
Declared in	FoundationErrors.h IKPictureTaker.h NSAffineTransform.h NSAppleEventDescriptor.h NSAppleEventManager.h NSAppleScript.h NSArchiver.h NSArray.h NSAttributedString.h NSAutoreleasePool.h NSBundle.h NSByteOrder.h NSCalendar.h NSCalendarDate.h NSCharacterSet.h NSClassDescription.h NSCoder.h NSComparisonPredicate.h NSCompoundPredicate.h NSConnection.h NSData.h NSDate.h NSDateFormatter.h NSDecimal.h NSDecimalNumber.h NSDictionary.h NSDistantObject.h NSDistributedLock.h NSDistributedNotificationCenter.h NSEnumerator.h NSError.h NSException.h NSExpression.h NSFileHandle.h NSFileManager.h NSFormatter.h NSGarbageCollector.h NSGeometry.h NSHFSFileTypes.h NSHTTPCookie.h NSHTTPCookieStorage.h NSHashTable.h NSHost.h

NSIndexPath.h
NSIndexSet.h
NSInvocation.h
NSJavaSetup.h
NSKeyValueCoding.h
NSKeyValueObserving.h
NSKeyedArchiver.h
NSLocale.h
NSLock.h
NSMapTable.h
NSMetadata.h
NSMethodSignature.h
NSNetServices.h
NSNotification.h
NSNotificationQueue.h
NSNull.h
NSNumberFormatter.h
NSObjCRuntime.h
NSObject.h
NSObjectScripting.h
NSOperation.h
NSPathUtilities.h
NSPointerArray.h
NSPointerFunctions.h
NSPort.h
NSPortCoder.h
NSPortMessage.h
NSPortNameServer.h
NSPredicate.h
NSProcessInfo.h
NSPropertyList.h
NSProtocolChecker.h
NSProxy.h
NSRange.h
NSRunLoop.h
NSScanner.h
NSScriptClassDescription.h
NSScriptCoercionHandler.h
NSScriptCommand.h
NSScriptCommandDescription.h
NSScriptExecutionContext.h
NSScriptKeyValueCoding.h
NSScriptObjectSpecifiers.h
NSScriptStandardSuiteCommands.h
NSScriptSuiteRegistry.h
NSScriptWhoseTests.h
NSSerialization.h
NSSet.h
NSSortDescriptor.h
NSSpellServer.h
NSStream.h
NSString.h
NSTask.h

NSThread.h
NSTimeZone.h
NSTimer.h
NSURL.h
NSURLAuthenticationChallenge.h
NSURLCache.h
NSURLConnection.h
NSURLCredential.h
NSURLCredentialStorage.h
NSURLDownload.h
NSURLError.h
NSURLHandle.h
NSURLProtectionSpace.h
NSURLProtocol.h
NSURLRequest.h
NSURLResponse.h
NSUndoManager.h
NSUserDefaults.h
NSValue.h
NSValueTransformer.h
NSXMLDTD.h
NSXMLDTDNode.h
NSXMLDocument.h
NSXMLElement.h
NSXMLNode.h
NSXMLNodeOptions.h
NSXMLParser.h
NSZone.h
QTKitDefines.h

Introduction

The Foundation framework defines a base layer of Objective-C classes. In addition to providing a set of useful primitive object classes, it introduces several paradigms that define functionality not covered by the Objective-C language. The Foundation framework is designed with these goals in mind:

- Provide a small set of basic utility classes.
- Make software development easier by introducing consistent conventions for things such as deallocation.
- Support Unicode strings, object persistence, and object distribution.
- Provide a level of OS independence, to enhance portability.

The Foundation framework includes the root object class, classes representing basic data types such as strings and byte arrays, collection classes for storing other objects, classes representing system information such as dates, and classes representing communication ports. See Figure I-1 (page 40) for a list of those classes that make up the Foundation framework.

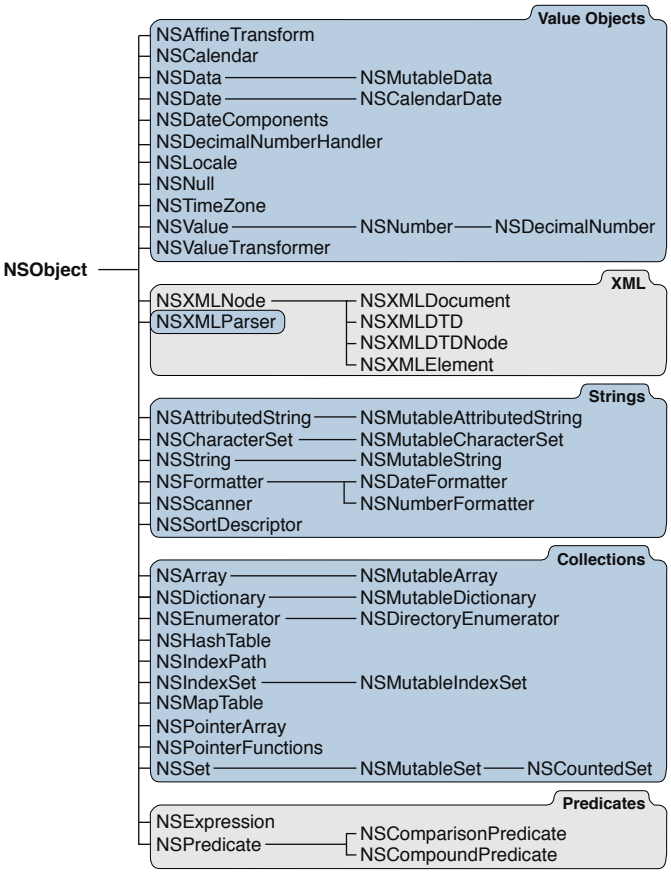
The Foundation framework introduces several paradigms to avoid confusion in common situations, and to introduce a level of consistency across class hierarchies. This consistency is done with some standard policies, such as that for object ownership (that is, who is responsible for disposing of objects), and with abstract classes like `NSEnumerator`. These new paradigms reduce the number of special and exceptional cases in an API and allow you to code more efficiently by reusing the same mechanisms with various kinds of objects.

Foundation Framework Classes

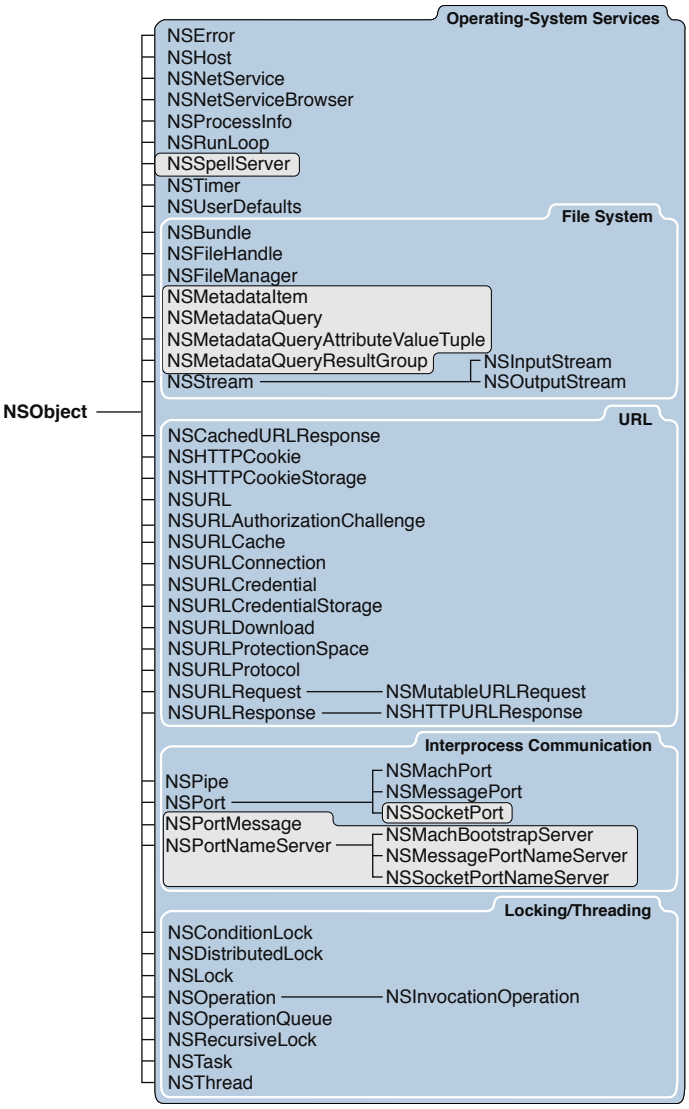
The Foundation class hierarchy is rooted in the Foundation framework's `NSObject` class (see Figure I-1 (page 40)). The remainder of the Foundation framework consists of several related groups of classes as well as a few individual classes. Many of the groups form what are called class clusters—abstract classes that work as umbrella interfaces to a versatile set of private subclasses. `NSString` and `NSMutableString`, for example, act as brokers for instances of various private subclasses optimized for different kinds of storage needs. Depending on the method you use to create a string, an instance of the appropriate optimized class will be returned to you.

Note: In the following class-hierarchy diagrams, blue-shaded areas include classes that are available in Mac OS X and iPhone OS; gray-shaded areas include classes that are available in Mac OS X only.

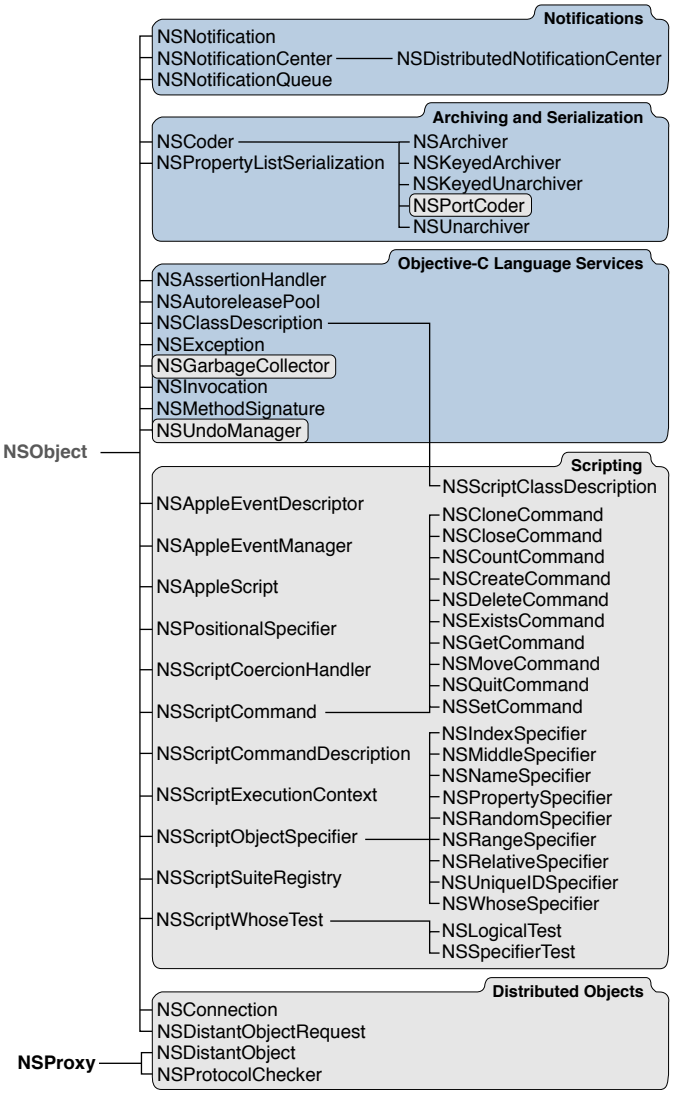
Figure I-1 Cocoa Objective-C Hierarchy for Foundation



Objective-C Foundation Continued



Objective-C Foundation Continued



Many of these classes have closely related functionality:

- Data storage. `NSData` and `NSString` provide object-oriented storage for arrays of bytes. NSValue and NSNumber provide object-oriented storage for arrays of simple C data values. `NSArray`, `NSDictionary`, and `NSSet` provide storage for Objective-C objects of any class.
- Text and strings. `NSCharacterSet` represents various groupings of characters that are used by the `NSString` and `NSScanner` classes. The `NSString` classes represent text strings and provide methods for searching, combining, and comparing strings. An `NSScanner` object is used to scan numbers and words from an `NSString` object.
- Dates and times. The `NSDate`, `NSTimeZone`, and `NSCalendar` classes store times and dates and represent calendrical information. They offer methods for calculating date and time differences. Together with `NSLocale`, they provide methods for displaying dates and times in many formats, and for adjusting times and dates based on location in the world.

- Application coordination and timing. `NSNotification`, `NSNotificationCenter`, and `NSNotificationQueue` provide systems that an object can use to notify all interested observers of changes that occur. You can use an NSTimer object to send a message to another object at specific intervals.
- Object creation and disposal. `NSAutoreleasePool` is used to implement the delayed-release feature of the Foundation framework.
- Object distribution and persistence. The data that an object contains can be represented in an architecture-independent way using `NSPropertyListSerialization`. The `NSCoder` and its subclasses take this process a step further by allowing class information to be stored along with the data. The resulting representations are used for archiving and for object distribution.
- Operating-system services. Several classes are designed to insulate you from the idiosyncrasies of various operating systems. `NSFileManager` provides a consistent interface for file operations (creating, renaming, deleting, and so on). `NSThread` and `NSProcessInfo` let you create multithreaded applications and query the environment in which an application runs.
- URL loading system. A set of classes and protocols provide access to common Internet protocols.

Classes

NSAffineTransform Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSAffineTransform.h
Companion guide	Cocoa Drawing Guide
Related sample code	DockTile SpeedometerView Transformed Image WebKitPluginStarter WebKitPluginWithJavaScript

Overview

The `NSAffineTransform` class provides methods for creating, concatenating, and applying affine transformations.

A transformation specifies how points in one coordinate system are transformed to points in another coordinate system. An affine transformation is a special type of transformation that preserves parallel lines in a path but does not necessarily preserve lengths or angles. Scaling, rotation, and translation are the most commonly used manipulations supported by affine transforms, but shearing is also possible.

Note: In Mac OS X v10.3 and earlier the `NSAffineTransform` class was declared and implemented entirely in the Application Kit framework. As of Mac OS X v10.4 the `NSAffineTransform` class has been split across the Foundation Kit and Application Kit frameworks.

Methods for applying affine transformations to the current graphics context and a method for applying an affine transformation to an `NSBezierPath` object are described in NSAffineTransform Additions in the Application Kit.

Adopted Protocols

NSCoding
- `- encodeWithCoder:` (page 2034)
- `- initWithCoder:` (page 2034)

NSCopying
- `- copyWithZone:` (page 2042)

Tasks

Creating an NSAffineTransform Object

`+ transform` (page 49)
Creates and returns a new `NSAffineTransform` object initialized to the identity matrix.

`- initWithTransform:` (page 50)
Initializes the receiver's matrix using another transform object and returns the receiver.

Accumulating Transformations

`- rotateByDegrees:` (page 51)
Applies a rotation factor (measured in degrees) to the receiver's transformation matrix.

`- rotateByRadians:` (page 52)
Applies a rotation factor (measured in radians) to the receiver's transformation matrix.

`- scaleBy:` (page 53)
Applies the specified scaling factor along both x and y axes to the receiver's transformation matrix.

`- scaleXBy:yBy:` (page 53)
Applies scaling factors to each axis of the receiver's transformation matrix.

`- translateXBy:yBy:` (page 56)
Applies the specified translation factors to the receiver's transformation matrix.

`- appendTransform:` (page 49)
Appends the specified matrix to the receiver's matrix.

`- prependTransform:` (page 51)
Prepends the specified matrix to the receiver's matrix.

`- invert` (page 50)
Replaces the receiver's matrix with its inverse matrix.

Transforming Data and Objects

`- transformPoint:` (page 54)
Applies the receiver's transform to the specified `NSPoint` data type and returns the results.

- `transformSize:` (page 55)
  Applies the receiver's transform to the specified `NSSize` data type and returns the results.

Accessing the Transformation Structure

- `transformStruct` (page 55)
  Returns the matrix coefficients stored in the receiver's matrix.
- `setTransformStruct:` (page 54)
  Replaces the receiver's transformation matrix with the specified values.

Class Methods

transform

Creates and returns a new `NSAffineTransform` object initialized to the identity matrix.

```
+ (NSAffineTransform *)transform
```

Return Value
A new identity transform object. This matrix transforms any point to the same point.

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

See Also
- `initWithTransform:` (page 50)

Related Sample Code
DockTile
Sketch-112
SpeedometerView
WebKitPluginStarter
WebKitPluginWithJavaScript

Declared In
`NSAffineTransform.h`

Instance Methods

appendTransform:

Appends the specified matrix to the receiver's matrix.

```
- (void)appendTransform:(NSAffineTransform *)aTransform
```

Parameters

aTransform

The matrix to append to the receiver.

Discussion

This method multiplies the receiver's matrix by the matrix in *aTransform* and replaces the receiver's matrix with the results. This type of operation is the same as applying the transformations in the receiver followed by the transformations in *aTransform*.

Availability

Available in Mac OS X v10.0 and later.

See Also

- prependTransform: (page 51)

Declared In

NSAffineTransform.h

initWithTransform:

Initializes the receiver's matrix using another transform object and returns the receiver.

```
- (id)initWithTransform:(NSAffineTransform *)aTransform
```

Parameters

aTransform

The transform object whose matrix values should be copied to this object.

Return Value

A new transform object initialized with the matrix values of *aTransform*.

Availability

Available in Mac OS X v10.0 and later.

See Also

+ transform (page 49)

Related Sample Code

DockTile

SpeedometerView

WebKitPluginStarter

WebKitPluginWithJavaScript

Declared In

NSAffineTransform.h

invert

Replaces the receiver's matrix with its inverse matrix.

```
- (void)invert
```

Discussion
Inverse matrices are useful for undoing the effects of a matrix. If a previous point (x,y) was transformed to (x',y'), inverting the matrix and applying it to point (x',y') yields the point (x,y).

You can also use inverse matrices in conjunction with the `concat` method to remove the effects of concatenating the matrix to the current transformation matrix of the current graphic context.

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

Related Sample Code
DockTile
SpeedometerView
WebKitPluginStarter
WebKitPluginWithJavaScript

Declared In
`NSAffineTransform.h`

prependTransform:

Prepends the specified matrix to the receiver's matrix.

```
- (void)prependTransform:(NSAffineTransform *)aTransform
```

Parameters

aTransform
The matrix to prepend to the receiver.

Discussion
This method multiplies the matrix in *aTransform* by the receiver's matrix and replaces the receiver's matrix with the result. This type of operation is the same as applying the transformations in *aTransform* followed by the transformations in the receiver.

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

See Also
- appendTransform: (page 49)

Declared In
`NSAffineTransform.h`

rotateByDegrees:

Applies a rotation factor (measured in degrees) to the receiver's transformation matrix.

```
- (void)rotateByDegrees:(CGFloat)angle
```

Parameters

angle
The rotation angle, measured in degrees.

Discussion
After invoking this method, applying the receiver's matrix turns the axes counterclockwise about the current origin by *angle* degrees, in addition to performing all previous transformations.

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

See Also
- `rotateByRadians:` (page 52)
- `scaleBy:` (page 53)
- `scaleXBy:yBy:` (page 53)
- `translateXBy:yBy:` (page 56)

Related Sample Code
DockTile
PDF Annotation Editor
SpeedometerView
WebKitPluginStarter
WebKitPluginWithJavaScript

Declared In
`NSAffineTransform.h`

rotateByRadians:

Applies a rotation factor (measured in radians) to the receiver's transformation matrix.

```
- (void)rotateByRadians:(CGFloat)angle
```

Parameters
angle
The rotation angle, measured in radians.

Discussion
After invoking this method, applying the receiver's matrix turns the axes counterclockwise about the current origin by *angle* radians, in addition to performing all previous transformations.

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

See Also
- `rotateByDegrees:` (page 51)
- `scaleBy:` (page 53)
- `scaleXBy:yBy:` (page 53)
- `translateXBy:yBy:` (page 56)

Related Sample Code
Polygons
TextLayoutDemo

Declared In
`NSAffineTransform.h`

scaleBy:

Applies the specified scaling factor along both x and y axes to the receiver's transformation matrix.

```
- (void)scaleBy:(CGFloat)scale
```

Parameters

scale

The scaling factor to apply to both axes. Specifying a negative value has the effect of inverting the direction of the axes in addition to scaling them. A scaling factor of 1.0 scales the content to exactly the same size.

Discussion

After invoking this method, applying the receiver's matrix modifies the unit lengths along the current x and y axes by a factor of *scale*, in addition to performing all previous transformations.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `rotateByDegrees:` (page 51)
- `rotateByRadians:` (page 52)
- `scaleXBy:yBy:` (page 53)
- `translateXBy:yBy:` (page 56)

Related Sample Code

Aperture Edit Plugin - Borders & Titles

CIAnnotation

Polygons

Transformed Image

Declared In

`NSAffineTransform.h`

scaleXBy:yBy:

Applies scaling factors to each axis of the receiver's transformation matrix.

```
- (void)scaleXBy:(CGFloat)scaleX yBy:(CGFloat)scaleY
```

Parameters

scaleX

The scaling factor to apply to the x axis.

scaleY

The scaling factor to apply to the y axis.

Discussion

After invoking this method, applying the receiver's matrix modifies the unit length on the x axis by a factor of *scaleX* and the y axis by a factor of *scaleY*, in addition to performing all previous transformations. A value of 1.0 for either axis scales the content on that axis to the same size.

Availability

Available in Mac OS X v10.0 and later.

See Also
- `rotateByDegrees:` (page 51)
- `rotateByRadians:` (page 52)
- `scaleBy:` (page 53)
- `translateXBy:yBy:` (page 56)

Related Sample Code
Sketch-112

Declared In
`NSAffineTransform.h`

setTransformStruct:

Replaces the receiver's transformation matrix with the specified values.

```
- (void)setTransformStruct:(NSAffineTransformStruct)aTransformStruct
```

Parameters

aTransformStruct

The structure containing the six transform values you want the receiver to use.

Discussion
The matrix is of the form shown in "Manipulating Transform Values", and the six-element structure defined by the `NSAffineTransformStruct` structure is of the form:

```
{m11, m12, m21, m22, tX, tY}
```

The `NSAffineTransformStruct` structure is an alternate representation of a transformation matrix that can be used to specify matrix values directly.

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

See Also
- `initWithTransform:` (page 50)
- `transformStruct` (page 55)

Related Sample Code
Transformed Image

Declared In
`NSAffineTransform.h`

transformPoint:

Applies the receiver's transform to the specified `NSPoint` data type and returns the results.

```
- (NSPoint)transformPoint:(NSPoint)aPoint
```

Parameters

aPoint

The point in the current coordinate system to which you want to apply the matrix.

Return Value

The resulting point after applying the receiver's transformations.

Availability

Available in Mac OS X v10.0 and later.

See Also

- transformSize: (page 55)

Declared In

NSAffineTransform.h

transformSize:

Applies the receiver's transform to the specified NSSize data type and returns the results.

```
- (NSSize)transformSize:(NSSize)aSize
```

Parameters

aSize

The size data to which you want to apply the matrix.

Return Value

The resulting size after applying the receiver's transformations.

Discussion

This method applies the current rotation and scaling factors to *aSize*; it does not apply translation factors. You can think of this method as transforming a vector whose origin is (0, 0) and whose end point is specified by the value in *aSize*. After the rotation and scaling factors are applied, this method effectively returns the end point of the new vector.

This method is useful for transforming delta or distance values when you need to take scaling and rotation factors into account.

Availability

Available in Mac OS X v10.0 and later.

See Also

- transformPoint: (page 54)

Declared In

NSAffineTransform.h

transformStruct

Returns the matrix coefficients stored in the receiver's matrix.

```
- (NSAffineTransformStruct)transformStruct
```

Return Value
The structure containing the receiver's six matrix values.

Discussion
The matrix is of the form shown in "Manipulating Transform Values", and the six-element structure defined by the `NSAffineTransformStruct` structure is of the form:

```
{m11, m12, m21, m22, tX, tY}
```

The `NSAffineTransformStruct` structure is an alternate representation of a transformation matrix that can be used to specify matrix values directly.

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

See Also
- `initWithTransform:` (page 50)
- `setTransformStruct:` (page 54)

Related Sample Code
Transformed Image

Declared In
`NSAffineTransform.h`

translateXBy:yBy:

Applies the specified translation factors to the receiver's transformation matrix.

```
- (void)translateXBy:(CGFloat)deltaX yBy:(CGFloat)deltaY
```

Parameters

deltaX
The number of units to move along the x axis.

deltaY
The number of units to move along the y axis.

Discussion
Subsequent transformations cause coordinates to be shifted by *deltaX* units along the x axis and by *deltaY* units along the y axis. Translation factors do not affect `NSSize` values, which specify a differential between points.

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

See Also
- `rotateByDegrees:` (page 51)
- `rotateByRadians:` (page 52)
- `scaleBy:` (page 53)
- `scaleXBy:yBy:` (page 53)

Related Sample Code
Cropped Image

PDF Annotation Editor
Sketch-112
WebKitPluginStarter
WebKitPluginWithJavaScript

Declared In
`NSAffineTransform.h`

Constants

NSAffineTransformStruct

This type defines the three-by-three matrix that performs an affine transform between two coordinate systems.

```
typedef struct _NSAffineTransformStruct {
    float m11, m12, m21, m22;
    float tX, tY;
} NSAffineTransformStruct;
```

Fields

`m11 , m12, m21, m22`
Elements of a two-by-two matrix for rotation, scale, and shear transformations.

`tX, tY`
x and y translation elements

Discussion
For more details, see *Cocoa Drawing Guide*.

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

Declared In
`NSAffineTransform.h`

NSAppleEventDescriptor Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSAppleEventDescriptor.h
Companion guide	Cocoa Scripting Guide
Related sample code	Apply Firmware Password AttachAScript CoreRecipes SimpleCarbonAppleScript Sketch-112

Overview

An instance of `NSAppleEventDescriptor` represents a descriptor—the basic building block for Apple events. This class is a wrapper for the underlying Apple event descriptor data type, `AEDesc`. Scriptable Cocoa applications frequently work with instances of `NSAppleEventDescriptor`, but should rarely need to work directly with the `AEDesc` data structure.

A *descriptor* is a data structure that stores data and an accompanying four-character code. A descriptor can store a value, or it can store a list of other descriptors (which may also be lists). All the information in an Apple event is stored in descriptors and lists of descriptors, and every Apple event is itself a descriptor list that matches certain criteria.

Important: An instance of `NSAppleEventDescriptor` can represent any kind of descriptor, from a simple value descriptor, to a descriptor list, to a full-fledged Apple event.

Descriptors can be used to build arbitrarily complex containers, so that one Apple event can represent a script statement such as `tell application "TextEdit" to get word 3 of paragraph 6 of document 3`.

In working with Apple event descriptors, it can be useful to understand some of the underlying data types. You'll find terms such as descriptor, descriptor list, Apple event record, and Apple event defined in Building an Apple Event in *Apple Events Programming Guide*. You'll also find information on the four-character codes

used to identify information within a descriptor. Apple event data types are defined in *Apple Event Manager Reference*. The values of many four-character codes used by Apple (and in some cases reused by developers) can be found in AppleScript Terminology and Apple Event Codes.

The most common reason to construct an Apple event with an instance of `NSAppleEventDescriptor` is to supply information in a return Apple event. The most common situation where you might need to extract information from an Apple event (as an instance of `NSAppleEventDescriptor`) is when an Apple event handler installed by your application is invoked, as described in "Installing an Apple Event Handler" in How Cocoa Applications Handle Apple Events. In addition, if you execute an AppleScript script using the `NSAppleScript` class, you get an instance of `NSAppleEventDescriptor` as the return value, from which you can extract any required information.

When you work with an instance of `NSAppleEventDescriptor`, you can access the underlying descriptor directly, if necessary, with the `aeDesc` (page 68) method. Other methods, including `descriptorWithDescriptorType:bytes:length:` (page 64) make it possible to create and initialize instances of `NSAppleEventDescriptor` without creating temporary instances of `NSData`.

The designated initializer for `NSAppleEventDescriptor` is `initWithAEDescNoCopy:` (page 73). However, it is unlikely that you will need to create a subclass of `NSAppleEventDescriptor`.

Cocoa doesn't currently provide a mechanism for applications to directly send raw Apple events (though compiling and executing an AppleScript script with `NSAppleScript` may result in Apple events being sent). However, Cocoa applications have full access to the Apple Event Manager C APIs for working with Apple events. So, for example, you might use an instance of `NSAppleEventDescriptor` to assemble an Apple event and call the Apple Event Manager function `AESend` to send it.

If you need to send Apple events, or if you need more information on some of the Apple event concepts described here, see *Apple Events Programming Guide* and *Apple Event Manager Reference*.

Adopted Protocols

NSCopying
- `copyWithZone:` (page 2042)

Tasks

Creating and Initializing Descriptors

`+ appleEventWithEventClass:eventID:targetDescriptor:returnID:transactionID:` (page 63)

Creates a descriptor that represents an Apple event, initialized according to the specified information.

`+ descriptorWithBoolean:` (page 64)

Creates a descriptor initialized with type `typeBoolean` that stores the specified Boolean value.

`+ descriptorWithDescriptorType:bytes:length:` (page 64)

Creates a descriptor initialized with the specified event type that stores the specified data (from a series of bytes).

+ descriptorWithDescriptorType:data: (page 65)
    Creates a descriptor initialized with the specified event type that stores the specified data (from an instance of NSData).
+ descriptorWithEnumCode: (page 65)
    Creates a descriptor initialized with type typeEnumerated that stores the specified enumerator data type value.
+ descriptorWithInt32: (page 66)
    Creates a descriptor initialized with Apple event type typeSInt32 that stores the specified integer value.
+ descriptorWithString: (page 66)
    Creates a descriptor initialized with type typeUnicodeText that stores the text from the specified string.
+ descriptorWithTypeCode: (page 67)
    Creates a descriptor initialized with type typeType that stores the specified type value.
+ listDescriptor (page 67)
    Creates and initializes an empty list descriptor.
+ nullDescriptor (page 67)
    Creates and initializes a descriptor with no parameter or attribute values set.
+ recordDescriptor (page 68)
    Creates and initializes a descriptor for an Apple event record whose data has yet to be set.
- initListDescriptor (page 72)
    Initializes a newly allocated instance as an empty list descriptor.
- initRecordDescriptor (page 73)
    Initializes a newly allocated instance as a descriptor that is an Apple event record.
- initWithAEDescNoCopy: (page 73)
    Initializes a newly allocated instance as a descriptor for the specified Carbon AEDesc structure.
- initWithDescriptorType:bytes:length: (page 74)
    Initializes a newly allocated instance as a descriptor with the specified descriptor type and data (from an arbitrary sequence of bytes and a length count).
- initWithDescriptorType:data: (page 74)
    Initializes a newly allocated instance as a descriptor with the specified descriptor type and data (from an instance of NSData).
- initWithEventClass:eventID:targetDescriptor:returnID:transactionID: (page 74)
    Initializes a newly allocated instance as a descriptor for an Apple event, initialized with the specified values.

Getting Information About a Descriptor

- aeDesc (page 68)
    Returns a pointer to the AEDesc structure that is encapsulated by the receiver, if it has one.
- booleanValue (page 69)
    Returns the contents of the receiver as a Boolean value, coercing (to typeBoolean) if necessary.
- coerceToDescriptorType: (page 69)
    Returns a descriptor obtained by coercing the receiver to the specified type.

- `data` (page 70)
  Returns the receiver's data as an `NSData` object.
- `descriptorType` (page 71)
  Returns the descriptor type of the receiver.
- `enumCodeValue` (page 71)
  Returns the contents of the receiver as an enumeration type, coercing (to `typeEnumerated`) if necessary.
- `int32Value` (page 76)
  Returns the contents of the receiver as an integer, coercing (to `typeSInt32`) if necessary.
- `numberOfItems` (page 77)
  Returns the number of descriptors in the receiver's descriptor list.
- `stringValue` (page 80)
  Returns the contents of the receiver as a Unicode text string, coercing (to `typeUnicodeText`) if necessary.
- `typeCodeValue` (page 81)
  Returns the contents of the receiver as a type, coercing (to `typeType`) if necessary.

Working With List Descriptors

- `descriptorAtIndex:` (page 70)
  Returns the descriptor at the specified (one-based) position in the receiving descriptor list.
- `insertDescriptor:atIndex:` (page 75)
  Inserts a descriptor at the specified (one-based) position in the receiving descriptor list, replacing the existing descriptor, if any, at that position.
- `removeDescriptorAtIndex:` (page 77)
  Removes the descriptor at the specified (one-based) position in the receiving descriptor list.

Working With Record Descriptors

- `descriptorForKeyword:` (page 70)
  Returns the receiver's descriptor for the specified keyword.
- `keywordForDescriptorAtIndex:` (page 76)
  Returns the keyword for the descriptor at the specified (one-based) position in the receiver.
- `removeDescriptorWithKeyword:` (page 78)
  Removes the receiver's descriptor identified by the specified keyword.
- `setDescriptor:forKeyword:` (page 79)
  Adds a descriptor, identified by a keyword, to the receiver.

Working With Apple Event Descriptors

- `attributeDescriptorForKeyword:` (page 69)
  Returns a descriptor for the receiver's Apple event attribute identified by the specified keyword.

- `eventClass` (page 71)
  Returns the event class for the receiver.
- `eventID` (page 72)
  Returns the event ID for the receiver.
- `paramDescriptorForKeyword:` (page 77)
  Returns a descriptor for the receiver's Apple event parameter identified by the specified keyword.
- `removeParamDescriptorWithKeyword:` (page 78)
  Removes the receiver's parameter descriptor identified by the specified keyword.
- `returnID` (page 79)
  Returns the receiver's return ID (the ID for a reply Apple event).
- `setAttributeDescriptor:forKeyword:` (page 79)
  Adds a descriptor to the receiver as an attribute identified by the specified keyword.
- `setParamDescriptor:forKeyword:` (page 80)
  Adds a descriptor to the receiver as an Apple event parameter identified by the specified keyword.
- `transactionID` (page 81)
  Returns the receiver's transaction ID, if any.

Class Methods

appleEventWithEventClass:eventID:targetDescriptor:returnID:transactionID:

Creates a descriptor that represents an Apple event, initialized according to the specified information.

```
+ (NSAppleEventDescriptor *)appleEventWithEventClass:(AEEventClass)eventClass
    eventID:(AEEventID)eventID targetDescriptor:(NSAppleEventDescriptor
    *)addressDescriptor returnID:(AEReturnID)returnID
    transactionID:(AETransactionID)transactionID
```

Parameters

eventClass
  The event class to be set in the returned descriptor.

eventID
  The event ID to be set in the returned descriptor.

addressDescriptor
  A pointer to a descriptor that identifies the target application for the Apple event. Passing `nil` results in an Apple event descriptor that has no `keyAddressAttr` attribute (it is valid for an Apple event to have no target address attribute).

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

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

Return Value
A descriptor for an Apple event, initialized according to the specified parameter values, or `nil` if an error occurs.

Discussion
Constants such as `kAutoGenerateReturnID` and `kAnyTransactionID` are defined in `AE.framework`, a subframework of `ApplicationServices.framework`.

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

Declared In
`NSAppleEventDescriptor.h`

descriptorWithBoolean:

Creates a descriptor initialized with type `typeBoolean` that stores the specified Boolean value.

```
+ (NSAppleEventDescriptor *)descriptorWithBoolean:(Boolean)boolean
```

Parameters
boolean
The Boolean value to be set in the returned descriptor.

Return Value
A descriptor with the specified Boolean value, or `nil` if an error occurs.

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

Declared In
`NSAppleEventDescriptor.h`

descriptorWithDescriptorType:bytes:length:

Creates a descriptor initialized with the specified event type that stores the specified data (from a series of bytes).

```
+ (NSAppleEventDescriptor *)descriptorWithDescriptorType:(DescType)descriptorType
    bytes:(const void *)bytes length:(NSUInteger)byteCount
```

Parameters
descriptorType
The descriptor type to be set in the returned descriptor.

bytes
The data, as a sequence of bytes, to be set in the returned descriptor.

byteCount
The length, in bytes, of the data to be set in the returned descriptor.

Return Value
A descriptor with the specified type and data, or `nil` if an error occurs.

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

Declared In
`NSAppleEventDescriptor.h`

descriptorWithDescriptorType:data:

Creates a descriptor initialized with the specified event type that stores the specified data (from an instance of `NSData`).

```
+ (NSAppleEventDescriptor *)descriptorWithDescriptorType:(DescType)descriptorType
    data:(NSData *)data
```

Parameters
descriptorType
The descriptor type to be set in the returned descriptor.

data
The data, as an instance of `NSData`, to be set in the returned descriptor.

Return Value
A descriptor with the specified type and data, or `nil` if an error occurs.

Discussion
You can use this method to create a descriptor that you can build into a complete Apple event by calling methods such as `setAttributeDescriptor:forKeyword:` (page 79), `setDescriptor:forKeyword:` (page 79), and `setParamDescriptor:forKeyword:` (page 80).

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

Declared In
`NSAppleEventDescriptor.h`

descriptorWithEnumCode:

Creates a descriptor initialized with type `typeEnumerated` that stores the specified enumerator data type value.

```
+ (NSAppleEventDescriptor *)descriptorWithEnumCode:(OSType)enumerator
```

Parameters
enumerator
A type code that identifies the type of enumerated data to be stored in the returned descriptor.

Return Value
A descriptor with the specified enumerator data type value, or `nil` if an error occurs.

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

Declared In
`NSAppleEventDescriptor.h`

descriptorWithInt32:

Creates a descriptor initialized with Apple event type `typeSInt32` that stores the specified integer value.

```
+ (NSAppleEventDescriptor *)descriptorWithInt32:(SInt32)signedInt
```

Parameters

signedInt
The integer value to be stored in the returned descriptor.

Return Value
A descriptor containing the specified integer value, or `nil` if an error occurs.

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

Related Sample Code
AttachAScript
SimpleCarbonAppleScript
Sketch-112

Declared In
`NSAppleEventDescriptor.h`

descriptorWithString:

Creates a descriptor initialized with type `typeUnicodeText` that stores the text from the specified string.

```
+ (NSAppleEventDescriptor *)descriptorWithString:(NSString *)string
```

Parameters

string
A string that specifies the text to be stored in the returned descriptor.

Return Value
A descriptor that contains the text from the specified string, or `nil` if an error occurs.

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

Related Sample Code
AttachAScript
SimpleCarbonAppleScript

Declared In
`NSAppleEventDescriptor.h`

descriptorWithTypeCode:

Creates a descriptor initialized with type `typeType` that stores the specified type value.

```
+ (NSAppleEventDescriptor *)descriptorWithTypeCode:(OSType)typeCode
```

Parameters

typeCode

The type value to be set in the returned descriptor.

Return Value
A descriptor with the specified type, or `nil` if an error occurs.

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

Declared In
`NSAppleEventDescriptor.h`

listDescriptor

Creates and initializes an empty list descriptor.

```
+ (NSAppleEventDescriptor *)listDescriptor
```

Return Value
An empty list descriptor, or `nil` if an error occurs.

Discussion
A list descriptor is a descriptor whose data consists of one or more descriptors. You can add items to the list by calling `insertDescriptor:atIndex:` (page 75) or remove them with `removeDescriptorAtIndex:` (page 77).

Invoking this method is equivalent to allocating an instance of `NSAppleEventDescriptor` and invoking `initListDescriptor` (page 72).

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

Related Sample Code
AttachAScript

Declared In
`NSAppleEventDescriptor.h`

nullDescriptor

Creates and initializes a descriptor with no parameter or attribute values set.

```
+ (NSAppleEventDescriptor *)nullDescriptor
```

Return Value
A descriptor with no parameter or attribute values set, or `nil` if an error occurs.

Discussion
You don't typically call this method, as most `NSAppleEventDescriptor` instance methods can't be safely called on the returned empty descriptor.

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

Declared In
`NSAppleEventDescriptor.h`

recordDescriptor

Creates and initializes a descriptor for an Apple event record whose data has yet to be set.

```
+ (NSAppleEventDescriptor *)recordDescriptor
```

Return Value
An Apple event descriptor whose data has yet to be set, or `nil` if an error occurs.

Discussion
An Apple event record is a descriptor whose data is a set of descriptors keyed by four-character codes. You can add information to the descriptor with methods such as `setAttributeDescriptor:forKeyword:` (page 79), `setDescriptor:forKeyword:` (page 79), and `setParamDescriptor:forKeyword:` (page 80).

Invoking this method is equivalent to allocating an instance of `NSAppleEventDescriptor` and invoking `initRecordDescriptor` (page 73).

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

Declared In
`NSAppleEventDescriptor.h`

Instance Methods

aeDesc

Returns a pointer to the `AEDesc` structure that is encapsulated by the receiver, if it has one.

```
- (const AEDesc *)aeDesc
```

Return Value
If the receiver has a valid `AEDesc` structure, returns a pointer to it; otherwise returns `nil`.

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

Declared In
`NSAppleEventDescriptor.h`

attributeDescriptorForKeyword:

Returns a descriptor for the receiver's Apple event attribute identified by the specified keyword.

```
- (NSAppleEventDescriptor *)attributeDescriptorForKeyword:(AEKeyword)keyword
```

Parameters

keyword

A keyword (a four-character code) that identifies the descriptor to obtain.

Return Value

The attribute descriptor for the specified keyword, or `nil` if an error occurs.

Discussion

The receiver must be an Apple event.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSAppleEventDescriptor.h`

booleanValue

Returns the contents of the receiver as a Boolean value, coercing (to `typeBoolean`) if necessary.

```
- (Boolean)booleanValue
```

Return Value

The contents of the descriptor, as a Boolean value, or `false` if an error occurs.

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

Apply Firmware Password

Declared In

`NSAppleEventDescriptor.h`

coerceToDescriptorType:

Returns a descriptor obtained by coercing the receiver to the specified type.

```
- (NSAppleEventDescriptor *)coerceToDescriptorType:(DescType)descriptorType
```

Parameters

descriptorType

The descriptor type to coerce the receiver to.

Return Value

A descriptor of the specified type, or `nil` if an error occurs.

Availability

Available in Mac OS X v10.0 and later.

Declared In
NSAppleEventDescriptor.h

data

Returns the receiver's data as an NSData object.

```
- (NSData *)data
```

Return Value
An instance of NSData containing the receiver's data, or nil if an error occurs.

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

Related Sample Code
Apply Firmware Password

Declared In
NSAppleEventDescriptor.h

descriptorAtIndex:

Returns the descriptor at the specified (one-based) position in the receiving descriptor list.

```
- (NSAppleEventDescriptor *)descriptorAtIndex:(NSInteger)anIndex
```

Parameters
anIndex
The one-based descriptor list position of the descriptor to return.

Return Value
The descriptor from the specified position (one-based) in the descriptor list, or nil if the specified descriptor cannot be obtained.

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

See Also
- insertDescriptor:atIndex: (page 75)
- removeDescriptorAtIndex: (page 77)

Related Sample Code
Apply Firmware Password
AttachAScript

Declared In
NSAppleEventDescriptor.h

descriptorForKeyword:

Returns the receiver's descriptor for the specified keyword.

```
- (NSAppleEventDescriptor *)descriptorForKeyword:(AEKeyword)keyword
```

Parameters

keyword

A keyword (a four-character code) that identifies the descriptor to obtain.

Return Value

A descriptor for the specified keyword, or `nil` if an error occurs.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSAppleEventDescriptor.h`

descriptorType

Returns the descriptor type of the receiver.

```
- (DescType)descriptorType
```

Return Value

The descriptor type of the receiver.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSAppleEventDescriptor.h`

enumCodeValue

Returns the contents of the receiver as an enumeration type, coercing (to `typeEnumerated`) if necessary.

```
- (OSType)enumCodeValue
```

Return Value

The contents of the descriptor, as an enumeration type, or 0 if an error occurs.

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

Apply Firmware Password

Declared In

`NSAppleEventDescriptor.h`

eventClass

Returns the event class for the receiver.

```
- (AEEventClass)eventClass
```

Return Value
The event class (a four-character code) for the receiver, or 0 if an error occurs.

Discussion
The receiver must be an Apple event. An Apple event is identified by its event class and event ID, a pair of four-character codes stored as 32-bit integers. For example, most events in the Standard suite have the four-character code `'core'` (defined as the constant `kAECoreSuite` in `AE.framework`, a subframework of `ApplicationServices.framework`). For more information on event classes and event IDs, see Building an Apple Event in *Apple Events Programming Guide*.

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

Declared In
`NSAppleEventDescriptor.h`

eventID

Returns the event ID for the receiver.

```
- (AEEventID)eventID
```

Return Value
The event ID (a four-character code) for the receiver, or 0 if an error occurs.

Discussion
The receiver must be an Apple event. An Apple event is identified by its event class and event ID, a pair of four-character codes stored as 32-bit integers. For example, the `open` Apple event from the Standard suite has the four-character code `'odoc'` (defined as the constant `kAEOpen` in `AE.framework`, a subframework of `ApplicationServices.framework`).

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

Declared In
`NSAppleEventDescriptor.h`

initListDescriptor

Initializes a newly allocated instance as an empty list descriptor.

```
- (id)initListDescriptor
```

Return Value
An empty list descriptor, or `nil` if an error occurs.

Discussion
You can add items to the empty list descriptor with `insertDescriptor:atIndex:` (page 75). The list indices are one-based.

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

See Also
`+ listDescriptor` (page 67)

Declared In
`NSAppleEventDescriptor.h`

initRecordDescriptor

Initializes a newly allocated instance as a descriptor that is an Apple event record.

```
- (id)initRecordDescriptor
```

Return Value
The initialized Apple event record, or `nil` if an error occurs.

Discussion
An Apple event record is a descriptor whose data is a set of descriptors keyed by four-character codes. You can add information to the descriptor with methods such as `setAttributeDescriptor:forKeyword:` (page 79), `setDescriptor:forKeyword:` (page 79), and `setParamDescriptor:forKeyword:` (page 80).

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

See Also
`+ recordDescriptor` (page 68)

Declared In
`NSAppleEventDescriptor.h`

initWithAEDescNoCopy:

Initializes a newly allocated instance as a descriptor for the specified Carbon `AEDesc` structure.

```
- (id)initWithAEDescNoCopy:(const AEDesc *)aeDesc
```

Parameters

aeDesc
A pointer to the `AEDesc` structure to associate with the descriptor.

Return Value
An instance of `NSAppleEventDescriptor` that is associated with the structure pointed to by *aeDesc*, or `nil` if an error occurs.

Discussion
The initialized object takes responsibility for calling the `AEDisposeDesc` function on the `AEDesc` at object deallocation time. This is the designated initializer for this class.

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

Declared In
`NSAppleEventDescriptor.h`

initWithDescriptorType:bytes:length:

Initializes a newly allocated instance as a descriptor with the specified descriptor type and data (from an arbitrary sequence of bytes and a length count).

```
- (id)initWithDescriptorType:(DescType)descriptorType bytes:(const void *)bytes
    length:(NSUInteger)byteCount
```

Parameters

descriptorType
The descriptor type to be set in the returned descriptor.

bytes
The data, as a sequence of bytes, to be set in the returned descriptor.

byteCount
The length, in bytes, of the data to be set in the returned descriptor.

Return Value
An instance of `NSAppleEventDescriptor` with the specified type and data. Returns `nil` if an error occurs.

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

Declared In
`NSAppleEventDescriptor.h`

initWithDescriptorType:data:

Initializes a newly allocated instance as a descriptor with the specified descriptor type and data (from an instance of `NSData`).

```
- (id)initWithDescriptorType:(DescType)descriptorType data:(NSData *)data
```

Parameters

descriptorType
The descriptor type to be set in the initialized descriptor.

data
The data to be set in the initialized descriptor.

Return Value
An instance of `NSAppleEventDescriptor` with the specified type and data. Returns `nil` if an error occurs.

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

See Also
`+ descriptorWithDescriptorType:data:` (page 65)

Declared In
`NSAppleEventDescriptor.h`

initWithEventClass:eventID:targetDescriptor:returnID:transactionID:

Initializes a newly allocated instance as a descriptor for an Apple event, initialized with the specified values.

```
- (id)initWithEventClass:(AEEventClass)eventClass eventID:(AEEventID)eventID
    targetDescriptor:(NSAppleEventDescriptor *)addressDescriptor
    returnID:(AEReturnID)returnID transactionID:(AETransactionID)transactionID
```

Parameters

eventClass

The event class to be set in the returned descriptor.

eventID

The event ID to be set in the returned descriptor.

addressDescriptor

A pointer to a descriptor that identifies the target application for the Apple event. Passing `nil` results in an Apple event descriptor that has no `keyAddressAttr` attribute (it is valid for an Apple event to have no target address attribute).

returnID

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

transactionID

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

Return Value

The initialized Apple event (an instance of `NSAppleEventDescriptor`), or `nil` if an error occurs.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSAppleEventDescriptor.h`

insertDescriptor:atIndex:

Inserts a descriptor at the specified (one-based) position in the receiving descriptor list, replacing the existing descriptor, if any, at that position.

```
- (void)insertDescriptor:(NSAppleEventDescriptor *)descriptor
    atIndex:(NSInteger)anIndex
```

Parameters

descriptor

The descriptor to insert in the receiver. Specifying an index of 0 or count + 1 causes appending to the end of the list.

anIndex

The one-based descriptor list position at which to insert the descriptor.

Discussion

Because it actually replaces the descriptor, if any, at the specified position, this method might better be called `replaceDescriptor:atIndex:`. The receiver must be a list descriptor. The indices are one-based. Currently provides no indication if an error occurs.

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

See Also
- `descriptorAtIndex:` (page 70)
- `removeDescriptorAtIndex:` (page 77)

Related Sample Code
AttachAScript

Declared In
`NSAppleEventDescriptor.h`

int32Value

Returns the contents of the receiver as an integer, coercing (to `typeSInt32`) if necessary.

```
- (SInt32)int32Value
```

Return Value
The contents of the descriptor, as an integer value, or 0 if an error occurs.

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

Related Sample Code
Apply Firmware Password
AttachAScript

Declared In
`NSAppleEventDescriptor.h`

keywordForDescriptorAtIndex:

Returns the keyword for the descriptor at the specified (one-based) position in the receiver.

```
- (AEKeyword)keywordForDescriptorAtIndex:(NSInteger)anIndex
```

Parameters
anIndex
The one-based descriptor list position of the descriptor to get the keyword for.

Return Value
The keyword (a four-character code) for the descriptor at the one-based location specified by *anIndex*, or 0 if an error occurs.

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

Declared In
`NSAppleEventDescriptor.h`

numberOfItems

Returns the number of descriptors in the receiver's descriptor list.

```
- (NSInteger)numberOfItems
```

Return Value
The number of descriptors in the receiver's descriptor list (possibly 0); returns 0 if an error occurs.

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

Related Sample Code
Apply Firmware Password

Declared In
`NSAppleEventDescriptor.h`

paramDescriptorForKeyword:

Returns a descriptor for the receiver's Apple event parameter identified by the specified keyword.

```
- (NSAppleEventDescriptor *)paramDescriptorForKeyword:(AEKeyword)keyword
```

Parameters
keyword
A keyword (a four-character code) that identifies the parameter descriptor to obtain.

Return Value
A descriptor for the specified keyword, or `nil` if an error occurs.

Discussion
The receiver must be an Apple event.

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

Related Sample Code
CoreRecipes

Declared In
`NSAppleEventDescriptor.h`

removeDescriptorAtIndex:

Removes the descriptor at the specified (one-based) position in the receiving descriptor list.

```
- (void)removeDescriptorAtIndex:(NSInteger)anIndex
```

Parameters
anIndex
The one-based position of the descriptor to remove.

Discussion
The receiver must be a list descriptor. The indices are one-based. Currently provides no indication if an error occurs.

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

See Also
- `descriptorAtIndex:` (page 70)
- `insertDescriptor:atIndex:` (page 75)

Declared In
`NSAppleEventDescriptor.h`

removeDescriptorWithKeyword:

Removes the receiver's descriptor identified by the specified keyword.

```
- (void)removeDescriptorWithKeyword:(AEKeyword)keyword
```

Parameters

keyword
A keyword (a four-character code) that identifies the descriptor to remove.

Discussion
The receiver must be an Apple event or Apple event record. Currently provides no indication if an error occurs.

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

Declared In
`NSAppleEventDescriptor.h`

removeParamDescriptorWithKeyword:

Removes the receiver's parameter descriptor identified by the specified keyword.

```
- (void)removeParamDescriptorWithKeyword:(AEKeyword)keyword
```

Parameters

keyword
A keyword (a four-character code) that identifies the parameter descriptor to remove. Currently provides no indication if an error occurs.

Discussion
The receiver must be an Apple event or Apple event record, both of which can contain parameters.

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

Declared In
`NSAppleEventDescriptor.h`

returnID

Returns the receiver's return ID (the ID for a reply Apple event).

```
- (AEReturnID)returnID
```

Return Value
The receiver's return ID (an integer value), or 0 if an error occurs.

Discussion
The receiver must be an Apple event.

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

Declared In
`NSAppleEventDescriptor.h`

setAttributeDescriptor:forKeyword:

Adds a descriptor to the receiver as an attribute identified by the specified keyword.

```
- (void)setAttributeDescriptor:(NSAppleEventDescriptor *)descriptor
    forKeyword:(AEKeyword)keyword
```

Parameters

descriptor
The attribute descriptor to add to the receiver.

keyword
A keyword (a four-character code) that identifies the attribute descriptor to add. If a descriptor with that keyword already exists in the receiver, it is replaced.

Discussion
The receiver must be an Apple event. Currently provides no indication if an error occurs.

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

Declared In
`NSAppleEventDescriptor.h`

setDescriptor:forKeyword:

Adds a descriptor, identified by a keyword, to the receiver.

```
- (void)setDescriptor:(NSAppleEventDescriptor *)descriptor
    forKeyword:(AEKeyword)keyword
```

Parameters

descriptor
The descriptor to add to the receiver.

keyword

A keyword (a four-character code) that identifies the descriptor to add. If a descriptor with that keyword already exists in the receiver, it is replaced.

Discussion
The receiver must be an Apple event or Apple event record. Currently provides no indication if an error occurs.

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

Related Sample Code
AttachAScript
SimpleCarbonAppleScript
Sketch-112

Declared In
`NSAppleEventDescriptor.h`

setParamDescriptor:forKeyword:

Adds a descriptor to the receiver as an Apple event parameter identified by the specified keyword.

```
- (void)setParamDescriptor:(NSAppleEventDescriptor *)descriptor
    forKeyword:(AEKeyword)keyword
```

Parameters

descriptor

The parameter descriptor to add to the receiver.

keyword

A keyword (a four-character code) that identifies the parameter descriptor to add. If a descriptor with that keyword already exists in the receiver, it is replaced.

Discussion
The receiver must be an Apple event or Apple event record, both of which can contain parameters.

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

Declared In
`NSAppleEventDescriptor.h`

stringValue

Returns the contents of the receiver as a Unicode text string, coercing (to `typeUnicodeText`) if necessary.

```
- (NSString *)stringValue
```

Return Value
The contents of the descriptor, as a string, or `nil` if an error occurs.

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

Related Sample Code
Apply Firmware Password
AttachAScript
CoreRecipes

Declared In
`NSAppleEventDescriptor.h`

transactionID

Returns the receiver's transaction ID, if any.

```
- (AETransactionID)transactionID
```

Return Value
The receiver's transaction ID (an integer value), or 0 if an error occurs.

Discussion
The receiver must be an Apple event. Currently provides no indication if an error occurs. For more information on transactions, see the description for `appleEventWithEventClass:eventID:targetDescriptor:returnID:transactionID:` (page 63).

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

Declared In
`NSAppleEventDescriptor.h`

typeCodeValue

Returns the contents of the receiver as a type, coercing (to `typeType`) if necessary.

```
- (OSType)typeCodeValue
```

Return Value
The contents of the descriptor, as a type, or 0 if an error occurs.

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

Related Sample Code
Apply Firmware Password

Declared In
`NSAppleEventDescriptor.h`

NSAppleEventManager Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSAppleEventManager.h
Companion guide	Cocoa Scripting Guide
Related sample code	CoreRecipes SimpleCarbonAppleScript Sketch-112

Overview

Provides a mechanism for registering handler routines for specific types of Apple events and dispatching events to those handlers.

Cocoa provides built-in scriptability support that uses scriptability information supplied by an application to automatically convert Apple events into script command objects that perform the desired operation. However, some applications may want to perform more basic Apple event handling, in which an application registers handlers for the Apple events it can process, then calls on the Apple Event Manager to dispatch received Apple events to the appropriate handler. `NSAppleEventManager` supports these mechanisms by providing methods to register and remove handlers and to dispatch Apple events to the appropriate handler, if one exists. For related information, see "How Cocoa Applications Handle Apple Events."

Each application has at most one instance of `NSAppleEventManager`. To obtain a reference to it, you call the class method `sharedAppleEventManager` (page 85), which creates the instance if it doesn't already exist.

For information about the Apple Event Manager, see *Apple Event Manager Reference* and *Apple Events Programming Guide*.

Tasks

Getting an Event Manager

+ `sharedAppleEventManager` (page 85)
  Returns the single instance of `NSAppleEventManager`, creating it first if it doesn't exist.

Working with Event Handlers

- `removeEventHandlerForEventClass:andEventID:` (page 87)
  If an Apple event handler has been registered for the event specified by *`eventClass`* and *`eventID`*, removes it.
- `setEventHandler:andSelector:forEventClass:andEventID:` (page 89)
  Registers the Apple event handler specified by *`handler`* for the event specified by *`eventClass`* and *`eventID`*.

Working with Events

- `dispatchRawAppleEvent:withRawReply:handlerRefCon:` (page 87)
  Causes the Apple event specified by *`theAppleEvent`* to be dispatched to the appropriate Apple event handler, if one has been registered by calling `setEventHandler:andSelector:forEventClass:andEventID:` (page 89).

Suspending and Resuming Apple Events

- `appleEventForSuspensionID:` (page 85)
  Given a nonzero *`suspensionID`* returned by an invocation of `suspendCurrentAppleEvent` (page 89), returns the descriptor for the event whose handling was suspended.
- `currentAppleEvent` (page 86)
  Returns the descriptor for *`currentAppleEvent`* if an Apple event is being handled on the current thread.
- `currentReplyAppleEvent` (page 86)
  Returns the corresponding reply event descriptor if an Apple event is being handled on the current thread.
- `replyAppleEventForSuspensionID:` (page 87)
  Given a nonzero *`suspensionID`* returned by an invocation of `suspendCurrentAppleEvent` (page 89), returns the corresponding reply event descriptor.
- `resumeWithSuspensionID:` (page 88)
  Given a nonzero *`suspensionID`* returned by an invocation of `suspendCurrentAppleEvent` (page 89), signal that handling of the suspended event may now continue.
- `setCurrentAppleEventAndReplyEventWithSuspensionID:` (page 88)
  Given a nonzero *`suspensionID`* returned by an invocation of `suspendCurrentAppleEvent` (page 89), sets the values that will be returned by subsequent invocations of `currentAppleEvent` (page

86) and `currentReplyAppleEvent` (page 86) to be the event whose handling was suspended and its corresponding reply event, respectively.

- `suspendCurrentAppleEvent` (page 89)
  Suspends the handling of the current event and returns an ID that must be used to resume the handling of the event if an Apple event is being handled on the current thread.

Class Methods

sharedAppleEventManager

Returns the single instance of `NSAppleEventManager`, creating it first if it doesn't exist.

```
+ (NSAppleEventManager *)sharedAppleEventManager
```

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

Related Sample Code
CoreRecipes
SimpleCarbonAppleScript
Sketch-112

Declared In
`NSAppleEventManager.h`

Instance Methods

appleEventForSuspensionID:

Given a nonzero *`suspensionID`* returned by an invocation of `suspendCurrentAppleEvent` (page 89), returns the descriptor for the event whose handling was suspended.

```
- (NSAppleEventDescriptor
    *)appleEventForSuspensionID:(NSAppleEventManagerSuspensionID)suspensionID
```

Discussion
The effects of mutating or retaining the returned descriptor are undefined, although it may be copied. `appleEventForSuspensionID:` may be invoked in any thread, not just the one in which the corresponding invocation of `suspendCurrentAppleEvent` occurred.

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

See Also
- `currentAppleEvent` (page 86)
- `currentReplyAppleEvent` (page 86)

Declared In
NSAppleEventManager.h

currentAppleEvent

Returns the descriptor for *currentAppleEvent* if an Apple event is being handled on the current thread.

```
- (NSAppleEventDescriptor *)currentAppleEvent
```

Discussion
An Apple event is being handled on the current thread if a handler that was registered with `setEventHandler:andSelector:forEventClass:andEventID:` (page 89) is being messaged at this instant or `setCurrentAppleEventAndReplyEventWithSuspensionID:` (page 88) has just been invoked. Returns `nil` otherwise. The effects of mutating or retaining the returned descriptor are undefined, although it may be copied.

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

See Also
- `currentReplyAppleEvent` (page 86)

Declared In
NSAppleEventManager.h

currentReplyAppleEvent

Returns the corresponding reply event descriptor if an Apple event is being handled on the current thread.

```
- (NSAppleEventDescriptor *)currentReplyAppleEvent
```

Discussion
An Apple event is being handled on the current thread if `currentAppleEvent` (page 86) does not return `nil`. Returns `nil` otherwise. This descriptor, including any mutations, will be returned to the sender of the current event when all handling of the event has been completed, if the sender has requested a reply. The effects of retaining the descriptor are undefined; it may be copied, but mutations of the copy are not returned to the sender of the current event.

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

See Also
- `setCurrentAppleEventAndReplyEventWithSuspensionID:` (page 88)

Related Sample Code
SimpleCarbonAppleScript
Sketch-112

Declared In
NSAppleEventManager.h

dispatchRawAppleEvent:withRawReply:handlerRefCon:

Causes the Apple event specified by *theAppleEvent* to be dispatched to the appropriate Apple event handler, if one has been registered by calling setEventHandler:andSelector:forEventClass:andEventID: (page 89).

```
- (OSErr)dispatchRawAppleEvent:(const AppleEvent *)theAppleEvent
    withRawReply:(AppleEvent *)theReply handlerRefCon:(UInt32)handlerRefcon
```

Discussion
The *theReply* parameter always specifies a reply Apple event, never nil. However, the handler should not fill out the reply if the descriptor type for the reply event is typeNull, indicating the sender does not want a reply.

The *handlerRefcon* parameter provides 4 bytes of data to the handler; a common use for this parameter is to pass a pointer to additional data.

This method is primarily intended for Cocoa's internal use. Note that *dispatching* an event means routing an event to an appropriate handler in the current application. You cannot use this method to *send* an event to other applications.

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

Declared In
NSAppleEventManager.h

removeEventHandlerForEventClass:andEventID:

If an Apple event handler has been registered for the event specified by *eventClass* and *eventID*, removes it.

```
- (void)removeEventHandlerForEventClass:(AEEventClass)eventClass
    andEventID:(AEEventID)eventID
```

Discussion
Otherwise does nothing.

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

See Also
- setEventHandler:andSelector:forEventClass:andEventID: (page 89)

Declared In
NSAppleEventManager.h

replyAppleEventForSuspensionID:

Given a nonzero *suspensionID* returned by an invocation of suspendCurrentAppleEvent (page 89), returns the corresponding reply event descriptor.

```
- (NSAppleEventDescriptor
    *)replyAppleEventForSuspensionID:(NSAppleEventManagerSuspensionID)suspensionID
```

Discussion
This descriptor, including any mutations, will be returned to the sender of the suspended event when handling of the event is resumed, if the sender has requested a reply. The effects of retaining the descriptor are undefined; it may be copied, but mutations of the copy are returned to the sender of the suspended event. `replyAppleEventForSuspensionID:` may be invoked in any thread, not just the one in which the corresponding invocation of `suspendCurrentAppleEvent` occurred.

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

See Also
- `appleEventForSuspensionID:` (page 85)
- `currentAppleEvent` (page 86)
- `currentReplyAppleEvent` (page 86)
- `setCurrentAppleEventAndReplyEventWithSuspensionID:` (page 88)

Declared In
`NSAppleEventManager.h`

resumeWithSuspensionID:

Given a nonzero *suspensionID* returned by an invocation of `suspendCurrentAppleEvent` (page 89), signal that handling of the suspended event may now continue.

```
- (void)resumeWithSuspensionID:(NSAppleEventManagerSuspensionID)suspensionID
```

Discussion
This may result in the immediate sending of the reply event to the sender of the suspended event, if the sender has requested a reply. If *suspensionID* has been used in a previous invocation of `setCurrentAppleEventAndReplyEventWithSuspensionID:` (page 88) the effects of that invocation are completely undone. Redundant invocations of `resumeWithSuspensionID:` are ignored. Subsequent invocations of other `NSAppleEventManager` methods using the same suspension ID are invalid. `resumeWithSuspensionID:` may be invoked in any thread, not just the one in which the corresponding invocation of `suspendCurrentAppleEvent` occurred.

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

Declared In
`NSAppleEventManager.h`

setCurrentAppleEventAndReplyEventWithSuspensionID:

Given a nonzero *suspensionID* returned by an invocation of `suspendCurrentAppleEvent` (page 89), sets the values that will be returned by subsequent invocations of `currentAppleEvent` (page 86) and `currentReplyAppleEvent` (page 86) to be the event whose handling was suspended and its corresponding reply event, respectively.

```
- (void)setCurrentAppleEventAndReplyEventWithSuspensionID:(NSAppleEventManagerSuspensionID)suspensionID
```

Discussion
Redundant invocations of `setCurrentAppleEventAndReplyEventWithSuspensionID:` are ignored.

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

Declared In
`NSAppleEventManager.h`

setEventHandler:andSelector:forEventClass:andEventID:

Registers the Apple event handler specified by *handler* for the event specified by *eventClass* and *eventID*.

```
- (void)setEventHandler:(id)handler andSelector:(SEL)handleEventSelector
    forEventClass:(AEEventClass)eventClass andEventID:(AEEventID)eventID
```

Discussion
If an event handler is already registered for the specified event class and event ID, removes it. The signature for *handler* should match the following:

```
- (void)handleAppleEvent:(NSAppleEventDescriptor *)event withReplyEvent:
(NSAppleEventDescriptor *)replyEvent;
```

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

See Also
- `removeEventHandlerForEventClass:andEventID:` (page 87)

Related Sample Code
CoreRecipes

Declared In
`NSAppleEventManager.h`

suspendCurrentAppleEvent

Suspends the handling of the current event and returns an ID that must be used to resume the handling of the event if an Apple event is being handled on the current thread.

```
- (NSAppleEventManagerSuspensionID)suspendCurrentAppleEvent
```

Discussion
An Apple event is being handled on the current thread if `currentAppleEvent` (page 86) does not return `nil`. Returns zero otherwise. The suspended event is no longer the current event after this method returns.

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

See Also
- `currentReplyAppleEvent` (page 86)
- `resumeWithSuspensionID:` (page 88)

Declared In
`NSAppleEventManager.h`

Constants

NSAppleEvent Timeouts

The following constants should not be used and may eventually be removed.

```
extern const double NSAppleEventTimeOutDefault;
extern const double NSAppleEventTimeOutNone;
```

Constants

`NSAppleEventTimeOutDefault`
Specifies that an event-processing operation should continue until a timeout occurs based on a value determined by the Apple Event Manager (about 1 minute). Not currently used by applications.

Available in Mac OS X v10.0 and later.

Declared in `NSAppleEventManager.h`.

`NSAppleEventTimeOutNone`
Specifies that the application is willing to wait indefinitely for the current operation to complete. Not currently used by applications.

Available in Mac OS X v10.0 and later.

Declared in `NSAppleEventManager.h`.

Declared In
`NSAppleEventManager.h`

Notifications

NSAppleEventManagerWillProcessFirstEventNotification

Posted by `NSAppleEventManager` before it first dispatches an Apple event. Your application can use this notification to avoid registering any Apple event handlers until the first time at which they may be needed. The notification object is the `NSAppleEventManager`. This notification does not contain a *`userInfo`* dictionary.

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

Declared In
`NSAppleEventManager.h`

NSAppleScript Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSAppleScript.h AppKit/NSAppleScriptExtensions.h
Availability	Available in Mac OS X v10.2 and later.
Companion guide	Cocoa Scripting Guide
Related sample code	AttachAScript

Overview

The `NSAppleScript` class provides the ability to load, compile, and execute scripts.

Important: You should access `NSAppleScript` only from the main thread.

This class provides applications with the ability to

- load a script from a URL or from a text string
- compile or execute a script or an individual Apple event
- obtain an `NSAppleEventDescriptor` containing the reply from an executed script or event
- obtain an attributed string for a compiled script, suitable for display in a script editor
- obtain various kinds of information about any errors that may occur

Important: `NSAppleScript` provides the `executeAppleEvent:error:` (page 94) method so that you can send an Apple event to invoke a handler in a script. (In an AppleScript script, a handler is the equivalent of a function.) However, you cannot use this method to send Apple events to other applications.

When you create an instance of `NSAppleScript` object, you can use a URL to specify a script that can be in either text or compiled form, or you can supply the script as a string. Should an error occur when compiling or executing the script, several of the methods return a dictionary containing error information. The keys for obtaining error information, such as `NSAppleScriptErrorMessage` (page 96), are described in the Constants section.

See also NSAppleScript Additions in the Application Kit framework, which defines a method that returns the syntax-highlighted source code for a script.

Adopted Protocols

NSCopying
- `copyWithZone:` (page 2042)

Tasks

Initializing a Script

- `initWithContentsOfURL:error:` (page 94)
  Initializes a newly allocated script instance from the source identified by the passed URL.
- `initWithSource:` (page 95)
  Initializes a newly allocated script instance from the passed source.

Getting Information About a Script

- `isCompiled` (page 95)
  Returns a Boolean value that indicates whether the receiver's script has been compiled.
- `source` (page 95)
  Returns the script source for the receiver.

Compiling and Executing a Script

- `compileAndReturnError:` (page 93)
  Compiles the receiver, if it is not already compiled.
- `executeAndReturnError:` (page 93)
  Executes the receiver, compiling it first if it is not already compiled.

`- executeAppleEvent:error:` (page 94)
Executes an Apple event in the context of the receiver, as a means of allowing the application to invoke a handler in the script.

Instance Methods

compileAndReturnError:

Compiles the receiver, if it is not already compiled.

```
- (BOOL)compileAndReturnError:(NSDictionary **)errorInfo
```

Parameters

errorInfo
On return, if an error occurs, a pointer to an error information dictionary.

Return Value
`YES` for success or if the script was already compiled, `NO` otherwise.

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

Declared In
`NSAppleScript.h`

executeAndReturnError:

Executes the receiver, compiling it first if it is not already compiled.

```
- (NSAppleEventDescriptor *)executeAndReturnError:(NSDictionary **)errorInfo
```

Parameters

errorInfo
On return, if an error occurs, a pointer to an error information dictionary.

Return Value
The result of executing the event, or `nil` if an error occurs.

Discussion
Any changes to property values caused by executing the script do not persist.

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

Declared In
`NSAppleScript.h`

executeAppleEvent:error:

Executes an Apple event in the context of the receiver, as a means of allowing the application to invoke a handler in the script.

```
- (NSAppleEventDescriptor *)executeAppleEvent:(NSAppleEventDescriptor *)event
    error:(NSDictionary **)errorInfo
```

Parameters

event

The Apple event to execute.

errorInfo

On return, if an error occurs, a pointer to an error information dictionary.

Return Value

The result of executing the event, or `nil` if an error occurs.

Discussion

Compiles the receiver before executing it if it is not already compiled.

Important: You cannot use this method to send Apple events to other applications.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`NSAppleScript.h`

initWithContentsOfURL:error:

Initializes a newly allocated script instance from the source identified by the passed URL.

```
- (id)initWithContentsOfURL:(NSURL *)url error:(NSDictionary **)errorInfo
```

Parameters

url

A URL that locates a script, in either text or compiled form.

errorInfo

On return, if an error occurs, a pointer to an error information dictionary.

Return Value

The initialized script object, `nil` if an error occurs.

Discussion

This method is a designated initializer for `NSAppleScript`.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`NSAppleScript.h`

initWithSource:

Initializes a newly allocated script instance from the passed source.

```
- (id)initWithSource:(NSString *)source
```

Parameters

source

A string containing the source code of a script.

Return Value

The initialized script object, `nil` if an error occurs.

Discussion

This method is a designated initializer for `NSAppleScript`.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`NSAppleScript.h`

isCompiled

Returns a Boolean value that indicates whether the receiver's script has been compiled.

```
- (BOOL)isCompiled
```

Return Value

`YES` if the receiver is already compiled, `NO` otherwise.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`NSAppleScript.h`

source

Returns the script source for the receiver.

```
- (NSString *)source
```

Return Value

The script source code of the receiver if it is available, `nil` otherwise.

Discussion

It is possible for an `NSAppleScript` that has been instantiated with `initWithContentsOfURL:error:` (page 94) to be a script for which the source code is not available but is nonetheless executable.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`NSAppleScript.h`

Constants

Error Dictionary Keys

If the result of `initWithContentsOfURL:error:` (page 94), `compileAndReturnError:` (page 93), `executeAndReturnError:` (page 93), or `executeAppleEvent:error:` (page 94), signals failure (`nil`, `NO`, `nil`, or `nil`, respectively), a pointer to an autoreleased dictionary is put at the location pointed to by the error parameter. The error info dictionary may contain entries that use any combination of the following keys, including no entries at all.

```
extern NSString *NSAppleScriptErrorMessage;
extern NSString *NSAppleScriptErrorNumber;
extern NSString *NSAppleScriptErrorAppName;
extern NSString *NSAppleScriptErrorBriefMessage;
extern NSString *NSAppleScriptErrorRange;
```

Constants

`NSAppleScriptErrorMessage`

An `NSString` that supplies a detailed description of the error condition.

Available in Mac OS X v10.2 and later.

Declared in `NSAppleScript.h`.

`NSAppleScriptErrorNumber`

An `NSNumber` that specifies the error number.

Available in Mac OS X v10.2 and later.

Declared in `NSAppleScript.h`.

`NSAppleScriptErrorAppName`

An `NSString` that specifies the name of the application that generated the error.

Available in Mac OS X v10.2 and later.

Declared in `NSAppleScript.h`.

`NSAppleScriptErrorBriefMessage`

An `NSString` that provides a brief description of the error.

Available in Mac OS X v10.2 and later.

Declared in `NSAppleScript.h`.

`NSAppleScriptErrorRange`

An `NSValue` that specifies a range.

Available in Mac OS X v10.2 and later.

Declared in `NSAppleScript.h`.

Declared In

`NSAppleScript.h`

NSArchiver Class Reference

Inherits from	NSCoder : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSArchiver.h
Companion guide	Archives and Serializations Programming Guide for Cocoa
Related sample code	Departments and Employees MenuItemView QTMetadataEditor Sketch-112 StickiesExample

Overview

`NSArchiver`, a concrete subclass of `NSCoder`, provides a way to encode objects into an architecture-independent format that can be stored in a file. When you archive a graph of objects, the class information and instance variables for each object are written to the archive. `NSArchiver`'s companion class, NSUnarchiver, decodes the data in an archive and creates a graph of objects equivalent to the original set.

`NSArchiver` stores the archive data in a mutable data object (`NSMutableData`). After encoding the objects, you can have the `NSArchiver` object write this mutable data object immediately to a file, or you can retrieve the mutable data object for some other use.

In Mac OS X v10.2 and later, `NSArchiver` and `NSUnarchiver` have been replaced by `NSKeyedArchiver` and `NSKeyedUnarchiver` respectively—see *Archives and Serializations Programming Guide for Cocoa*.

Tasks

Initializing an NSArchiver

`- initForWritingWithMutableData:` (page 102)
Returns an archiver, initialized to encode stream and version information into a given mutable data object.

Archiving Data

`+ archivedDataWithRootObject:` (page 98)
Returns a data object containing the encoded form of the object graph whose root object is given.

`+ archiveRootObject:toFile:` (page 99)
Creates a temporary instance of `NSArchiver` and archives an object graph by encoding it into a data object and writing the resulting data object to a specified file.

`- encodeRootObject:` (page 101)
Archives a given object along with all the objects to which it is connected.

`- encodeConditionalObject:` (page 101)
Conditionally archives a given object.

Getting the Archived Data

`- archiverData` (page 100)
Returns the receiver's archive data.

Substituting Classes or Objects

`- classNameEncodedForTrueClassName:` (page 100)
Returns the name of the class used to archive instances of the class with a given true name.

`- encodeClassName:intoClassName:` (page 100)
Encodes a substitute name for the class with a given true name.

`- replaceObject:withObject:` (page 102)
Causes the receiver to treat subsequent requests to encode a given object as though they were requests to encode another given object.

Class Methods

archivedDataWithRootObject:

Returns a data object containing the encoded form of the object graph whose root object is given.

```
+ (NSData *)archivedDataWithRootObject:(id)rootObject
```

Parameters

rootObject
The root object of the object graph to archive.

Return Value
A data object containing the encoded form of the object graph whose root object is *rootObject*.

Discussion
This method invokes `initForWritingWithMutableData:` (page 102) and `encodeRootObject:` (page 101) to create a temporary archiver that encodes the object graph.

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

See Also
- `initForWritingWithMutableData:` (page 102)
- `encodeRootObject:` (page 101)

Related Sample Code
Departments and Employees
MenuItemView
QTMetadataEditor
Sketch-112
StickiesExample

Declared In
`NSArchiver.h`

archiveRootObject:toFile:

Creates a temporary instance of `NSArchiver` and archives an object graph by encoding it into a data object and writing the resulting data object to a specified file.

```
+ (BOOL)archiveRootObject:(id)rootObject toFile:(NSString *)path
```

Parameters

rootObject
The root object of the object graph to archive.

path
The location of the the file into which to write the archive.

Return Value
`YES` if the archive was written successfully, otherwise `NO`.

Discussion
This convenience method invokes `archivedDataWithRootObject:` (page 98) to get the encoded data, and then sends that data object the message `writeToFile:atomically:` (page 384), using *path* for the first argument and `YES` for the second.

The archived data should be retrieved from the archive by an NSUnarchiver object.

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

See Also
- `archivedDataWithRootObject:` (page 98)
- `writeToFile:atomically:` (page 384) (`NSData`)

Declared In
`NSArchiver.h`

Instance Methods

archiverData

Returns the receiver's archive data.

```
- (NSMutableData *)archiverData
```

Return Value
The receiver's archive data.

Discussion
The returned data object is the same one specified as the argument to `initForWritingWithMutableData:` (page 102). It contains whatever data has been encoded thus far by invocations of the various encoding methods. It is safest not to invoke this method until after `encodeRootObject:` (page 101) has returned. In other words, although it is possible for a class to invoke this method from within its `encodeWithCoder:` (page 2034) method, that method must not alter the data.

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

Declared In
`NSArchiver.h`

classNameEncodedForTrueClassName:

Returns the name of the class used to archive instances of the class with a given true name.

```
- (NSString *)classNameEncodedForTrueClassName:(NSString *)trueName
```

Parameters

trueName
The real name of an encoded class.

Return Value
The name of the class used to archive instances of the class *trueName*.

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

See Also
`- encodeClassName:intoClassName:` (page 100)

Declared In
`NSArchiver.h`

encodeClassName:intoClassName:

Encodes a substitute name for the class with a given true name.

```
- (void)encodeClassName:(NSString *)trueName intoClassName:(NSString *)inArchiveName
```

Parameters

trueName

The real name of a class in the object graph being archived.

inArchiveName

The name of the class to use in the archive in place of *trueName*.

Discussion

Any subsequently encountered objects of class *trueName* are archived as instances of class *inArchiveName*. It is safest not to invoke this method during the archiving process (that is, within an `encodeWithCoder:` (page 2034) method). Instead, invoke it before `encodeRootObject:` (page 101).

Availability

Available in Mac OS X v10.0 and later.

See Also

`- classNameEncodedForTrueClassName:` (page 100)

Declared In

`NSArchiver.h`

encodeConditionalObject:

Conditionally archives a given object.

```
- (void)encodeConditionalObject:(id)object
```

Parameters

object

The object to archive.

Discussion

This method overrides the superclass implementation to allow *object* to be encoded only if it is also encoded unconditionally by another object in the object graph. Conditional encoding lets you encode one part of a graph detached from the rest. (See *Archives and Serializations Programming Guide for Cocoa* for more information.)

This method should be invoked only from within an `encodeWithCoder:` (page 2034) method. If *object* is `nil`, the `NSArchiver` object encodes it unconditionally as `nil`. This method raises an `NSInvalidArgumentException` if no root object has been encoded.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSArchiver.h`

encodeRootObject:

Archives a given object along with all the objects to which it is connected.

```
- (void)encodeRootObject:(id)rootObject
```

Parameters

rootObject

The root object of the object graph to archive.

Discussion

If any object is encountered more than once while traversing the graph, it is encoded only once, but the multiple references to it are stored. (See *Archives and Serializations Programming Guide for Cocoa* for more information.)

This message must not be sent more than once to a given `NSArchiver` object; an `NSInvalidArgumentException` is raised if a root object has already been encoded. If you need to encode multiple object graphs, therefore, don't attempt to reuse an `NSArchiver` instance; instead, create a new one for each graph.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSArchiver.h`

initForWritingWithMutableData:

Returns an archiver, initialized to encode stream and version information into a given mutable data object.

```
- (id)initForWritingWithMutableData:(NSMutableData *)data
```

Parameters

data

The mutable data object into which to write the archive. This value must not be `nil`.

Return Value

An archiver object, initialized to encode stream and version information into *data*.

Discussion

Raises an `NSInvalidArgumentException` if *data* is `nil`.

Availability

Available in Mac OS X v10.0 and later.

See Also

- archiverData (page 100)

Declared In

`NSArchiver.h`

replaceObject:withObject:

Causes the receiver to treat subsequent requests to encode a given object as though they were requests to encode another given object.

```
- (void)replaceObject:(id)object withObject:(id)newObject
```

Parameters

object

An object in the object graph being archived.

newObject

The object with which to replace *object* in the archive.

Discussion

Both *object* and *newObject* must be valid objects.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSArchiver.h

Constants

Archiving Exception Names

Raised by NSArchiver if there are problems initializing or encoding.

```
extern NSString *NSInconsistentArchiveException;
```

Constants

NSInconsistentArchiveException

The name of an exception raised by NSArchiver if there are problems initializing or encoding.

Available in Mac OS X v10.0 and later.

Declared in NSArchiver.h.

Declared In

NSArchiver.h

NSArray Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSMutableCopying NSFastEnumeration NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSArray.h Foundation/NSKeyValueCoding.h Foundation/NSKeyValueObserving.h Foundation/NSPathUtilities.h Foundation/NSPredicate.h Foundation/NSSortDescriptor.h
Companion guides	Collections Programming Topics for Cocoa Key-Value Coding Programming Guide Property List Programming Guide Predicate Programming Guide
Related sample code	CoreRecipes iSpend Quartz Composer WWDC 2005 TextEdit Sketch-112 StickiesExample

Overview

`NSArray` and its subclass `NSMutableArray` manage collections of objects called **arrays**. `NSArray` creates static arrays, and `NSMutableArray` creates dynamic arrays.

The `NSArray` and `NSMutableArray` classes adopt the `NSCopying` and `NSMutableCopying` protocols, making it convenient to convert an array of one type to the other.

`NSArray` and `NSMutableArray` are part of a class cluster, so arrays are not actual instances of the `NSArray` or `NSMutableArray` classes but of one of their private subclasses. Although an array's class is private, its interface is public, as declared by these abstract superclasses, `NSArray` and `NSMutableArray`.

NSArray's two primitive methods—count (page 119) and objectAtIndex: (page 131)—provide the basis for all other methods in its interface. The count method returns the number of elements in the array; objectAtIndex: gives you access to the array elements by index, with index values starting at 0.

The methods objectEnumerator (page 131) and reverseObjectEnumerator (page 134) also grant sequential access to the elements of the array, differing only in the direction of travel through the elements. These methods are provided so that arrays can be traversed in a manner similar to that used for objects of other collection classes, such as NSDictionary. See the objectEnumerator method description for a code excerpt that shows how to use these methods to access the elements of an array. In Mac OS X v10.5 and later, it is more efficient to use the fast enumeration protocol (see NSFastEnumeration).

NSArray provides methods for querying the elements of the array. The indexOfObject: (page 123) method searches the array for the object that matches its argument. To determine whether the search is successful, each element of the array is sent an isEqual: (page 2101) message, as declared in the NSObject protocol. Another method, indexOfObjectIdenticalTo: (page 124), is provided for the less common case of determining whether a specific object is present in the array. The indexOfObjectIdenticalTo: method tests each element in the array to see whether its id matches that of the argument.

NSArray's filteredArrayUsingPredicate: (page 121) method allows you to create a new array from an existing array filtered using a predicate (see *Predicate Programming Guide*).

NSArray's makeObjectsPerformSelector: (page 129) and makeObjectsPerformSelector:withObject: (page 130) methods let you send messages to all objects in the array. To act on the array as a whole, a variety of other methods are defined. You can create a sorted version of the array (sortedArrayUsingSelector: (page 138) and sortedArrayUsingFunction:context: (page 136), extract a subset of the array (subarrayWithRange: (page 138)), or concatenate the elements of an array of NSString objects into a single string (componentsJoinedByString: (page 118)). In addition, you can compare two arrays using the isEqualToArray: (page 129) and firstObjectCommonWithArray: (page 122) methods. Finally, you can create new arrays that contain the objects in an existing array and one or more additional objects with arrayByAddingObject: (page 117) and arrayByAddingObjectsFromArray: (page 117).

Arrays maintain strong references to their contents—in a managed memory environment, each object receives a retain message before its id is added to the array and a release message when it is removed from the array or when the array is deallocated. If you want a collection with different object ownership semantics, consider using *CFArray Reference*, NSPointerArray, or NSHashTable instead.

NSArray is "toll-free bridged" with its Core Foundation counterpart, *CFArray Reference*. What this means is that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object, providing you cast one type to the other. Therefore, in an API where you see an NSArray * parameter, you can pass in a CFArrayRef, and in an API where you see a CFArrayRef parameter, you can pass in an NSArray instance. This arrangement also applies to your concrete subclasses of NSArray. See *Carbon-Cocoa Integration Guide* for more information on toll-free bridging.

Subclassing Notes

Most developers would not have any reason to subclass NSArray. The class does well what it is designed to do—maintain an ordered collection of objects. But there are situations where a custom NSArray object might come in handy. Here are a few possibilities:

- Changing how NSArray stores the elements of its collection. You might do this for performance reasons or for better compatibility with legacy code.

- Changing how NSArray retains and releases its elements.
- Acquiring more information about what is happening to the collection (for example, statistics gathering).

Methods to Override

Any subclass of NSArray *must* override the primitive instance methods count (page 119) and objectAtIndex: (page 131). These methods must operate on the backing store that you provide for the elements of the collection. For this backing store you can use a static array, a standard NSArray object, or some other data type or mechanism. You may also choose to override, partially or fully, any other NSArray method for which you want to provide an alternative implementation.

You might want to implement an initializer for your subclass that is suited to the backing store that the subclass is managing. The NSArray class does not have a designated initializer, so your initializer need only invoke the init (page 1178) method of super. The NSArray class adopts the NSCopying, NSMutableCopying, and NSCoding protocols; if you want instances of your own custom subclass created from copying or coding, override the methods in these protocols.

Remember that NSArray is the public interface for a class cluster and what this entails for your subclass. The primitive methods of NSArray do not include any designated initializers. This means that you must provide the storage for your subclass and implement the primitive methods that directly act on that storage.

Special Considerations

In most cases your custom NSArray class should conform to Cocoa's object-ownership conventions. Thus you must send retain (page 2108) to each object that you add to your collection and release (page 2106) to each object that you remove from the collection. Of course, if the reason for subclassing NSArray is to implement object-retention behavior different from the norm (for example, a non-retaining array), then you can ignore this requirement.

Alternatives to Subclassing

Before making a custom class of NSArray, investigate NSPointerArray, NSHashTable, and the corresponding Core Foundation type, *CFArray Reference*. Because NSArray and CFArray are "toll-free bridged," you can substitute a CFArray object for a NSArray object in your code (with appropriate casting). Although they are corresponding types, CFArray and NSArray do not have identical interfaces or implementations, and you can sometimes do things with CFArray that you cannot easily do with NSArray. For example, CFArray provides a set of callbacks, some of which are for implementing custom retain-release behavior. If you specify NULL implementations for these callbacks, you can easily get a non-retaining array.

If the behavior you want to add supplements that of the existing class, you could write a category on NSArray. Keep in mind, however, that this category will be in effect for all instances of NSArray that you use, and this might have unintended consequences.

Adopted Protocols

NSCoding
- encodeWithCoder: (page 2034)
- initWithCoder: (page 2034)

NSCopying
- copyWithZone: (page 2042)

NSMutableCopying
- mutableCopyWithZone: (page 2094)

Tasks

Creating an Array

+ array (page 111)
Creates and returns an empty array.

+ arrayWithArray: (page 112)
Creates and returns an array containing the objects in another given array.

+ arrayWithContentsOfFile: (page 113)
Creates and returns an array containing the contents of the file specified by a given path.

+ arrayWithContentsOfURL: (page 113)
Creates and returns an array containing the contents specified by a given URL.

+ arrayWithObject: (page 114)
Creates and returns an array containing a given object.

+ arrayWithObjects: (page 114)
Creates and returns an array containing the objects in the argument list.

+ arrayWithObjects:count: (page 115)
Creates and returns an array that includes a given number of objects from a given C array.

Initializing an Array

- initWithArray: (page 125)
Initializes a newly allocated array by placing in it the objects contained in a given array.

- initWithArray:copyItems: (page 126)
Initializes a newly allocated array using *anArray* as the source of data objects for the array.

- initWithContentsOfFile: (page 126)
Initializes a newly allocated array with the contents of the file specified by a given path.

- initWithContentsOfURL: (page 127)
Initializes a newly allocated array with the contents of the location specified by a given URL.

- initWithObjects: (page 127)
Initializes a newly allocated array by placing in it the objects in the argument list.

- initWithObjects:count: (page 128)
Initializes a newly allocated array to include a given number of objects from a given C array.

Querying an Array

- `containsObject:` (page 119)
  Returns a Boolean value that indicates whether a given object is present in the receiver.
- `count` (page 119)
  Returns the number of objects currently in the receiver.
- `getObjects:` (page 122)
  Copies all the objects contained in the receiver to *aBuffer*.
- `getObjects:range:` (page 123)
  Copies the objects contained in the receiver that fall within the specified range to *aBuffer*.
- `indexOfObject:` (page 123)
  Returns the lowest index whose corresponding array value is equal to a given object.
- `indexOfObject:inRange:` (page 123)
  Returns the lowest index within a specified range whose corresponding array value is equal to a given object .
- `indexOfObjectIdenticalTo:` (page 124)
  Returns the lowest index whose corresponding array value is identical to a given object.
- `indexOfObjectIdenticalTo:inRange:` (page 125)
  Returns the lowest index within a specified range whose corresponding array value is equal to a given object .
- `lastObject` (page 129)
  Returns the object in the array with the highest index value.
- `objectAtIndex:` (page 131)
  Returns the object located at *index*.
- `objectsAtIndexes:` (page 132)
  Returns an array containing the objects in the receiver at the indexes specified by a given index set.
- `objectEnumerator` (page 131)
  Returns an enumerator object that lets you access each object in the receiver.
- `reverseObjectEnumerator` (page 134)
  Returns an enumerator object that lets you access each object in the receiver, in reverse order.

Sending Messages to Elements

- `makeObjectsPerformSelector:` (page 129)
  Sends to each object in the receiver the message identified by a given selector, starting with the first object and continuing through the array to the last object.
- `makeObjectsPerformSelector:withObject:` (page 130)
  Sends the *aSelector* message to each object in the array, starting with the first object and continuing through the array to the last object.

Comparing Arrays

- `firstObjectCommonWithArray:` (page 122)
  Returns the first object contained in the receiver that's equal to an object in another given array.

- `isEqualToArray:` (page 129)
  Compares the receiving array to another array.

Deriving New Arrays

- `arrayByAddingObject:` (page 117)
  Returns a new array that is a copy of the receiver with a given object added to the end.
- `arrayByAddingObjectsFromArray:` (page 117)
  Returns a new array that is a copy of the receiver with the objects contained in another array added to the end.
- `filteredArrayUsingPredicate:` (page 121)
  Evaluates a given predicate against each object in the receiver and returns a new array containing the objects for which the predicate returns true.
- `subarrayWithRange:` (page 138)
  Returns a new array containing the receiver's elements that fall within the limits specified by a given range.

Sorting

- `sortedArrayHint` (page 135)
  Analyzes the receiver and returns a "hint" that speeds the sorting of the array when the hint is supplied to `sortedArrayUsingFunction:context:hint:` (page 137).
- `sortedArrayUsingFunction:context:` (page 136)
  Returns a new array that lists the receiver's elements in ascending order as defined by the comparison function *`comparator`*.
- `sortedArrayUsingFunction:context:hint:` (page 137)
  Returns a new array that lists the receiver's elements in ascending order as defined by the comparison function *`comparator`*.
- `sortedArrayUsingDescriptors:` (page 135)
  Returns a copy of the receiver sorted as specified by a given array of sort descriptors.
- `sortedArrayUsingSelector:` (page 138)
  Returns an array that lists the receiver's elements in ascending order, as determined by the comparison method specified by a given selector.

Working with String Elements

- `componentsJoinedByString:` (page 118)
  Constructs and returns an `NSString` object that is the result of interposing a given separator between the elements of the receiver's array.

Creating a Description

- `description` (page 120)
  Returns a string that represents the contents of the receiver, formatted as a property list.

- `descriptionWithLocale:` (page 120)
  Returns a string that represents the contents of the receiver, formatted as a property list.
- `descriptionWithLocale:indent:` (page 121)
  Returns a string that represents the contents of the receiver, formatted as a property list.
- `writeToFile:atomically:` (page 139)
  Writes the contents of the receiver to a file at a given path.
- `writeToURL:atomically:` (page 140)
  Writes the contents of the receiver to the location specified by a given URL.

Collecting Paths

- `pathsMatchingExtensions:` (page 133)
  Returns an array containing all the pathname elements in the receiver that have filename extensions from a given array.

Key-Value Observing

- `addObserver:forKeyPath:options:context:` (page 116)
  Raises an exception.
- `removeObserver:forKeyPath:` (page 133)
  Raises an exception.
- `addObserver:toObjectsAtIndexes:forKeyPath:options:context:` (page 116)
  Registers *`anObserver`* to receive key value observer notifications for the specified *`keypath`* relative to the objects at *`indexes`*.
- `removeObserver:fromObjectsAtIndexes:forKeyPath:` (page 134)
  Removes *`anObserver`* from all key value observer notifications associated with the specified *`keyPath`* relative to the receiver's objects at *`indexes`*.

Key-Value Coding

- `setValue:forKey:` (page 135)
  Invokes `setValue:forKey:` on each of the receiver's items using the specified *`value`* and *`key`*.
- `valueForKey:` (page 139)
  Returns an array containing the results of invoking `valueForKey:` using *`key`* on each of the receiver's objects.

Class Methods

array

Creates and returns an empty array.

```
+ (id)array
```

Return Value
An empty array.

Discussion
This method is used by mutable subclasses of `NSArray`.

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

See Also
`+ arrayWithObject:` (page 114)
`+ arrayWithObjects:` (page 114)

Related Sample Code
CoreRecipes
Dicey
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus

Declared In
`NSArray.h`

arrayWithArray:

Creates and returns an array containing the objects in another given array.

```
+ (id)arrayWithArray:(NSArray *)anArray
```

Parameters
anArray
An array.

Return Value
An array containing the objects in *anArray*.

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

See Also
`+ arrayWithObjects:` (page 114)
`- initWithObjects:` (page 127)

Related Sample Code
CoreRecipes
iSpend
QTKitMovieShuffler
Reminders
Squiggles

Declared In
`NSArray.h`

arrayWithContentsOfFile:

Creates and returns an array containing the contents of the file specified by a given path.

```
+ (id)arrayWithContentsOfFile:(NSString *)aPath
```

Parameters

aPath

The path to a file containing a string representation of an array produced by the `writeToFile:atomically:` (page 139) method.

Return Value

An array containing the contents of the file specified by *aPath*. Returns `nil` if the file can't be opened or if the contents of the file can't be parsed into an array.

Discussion

The array representation in the file identified by *aPath* must contain only property list objects (`NSString`, `NSData`, `NSArray`, or `NSDictionary` objects).

Availability

Available in Mac OS X v10.0 and later.

See Also

`- writeToFile:atomically:` (page 139)

Related Sample Code

LSMSmartCategorizer

Mountains

URL CacheInfo

Declared In

`NSArray.h`

arrayWithContentsOfURL:

Creates and returns an array containing the contents specified by a given URL.

```
+ (id)arrayWithContentsOfURL:(NSURL *)aURL
```

Parameters

aURL

The location of a file containing a string representation of an array produced by the `writeToURL:atomically:` (page 140) method.

Return Value

An array containing the contents specified by *aURL*. Returns `nil` if the location can't be opened or if the contents of the location can't be parsed into an array.

Discussion

The array representation at the location identified by *aURL* must contain only property list objects (`NSString`, `NSData`, `NSArray`, or `NSDictionary` objects).

Availability

Available in Mac OS X v10.0 and later.

See Also
- writeToURL:atomically: (page 140)

Declared In
NSArray.h

arrayWithObject:

Creates and returns an array containing a given object.

```
+ (id)arrayWithObject:(id)anObject
```

Parameters

anObject
An object.

Return Value
An array containing the single element *anObject*.

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

See Also
+ array (page 111)
+ arrayWithObjects: (page 114)

Related Sample Code
CoreRecipes
Dicey
Quartz Composer WWDC 2005 TextEdit
StickiesExample
TextEditPlus

Declared In
NSArray.h

arrayWithObjects:

Creates and returns an array containing the objects in the argument list.

```
+ (id)arrayWithObjects:(id)firstObj, ...
```

Parameters

firstObj, ...
A comma-separated list of objects ending with nil.

Return Value
An array containing the objects in the argument list.

Discussion
This code example creates an array containing three different types of element:

```
NSArray *myArray;
```

```
NSDate *aDate = [NSDate distantFuture];
NSValue *aValue = [NSNumber numberWithInt:5];
NSString *aString = @"a string";

myArray = [NSArray arrayWithObjects:aDate, aValue, aString, nil];
```

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

See Also
`+ array` (page 111)
`+ arrayWithObject:` (page 114)

Related Sample Code
CoreRecipes
iSpend
QTCoreVideo301
Sketch-112
TimelineToTC

Declared In
`NSArray.h`

arrayWithObjects:count:

Creates and returns an array that includes a given number of objects from a given C array.

```
+ (id)arrayWithObjects:(const id *)objects count:(NSUInteger)count
```

Parameters

objects
A C array of objects.

count
The number of values from the *objects* C array to include in the new array. This number will be the count of the new array—it must not be negative or greater than the number of elements in *objects*.

Return Value
A new array including the first *count* objects from *objects*.

Discussion
Elements are added to the new array in the same order they appear in *objects*, up to but not including index *count*.

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

See Also
`- getObjects:` (page 122)
`- getObjects:range:` (page 123)

Declared In
`NSArray.h`

Instance Methods

addObserver:forKeyPath:options:context:

Raises an exception.

```
- (void)addObserver:(NSObject *)observer forKeyPath:(NSString *)keyPath
    options:(NSKeyValueObservingOptions)options context:(void *)context
```

Parameters

observer

The object to register for KVO notifications. The observer must implement the key-value observing method `observeValueForKeyPath:ofObject:change:context:` (page 2081).

keyPath

The key path, relative to the receiver, of the property to observe. This value must not be `nil`.

options

A combination of the `NSKeyValueObservingOptions` (page 2086) values that specifies what is included in observation notifications. For possible values, see `NSKeyValueObservingOptions`.

context

Arbitrary data that is passed to *observer* in `observeValueForKeyPath:ofObject:change:context:` (page 2081).

Special Considerations

`NSArray` objects are not observable, so this method raises an exception when invoked on an `NSArray` object. Instead of observing an array, observe the to-many relationship for which the array is the collection of related objects.

Availability

Available in Mac OS X v10.4 and later.

See Also

- `removeObserver:forKeyPath:` (page 133)
- `addObserver:toObjectsAtIndexes:forKeyPath:options:context:` (page 116)

Declared In

`NSKeyValueObserving.h`

addObserver:toObjectsAtIndexes:forKeyPath:options:context:

Registers *anObserver* to receive key value observer notifications for the specified *keypath* relative to the objects at *indexes*.

```
- (void)addObserver:(NSObject *)anObserver toObjectsAtIndexes:(NSIndexSet *)indexes
    forKeyPath:(NSString *)keyPath options:(NSKeyValueObservingOptions)options
   context:(void *)context
```

Discussion

The *options* determine what is included in the notifications, and the *context* is passed in the notifications.

This is not merely a convenience method; invoking this method is potentially much faster than repeatedly invoking `addObserver:forKeyPath:options:context:` (page 2079).

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

See Also
`- removeObserver:fromObjectsAtIndexes:forKeyPath:` (page 134)

Related Sample Code
iSpend

Declared In
`NSKeyValueObserving.h`

arrayByAddingObject:

Returns a new array that is a copy of the receiver with a given object added to the end.

```
- (NSArray *)arrayByAddingObject:(id)anObject
```

Parameters
anObject
An object.

Return Value
A new array that is a copy of the receiver with *anObject* added to the end.

Discussion
If *anObject* is `nil`, an `NSInvalidArgumentException` is raised.

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

See Also
`- addObject:` (page 911) (`NSMutableArray`)

Related Sample Code
UIElementInspector

Declared In
`NSArray.h`

arrayByAddingObjectsFromArray:

Returns a new array that is a copy of the receiver with the objects contained in another array added to the end.

```
- (NSArray *)arrayByAddingObjectsFromArray:(NSArray *)otherArray
```

Parameters
otherArray
An array.

Return Value
A new array that is a copy of the receiver with the objects contained in *otherArray* added to the end.

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

See Also
- addObjectsFromArray: (page 911) (NSMutableArray)

Related Sample Code
QTRecorder

Declared In
NSArray.h

componentsJoinedByString:

Constructs and returns an NSString object that is the result of interposing a given separator between the elements of the receiver's array.

```
- (NSString *)componentsJoinedByString:(NSString *)separator
```

Parameters

separator
The string to interpose between the elements of the receiver's array.

Return Value
An NSString object that is the result of interposing *separator* between the elements of the receiver's array. If the receiver has no elements, returns an NSString object representing an empty string.

Discussion
For example, this code excerpt writes "here be dragons" to the console:

```
NSArray *pathArray = [NSArray arrayWithObjects:@"here",
    @"be", @"dragons", nil];
NSLog(@"%@",
    [pathArray componentsJoinedByString:@" "]);
```

Special Considerations

Each element in the receiver's array must handle description.

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

See Also
- componentsSeparatedByString: (page 1547) (NSString)

Related Sample Code
Aperture Edit Plugin - Borders & Titles
AttachAScript
CoreRecipes
Sproing
TipWrapper

Declared In
NSArray.h

containsObject:

Returns a Boolean value that indicates whether a given object is present in the receiver.

```
- (BOOL)containsObject:(id)anObject
```

Parameters

anObject
An object.

Return Value
YES if *anObject* is present in the receiver, otherwise NO.

Discussion
This method determines whether *anObject* is present in the receiver by sending an isEqual: (page 2101) message to each of the receiver's objects (and passing *anObject* as the parameter to each isEqual: message).

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

See Also
- indexOfObject: (page 123)
- indexOfObjectIdenticalTo: (page 124)

Related Sample Code
TimelineToTC

Declared In
NSArray.h

count

Returns the number of objects currently in the receiver.

```
- (NSUInteger)count
```

Return Value
The number of objects currently in the receiver.

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

See Also
- objectAtIndex: (page 131)

Related Sample Code
CoreRecipes
iSpend
Quartz Composer WWDC 2005 TextEdit

Sketch-112
TextEditPlus

Declared In
`NSArray.h`

description

Returns a string that represents the contents of the receiver, formatted as a property list.

```
- (NSString *)description
```

Return Value
A string that represents the contents of the receiver, formatted as a property list.

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

See Also
- `descriptionWithLocale:` (page 120)
- `descriptionWithLocale:indent:` (page 121)

Declared In
`NSArray.h`

descriptionWithLocale:

Returns a string that represents the contents of the receiver, formatted as a property list.

```
- (NSString *)descriptionWithLocale:(id)locale
```

Parameters
locale
An `NSLocale` object or an `NSDictionary` object that specifies options used for formatting each of the receiver's elements (where recognized). Specify `nil` if you don't want the elements formatted.

Return Value
A string that represents the contents of the receiver, formatted as a property list.

Discussion
For a description of how *locale* is applied to each element in the receiving array, see `descriptionWithLocale:indent:` (page 121).

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

See Also
- `description` (page 120)
- `descriptionWithLocale:indent:` (page 121)

Declared In
`NSArray.h`

descriptionWithLocale:indent:

Returns a string that represents the contents of the receiver, formatted as a property list.

```
- (NSString *)descriptionWithLocale:(id)locale indent:(NSUInteger)level
```

Parameters

locale

An `NSLocale` object or an `NSDictionary` object that specifies options used for formatting each of the receiver's elements (where recognized). Specify `nil` if you don't want the elements formatted.

level

A level of indent, to make the output more readable: set *level* to `0` to use four spaces to indent, or `1` to indent the output with a tab character.

Return Value
A string that represents the contents of the receiver, formatted as a property list.

Discussion
The returned `NSString` object contains the string representations of each of the receiver's elements, in order, from first to last. To obtain the string representation of a given element, `descriptionWithLocale:indent:` proceeds as follows:

- If the element is an `NSString` object, it is used as is.
- If the element responds to `descriptionWithLocale:indent:`, that method is invoked to obtain the element's string representation.
- If the element responds to `descriptionWithLocale:` (page 120), that method is invoked to obtain the element's string representation.
- If none of the above conditions is met, the element's string representation is obtained by invoking its `description` (page 120) method.

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

See Also
- `description` (page 120)
- `descriptionWithLocale:` (page 120)

Declared In
`NSArray.h`

filteredArrayUsingPredicate:

Evaluates a given predicate against each object in the receiver and returns a new array containing the objects for which the predicate returns true.

```
- (NSArray *)filteredArrayUsingPredicate:(NSPredicate *)predicate
```

Parameters

predicate

The predicate against which to evaluate the receiver's elements.

Return Value
A new array containing the objects in the receiver for which *predicate* returns true.

Discussion
For more details, see *Predicate Programming Guide*.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSPredicate.h

firstObjectCommonWithArray:

Returns the first object contained in the receiver that's equal to an object in another given array.

```
- (id)firstObjectCommonWithArray:(NSArray *)otherArray
```

Parameters
otherArray
An array.

Return Value
Returns the first object contained in the receiver that's equal to an object in *otherArray*. If no such object is found, returns nil.

Discussion
This method uses isEqual: (page 2101) to check for object equality.

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

See Also
- containsObject: (page 119)

Declared In
NSArray.h

getObjects:

Copies all the objects contained in the receiver to *aBuffer*.

```
- (void)getObjects:(id *)aBuffer
```

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

See Also
+ arrayWithObjects:count: (page 115)

Declared In
NSArray.h

getObjects:range:

Copies the objects contained in the receiver that fall within the specified range to *aBuffer*.

```
- (void)getObjects:(id *)aBuffer range:(NSRange)aRange
```

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

See Also
+ arrayWithObjects:count: (page 115)

Declared In
NSArray.h

indexOfObject:

Returns the lowest index whose corresponding array value is equal to a given object.

```
- (NSUInteger)indexOfObject:(id)anObject
```

Parameters
anObject
An object.

Return Value
The lowest index whose corresponding array value is equal to *anObject*. If none of the objects in the receiver is equal to *anObject*, returns NSNotFound.

Discussion
Objects are considered equal if isEqual: (page 2101) returns YES.

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

See Also
- containsObject: (page 119)
- indexOfObjectIdenticalTo: (page 124)

Related Sample Code
Core Data HTML Store
NewsReader
WhackedTV

Declared In
NSArray.h

indexOfObject:inRange:

Returns the lowest index within a specified range whose corresponding array value is equal to a given object
.

```
- (NSUInteger)indexOfObject:(id)anObject inRange:(NSRange)range
```

Parameters

anObject

An object.

range

The range of indexes in the receiver within which to search for *anObject*.

Return Value

The lowest index within *range* whose corresponding array value is equal to *anObject*. If none of the objects within *range* is equal to *anObject*, returns `NSNotFound`.

Discussion

Objects are considered equal if `isEqual:` (page 2101) returns `YES`.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `containsObject:` (page 119)
- `indexOfObjectIdenticalTo:inRange:` (page 125)

Declared In

`NSArray.h`

indexOfObjectIdenticalTo:

Returns the lowest index whose corresponding array value is identical to a given object.

```
- (NSUInteger)indexOfObjectIdenticalTo:(id)anObject
```

Parameters

anObject

An object.

Return Value

The lowest index whose corresponding array value is identical to *anObject*. If none of the objects in the receiver is identical to *anObject*, returns `NSNotFound`.

Discussion

Objects are considered identical if their object addresses are the same.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `containsObject:` (page 119)
- `indexOfObject:` (page 123)

Related Sample Code

Quartz Composer WWDC 2005 TextEdit

Sketch-112

TextEditPlus

Declared In

`NSArray.h`

indexOfObjectIdenticalTo:inRange:

Returns the lowest index within a specified range whose corresponding array value is equal to a given object .

```
- (NSUInteger)indexOfObjectIdenticalTo:(id)anObject inRange:(NSRange)range
```

Parameters

anObject

An object.

range

The range of indexes in the receiver within which to search for *anObject*.

Return Value

The lowest index within *range* whose corresponding array value is identical to *anObject*. If none of the objects within *range* is identical to *anObject*, returns NSNotFound.

Discussion

Objects are considered identical if their object addresses are the same.

Availability

Available in Mac OS X v10.0 and later.

See Also

- containsObject: (page 119)
- indexOfObject:inRange: (page 123)

Declared In

NSArray.h

initWithArray:

Initializes a newly allocated array by placing in it the objects contained in a given array.

```
- (id)initWithArray:(NSArray *)anArray
```

Parameters

anArray

An array.

Return Value

An array initialized to contain the objects in *anArray*. The returned object might be different than the original receiver.

Discussion

After an immutable array has been initialized in this way, it cannot be modified.

Availability

Available in Mac OS X v10.0 and later.

See Also

+ arrayWithObject: (page 114)
- initWithObjects: (page 127)

Declared In
NSArray.h

initWithArray:copyItems:

Initializes a newly allocated array using *anArray* as the source of data objects for the array.

```
- (id)initWithArray:(NSArray *)array copyItems:(BOOL)flag
```

Parameters

array

An array.

flag

If YES, each object in *array* receives a copyWithZone: message to create a copy of the object. In a managed memory environment, this is instead of the retain message the object would otherwise receive. The object copy is then added to the returned array.

If NO, then in a managed memory environment each object in *array* simply receives a retain message as it's added to the returned array.

Return Value
An array initialized to contain the objects—or if *flag* is YES, copies of the objects—in *array*. The returned object might be different than the original receiver.

Discussion
After an immutable array has been initialized in this way, it cannot be modified.

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

See Also
- initWithArray: (page 125)
+ arrayWithObject: (page 114)
- initWithObjects: (page 127)

Declared In
NSArray.h

initWithContentsOfFile:

Initializes a newly allocated array with the contents of the file specified by a given path.

```
- (id)initWithContentsOfFile:(NSString *)aPath
```

Parameters

aPath

The path to a file containing a string representation of an array produced by the writeToFile:atomically: (page 139) method.

Return Value
An array initialized to contain the contents of the file specified by *aPath* or nil if the file can't be opened or the contents of the file can't be parsed into an array. The returned object might be different than the original receiver.

Discussion
The array representation in the file identified by *aPath* must contain only property list objects (`NSString`, `NSData`, `NSArray`, or `NSDictionary` objects).

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

See Also
`+ arrayWithContentsOfFile:` (page 113)
`- writeToFile:atomically:` (page 139)

Declared In
`NSArray.h`

initWithContentsOfURL:

Initializes a newly allocated array with the contents of the location specified by a given URL.

```
- (id)initWithContentsOfURL:(NSURL *)aURL
```

Parameters

aURL

The location of a file containing a string representation of an array produced by the `writeToURL:atomically:` (page 140) method.

Return Value
An array initialized to contain the contents specified by *aURL*. Returns `nil` if the location can't be opened or if the contents of the location can't be parsed into an array. The returned object might be different than the original receiver.

Discussion
The array representation at the location identified by *aURL* must contain only property list objects (`NSString`, `NSData`, `NSArray`, or `NSDictionary` objects).

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

See Also
`+ arrayWithContentsOfURL:` (page 113)
`- writeToURL:atomically:` (page 140)

Declared In
`NSArray.h`

initWithObjects:

Initializes a newly allocated array by placing in it the objects in the argument list.

```
- (id)initWithObjects:(id)firstObj, ...
```

Parameters

firstObj, ...

A comma-separated list of objects ending with `nil`.

Return Value
An array initialized to include the objects in the argument list. The returned object might be different than the original receiver.

Discussion
After an immutable array has been initialized in this way, it can't be modified.

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

See Also
- `initWithObjects:count:` (page 128)
+ `arrayWithObjects:` (page 114)
- `initWithArray:` (page 125)

Declared In
`NSArray.h`

initWithObjects:count:

Initializes a newly allocated array to include a given number of objects from a given C array.

```
- (id)initWithObjects:(const id *)objects
    count:(NSUInteger)count
```

Parameters

objects
A C array of objects.

count
The number of values from the *objects* C array to include in the new array. This number will be the count of the new array—it must not be negative or greater than the number of elements in *objects*.

Return Value
A newly allocated array including the first *count* objects from *objects*. The returned object might be different than the original receiver.

Discussion
Elements are added to the new array in the same order they appear in *objects*, up to but not including index *count*.

After an immutable array has been initialized in this way, it can't be modified.

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

See Also
- `initWithObjects:` (page 127)
+ `arrayWithObjects:` (page 114)
- `initWithArray:` (page 125)

Declared In
`NSArray.h`

isEqualToArray:

Compares the receiving array to another array.

```
- (BOOL)isEqualToArray:(NSArray *)otherArray
```

Parameters

otherArray

An array.

Return Value

`YES` if the contents of *otherArray* are equal to the contents of the receiver, otherwise `NO`.

Discussion

Two arrays have equal contents if they each hold the same number of objects and objects at a given index in each array satisfy the `isEqual:` (page 2101) test.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSArray.h`

lastObject

Returns the object in the array with the highest index value.

```
- (id)lastObject
```

Return Value

The object in the array with the highest index value. If the array is empty, returns `nil`.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- removeLastObject` (page 916) (`NSMutableArray`)

Related Sample Code

Core Data HTML Store

CoreRecipes

QTKitAdvancedDocument

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In

`NSArray.h`

makeObjectsPerformSelector:

Sends to each object in the receiver the message identified by a given selector, starting with the first object and continuing through the array to the last object.

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

Parameters

aSelector

A selector that identifies the message to send to the objects in the receiver. The method must not take any arguments, and must not have the side effect of modifying the receiving array.

Discussion

This method raises an `NSInvalidArgumentException` if *aSelector* is `NULL`.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- makeObjectsPerformSelector:withObject:` (page 130)

Related Sample Code

EnhancedDataBurn
QTKitMovieShuffler
Sketch-112
WhackedTV

Declared In

`NSArray.h`

makeObjectsPerformSelector:withObject:

Sends the *aSelector* message to each object in the array, starting with the first object and continuing through the array to the last object.

```
- (void)makeObjectsPerformSelector:(SEL)aSelector withObject:(id)anObject
```

Parameters

aSelector

A selector that identifies the message to send to the objects in the receiver. The method must take a single argument of type `id`, and must not have the side effect of modifying the receiving array.

anObject

The object to send as the argument to each invocation of the *aSelector* method.

Discussion

This method raises an `NSInvalidArgumentException` if *aSelector* is `NULL`.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- makeObjectsPerformSelector:` (page 129)

Related Sample Code

EnhancedDataBurn
ImageBackground
iSpend
QTKitMovieShuffler

Sketch-112

Declared In
NSArray.h

objectAtIndex:

Returns the object located at *index*.

```
- (id)objectAtIndex:(NSUInteger)index
```

Parameters

index

An index within the bounds of the receiver.

Return Value
The object located at *index*.

Discussion
If *index* is beyond the end of the array (that is, if *index* is greater than or equal to the value returned by `count`), an `NSRangeException` is raised.

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

See Also
- `count` (page 119)
- `objectsAtIndexes:` (page 132)

Related Sample Code
CoreRecipes
MyPhoto
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus

Declared In
NSArray.h

objectEnumerator

Returns an enumerator object that lets you access each object in the receiver.

```
- (NSEnumerator *)objectEnumerator
```

Return Value
An enumerator object that lets you access each object in the receiver, in order, from the element at the lowest index upwards.

Discussion
Returns an enumerator object that lets you access each object in the receiver, in order, starting with the element at index 0, as in:

```
NSEnumerator *enumerator = [myArray objectEnumerator];
id anObject;

while (anObject = [enumerator nextObject]) {
    /* code to act on each element as it is returned */
}
```

Special Considerations

When you use this method with mutable subclasses of `NSArray`, you must not modify the array during enumeration.

On Mac OS X v10.5 and later, it is more efficient to use the fast enumeration protocol (see `NSFastEnumeration`).

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

See Also
- `reverseObjectEnumerator` (page 134)
- `nextObject` (page 558) (`NSEnumerator`)

Related Sample Code
CoreRecipes
GridCalendar
iSpend
SimpleCalendar
StickiesExample

Declared In
`NSArray.h`

objectsAtIndexes:

Returns an array containing the objects in the receiver at the indexes specified by a given index set.

```
- (NSArray *)objectsAtIndexes:(NSIndexSet *)indexes
```

Return Value
An array containing the objects in the receiver at the indexes specified by *indexes*.

Discussion
The returned objects are in the ascending order of their indexes in *indexes*, so that object in returned array with higher index in indexes will follow the object with smaller index in *indexes*.

Raises an `NSRangeException` exception if any location in *indexes* exceeds the bounds of the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `count` (page 119)
- `objectAtIndex:` (page 131)

Declared In
NSArray.h

pathsMatchingExtensions:

Returns an array containing all the pathname elements in the receiver that have filename extensions from a given array.

```
- (NSArray *)pathsMatchingExtensions:(NSArray *)filterTypes
```

Parameters

filterTypes

An array of NSString objects containing filename extensions. The extensions should not include the dot (".") character.

Return Value
An array containing all the pathname elements in the receiver that have filename extensions from the *filterTypes* array.

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

Declared In
NSPathUtilities.h

removeObserver:forKeyPath:

Raises an exception.

```
- (void)removeObserver:(NSObject *)observer forKeyPath:(NSString *)keyPath
```

Parameters

observer

The object to remove as an observer.

keyPath

A key-path, relative to the receiver, for which *observer* is registered to receive KVO change notifications. This value must not be nil.

Special Considerations

NSArray objects are not observable, so this method raises an exception when invoked on an NSArray object. Instead of observing an array, observe the to-many relationship for which the array is the collection of related objects.

Availability
Available in Mac OS X v10.4 and later.

See Also
- addObserver:forKeyPath:options:context: (page 116)
- removeObserver:fromObjectsAtIndexes:forKeyPath: (page 134)

Declared In
NSKeyValueObserving.h

removeObserver:fromObjectsAtIndexes:forKeyPath:

Removes *anObserver* from all key value observer notifications associated with the specified *keyPath* relative to the receiver's objects at *indexes*.

```
- (void)removeObserver:(NSObject *)anObserver fromObjectsAtIndexes:(NSIndexSet
    *)indexes forKeyPath:(NSString *)keyPath
```

Discussion
This is not merely a convenience method; invoking this method is potentially much faster than repeatedly invoking `removeObserver:forKeyPath:` (page 2082).

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

See Also
- `addObserver:toObjectsAtIndexes:forKeyPath:options:context:` (page 116)

Related Sample Code
iSpend

Declared In
`NSKeyValueObserving.h`

reverseObjectEnumerator

Returns an enumerator object that lets you access each object in the receiver, in reverse order.

```
- (NSEnumerator *)reverseObjectEnumerator
```

Return Value
An enumerator object that lets you access each object in the receiver, in order, from the element at the highest index down to the element at index `0`.

Special Considerations

When you use this method with mutable subclasses of `NSArray`, you must not modify the array during enumeration.

On Mac OS X v10.5 and later, it is more efficient to use the fast enumeration protocol (see `NSFastEnumeration`).

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

See Also
- `objectEnumerator` (page 131)
- `nextObject` (page 558) (`NSEnumerator`)

Related Sample Code
EnhancedAudioBurn
QTKitMovieShuffler

Declared In
`NSArray.h`

setValue:forKey:

Invokes `setValue:forKey:` on each of the receiver's items using the specified *value* and *key*.

```
- (void)setValue:(id)value forKey:(NSString *)key
```

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

See Also
`- valueForKey:` (page 139)

Related Sample Code
CoreRecipes

Declared In
`NSKeyValueCoding.h`

sortedArrayHint

Analyzes the receiver and returns a "hint" that speeds the sorting of the array when the hint is supplied to `sortedArrayUsingFunction:context:hint:` (page 137).

```
- (NSData *)sortedArrayHint
```

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

See Also
`- sortedArrayUsingFunction:context:hint:` (page 137)

Declared In
`NSArray.h`

sortedArrayUsingDescriptors:

Returns a copy of the receiver sorted as specified by a given array of sort descriptors.

```
- (NSArray *)sortedArrayUsingDescriptors:(NSArray *)sortDescriptors
```

Parameters

sortDescriptors
An array of `NSSortDescriptor` objects.

Return Value
A copy of the receiver sorted as specified by *sortDescriptors*.

Discussion
The first descriptor specifies the primary key path to be used in sorting the receiver's contents. Any subsequent descriptors are used to further refine sorting of objects with duplicate values. See `NSSortDescriptor` for additional information.

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

See Also
- `sortedArrayUsingSelector:` (page 138)
- `sortedArrayUsingFunction:context:` (page 136)
- `sortedArrayUsingFunction:context:hint:` (page 137)

Related Sample Code
CoreRecipes

Declared In
`NSSortDescriptor.h`

sortedArrayUsingFunction:context:

Returns a new array that lists the receiver's elements in ascending order as defined by the comparison function *comparator*.

```
- (NSArray *)sortedArrayUsingFunction:(NSInteger (*)(id, id, void *))comparator
    context:(void *)context
```

Discussion
The new array contains references to the receiver's elements, not copies of them.

The comparison function is used to compare two elements at a time and should return `NSOrderedAscending` if the first element is smaller than the second, `NSOrderedDescending` if the first element is larger than the second, and `NSOrderedSame` if the elements are equal. Each time the comparison function is called, it's passed *context* as its third argument. This allows the comparison to be based on some outside parameter, such as whether character sorting is case-sensitive or case-insensitive.

Given *anArray* (an array of `NSNumber` objects) and a comparison function of this type:

```
NSInteger intSort(id num1, id num2, void *context)
{
    int v1 = [num1 intValue];
    int v2 = [num2 intValue];
    if (v1 < v2)
        return NSOrderedAscending;
    else if (v1 > v2)
        return NSOrderedDescending;
    else
        return NSOrderedSame;
}
```

A sorted version of *anArray* is created in this way:

```
NSArray *sortedArray; sortedArray = [anArray sortedArrayUsingFunction:intSort
context:NULL];
```

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

See Also
- `sortedArrayUsingDescriptors:` (page 135)
- `sortedArrayUsingFunction:context:hint:` (page 137)
- `sortedArrayUsingSelector:` (page 138)

Related Sample Code
Birthdays
NewsReader

Declared In
NSArray.h

sortedArrayUsingFunction:context:hint:

Returns a new array that lists the receiver's elements in ascending order as defined by the comparison function *comparator*.

```
- (NSArray *)sortedArrayUsingFunction:(NSInteger (*)(id, id, void *))comparator
    context:(void *)context hint:(NSData *)hint
```

Discussion
The new array contains references to the receiver's elements, not copies of them.

This method is similar to `sortedArrayUsingFunction:context:` (page 136), except that it uses the supplied hint to speed the sorting process. When you know the array is nearly sorted, this method is faster than `sortedArrayUsingFunction:context:`. If you sorted a large array (`N` entries) once, and you don't change it much (`P` additions and deletions, where `P` is much smaller than `N`), then you can reuse the work you did in the original sort by conceptually doing a merge sort between the `N` "old" items and the `P` "new" items.

To obtain an appropriate hint, use `sortedArrayHint` (page 135). You should obtain this hint when the original array has been sorted, and keep hold of it until you need it, after the array has been modified. The hint is computed by `sortedArrayHint` (page 135) in `O(N)` (where `N` is the number of items). This assumes that items in the array implement a `-hash` method. Given a suitable hint, and assuming that the hash function is a "good" hash function, `-sortedArrayUsingFunction:context:hint:` (page 137) sorts the array in `O(P*LOG(P)+N)` where `P` is the number of adds or deletes. This is an improvement over the unhinted sort, `O(N*LOG(N))`, when `P` is small.

The hint is simply an array of size `N` containing the `N` hashes. To re-sort you need internally to create a map table mapping a hash to the index. Using this map table on the new array, you can get a first guess for the indices, and then sort that. For example, a sorted array {A, B, D, E, F} with corresponding hash values {25, 96, 78, 32, 17}, may be subject to small changes that result in contents {E, A, C, B, F}. The mapping table maps the hashes {25, 96, 78, 32, 17} to the indices {#0, #1, #2, #3, #4}. If the hashes for {E, A, C, B, F} are {32, 25, 99, 96, 17}, then by using the mapping table you can get a first order sort {#3, #0, ?, #1, #4}, so therefore create an initial semi-sorted array {A, B, E, F}, and then perform a cheap merge sort with {C} that yields {A, B, C, E, F}.

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

See Also
- `sortedArrayUsingDescriptors:` (page 135)
- `sortedArrayUsingFunction:context:` (page 136)
- `sortedArrayUsingSelector:` (page 138)

Declared In
NSArray.h

sortedArrayUsingSelector:

Returns an array that lists the receiver's elements in ascending order, as determined by the comparison method specified by a given selector.

```
- (NSArray *)sortedArrayUsingSelector:(SEL)comparator
```

Parameters

comparator

A selector that identifies the method to use to compare two elements at a time. The method should return `NSOrderedAscending` if the receiver is smaller than the argument, `NSOrderedDescending` if the receiver is larger than the argument, and `NSOrderedSame` if they are equal.

Return Value

An array that lists the receiver's elements in ascending order, as determined by the comparison method specified by the selector *comparator*.

Discussion

The new array contains references to the receiver's elements, not copies of them.

The *comparator* message is sent to each object in the array and has as its single argument another object in the array.

For example, an array of `NSString` objects can be sorted by using the `caseInsensitiveCompare:` (page 1540) method declared in the `NSString` class. Assuming *anArray* exists, a sorted version of the array can be created in this way:

```
NSArray *sortedArray =
    [anArray sortedArrayUsingSelector:@selector(caseInsensitiveCompare:)];
```

Availability

Available in Mac OS X v10.0 and later.

See Also

- `sortedArrayUsingDescriptors:` (page 135)
- `sortedArrayUsingFunction:context:` (page 136)
- `sortedArrayUsingFunction:context:hint:` (page 137)

Related Sample Code

CoreRecipes

EnhancedAudioBurn

QTSSInspector

Declared In

`NSArray.h`

subarrayWithRange:

Returns a new array containing the receiver's elements that fall within the limits specified by a given range.

```
- (NSArray *)subarrayWithRange:(NSRange)range
```

Parameters

range

A range within the receiver's range of elements.

Return Value

A new array containing the receiver's elements that fall within the limits specified by *range*.

Discussion

If *range* isn't within the receiver's range of elements, an `NSRangeException` is raised.

For example, the following code example creates an array containing the elements found in the first half of *wholeArray* (assuming *wholeArray* exists).

```
NSArray *halfArray;
NSRange theRange;

theRange.location = 0;
theRange.length = [wholeArray count] / 2;

halfArray = [wholeArray subarrayWithRange:theRange];
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSArray.h`

valueForKey:

Returns an array containing the results of invoking `valueForKey:` using *key* on each of the receiver's objects.

```
- (id)valueForKey:(NSString *)key
```

Discussion

The returned array contains `NSNull` elements for each object that returns `nil`.

Availability

Available in Mac OS X v10.3 and later.

See Also

`- setValue:forKey:` (page 135)

Related Sample Code

Core Data HTML Store

CoreRecipes

StickiesExample

Declared In

`NSKeyValueCoding.h`

writeToFile:atomically:

Writes the contents of the receiver to a file at a given path.

```
- (BOOL)writeToFile:(NSString *)path atomically:(BOOL)flag
```

Parameters

path

The path at which to write the contents of the receiver.

If *path* contains a tilde (~) character, you must expand it with `stringByExpandingTildeInPath` (page 1602) before invoking this method.

flag

If `YES`, the array is written to an auxiliary file, and then the auxiliary file is renamed to *path*. If `NO`, the array is written directly to *path*. The `YES` option guarantees that *path*, if it exists at all, won't be corrupted even if the system should crash during writing.

Return Value

`YES` if the file is written successfully, otherwise `NO`.

Discussion

If the receiver's contents are all property list objects (`NSString`, `NSData`, `NSArray`, or `NSDictionary` objects), the file written by this method can be used to initialize a new array with the class method `arrayWithContentsOfFile:` (page 113) or the instance method `initWithContentsOfFile:` (page 126). This method recursively validates that all the contained objects are property list objects before writing out the file, and returns `NO` if all the objects are not property list objects, since the resultant file would not be a valid property list.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- initWithContentsOfFile:` (page 126)

Declared In

`NSArray.h`

writeToURL:atomically:

Writes the contents of the receiver to the location specified by a given URL.

```
- (BOOL)writeToURL:(NSURL *)aURL atomically:(BOOL)flag
```

Parameters

aURL

The location at which to write the receiver.

flag

If `YES`, the array is written to an auxiliary location, and then the auxiliary location is renamed to *aURL*. If `NO`, the array is written directly to *aURL*. The `YES` option guarantees that *aURL*, if it exists at all, won't be corrupted even if the system should crash during writing.

Return Value

`YES` if the location is written successfully, otherwise `NO`.

Discussion

If the receiver's contents are all property list objects (`NSString`, `NSData`, `NSArray`, or `NSDictionary` objects), the location written by this method can be used to initialize a new array with the class method `arrayWithContentsOfURL:` (page 113) or the instance method `initWithContentsOfURL:` (page 127).

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

See Also
`- initWithContentsOfURL:` (page 127)

Declared In
`NSArray.h`

NSAssertionHandler Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSException.h
Companion guide	Assertions and Logging

Overview

`NSAssertionHandler` objects are automatically created to handle false assertions. Assertion macros, such as `NSAssert` and `NSCAssert`, are used to evaluate a condition, and, if the condition evaluates to false, the macros pass a string to an `NSAssertionHandler` object describing the failure. Each thread has its own `NSAssertionHandler` object. When invoked, an assertion handler prints an error message that includes the method and class (or function) containing the assertion and raises an `NSInternalInconsistencyException`.

You create assertions only using the assertion macros—you rarely need to invoke `NSAssertionHandler` methods directly. The macros for use inside methods and functions send `handleFailureInMethod:object:file:lineNumber:description:` (page 145) and `handleFailureInFunction:file:lineNumber:description:` (page 144) messages respectively to the current assertion handler. The assertion handler for the current thread is obtained using the `currentHandler` (page 144) class method. If you need to customize the behavior of `NSAssertionHandler`, create a subclass, overriding the above two methods, and install your instance into the current thread's attributes dictionary with the key `NSAssertionHandler`.

Tasks

Handling Assertion Failures

`+ currentHandler` (page 144)
Returns the `NSAssertionHandler` object associated with the current thread.

`- handleFailureInFunction:file:lineNumber:description:` (page 144)
Logs (using `NSLog`) an error message that includes the name of the function, the name of the file, and the line number.

`- handleFailureInMethod:object:file:lineNumber:description:` (page 145)
Logs (using `NSLog`) an error message that includes the name of the method that failed, the class name of the object, the name of the source file, and the line number.

Class Methods

currentHandler

Returns the `NSAssertionHandler` object associated with the current thread.

```
+ (NSAssertionHandler *)currentHandler
```

Return Value
The `NSAssertionHandler` object associated with the current thread.

Discussion
If no assertion handler is associated with the current thread, this method creates one and assigns it to the thread.

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

Declared In
`NSException.h`

Instance Methods

handleFailureInFunction:file:lineNumber:description:

Logs (using `NSLog`) an error message that includes the name of the function, the name of the file, and the line number.

```
- (void)handleFailureInFunction:(NSString *)functionName file:(NSString *)fileName
    lineNumber:(NSInteger)line description:(NSString *)format, ...
```

Parameters
functionName
The function that failed.

object
The object that failed.

fileName
The name of the source file.

line
The line in which the failure occurred.

format,...
A format string followed by a comma-separated list of arguments to substitute into the format string. See Formatting String Objects for more information.

Discussion
Raises `NSInternalInconsistencyException`.

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

Declared In
`NSException.h`

handleFailureInMethod:object:file:lineNumber:description:

Logs (using `NSLog`) an error message that includes the name of the method that failed, the class name of the object, the name of the source file, and the line number.

```
- (void)handleFailureInMethod:(SEL)selector object:(id)object file:(NSString
    *)fileName lineNumber:(NSInteger)line description:(NSString *)format, ...
```

Parameters
selector
The selector for the method that failed

object
The object that failed.

fileName
The name of the source file.

line
The line in which the failure occurred.

format,...
A format string followed by a comma-separated list of arguments to substitute into the format string. See Formatting String Objects for more information.

Discussion
Raises `NSInternalInconsistencyException`.

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

Declared In
`NSException.h`

NSAttributedString Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSMutableCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSAttributedString.h
Companion guide	Attributed Strings Programming Guide
Related sample code	CIAnnotation CoreRecipes iSpend OpenGL Screensaver Sketch-112

Overview

`NSAttributedString` objects manage character strings and associated sets of attributes (for example, font and kerning) that apply to individual characters or ranges of characters in the string. An association of characters and their attributes is called an attributed string. The cluster's two public classes, `NSAttributedString` and `NSMutableAttributedString`, declare the programmatic interface for read-only attributed strings and modifiable attributed strings, respectively. The Foundation framework defines only the basic functionality for attributed strings; additional methods supporting RTF, graphics attributes, and drawing attributed strings are described in NSAttributedString Additions, found in the Application Kit. The Application Kit also uses a subclass of `NSMutableAttributedString`, called NSTextStorage, to provide the storage for the Application Kit's extended text-handling system.

The Application Kit also uses NSParagraphStyle and its subclass NSMutableParagraphStyle to encapsulate the paragraph or ruler attributes used by the `NSAttributedString` classes.

An attributed string identifies attributes by name, storing a value under the name in an `NSDictionary` object. Standard attribute keys are described in the "Constants" section of *NSAttributedString Application Kit Additions Reference*. You can also assign any attribute name/value pair you wish to a range of characters—it is up to your application to interpret custom attributes (see *Attributed Strings Programming Guide*).

Note that the default font for `NSAttributedString` objects is Helvetica 12-point, which differs from the Mac OS X system font Lucida Grande, so you may wish to create the string with non-default attributes suitable for your application using, for example, `initWithString:attributes:` (page 154).

Be aware that `isEqual:` comparison among `NSAttributedString` objects compares for exact equality, including not only literal character-by-character string equality but also equality of all attributes, which is not likely to be achieved in the case of many attributes such as attachments, lists, and tables, for example.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 2034)
- `initWithCoder:` (page 2034)

NSCopying
- `copyWithZone:` (page 2042)

NSMutableCopying
- `mutableCopyWithZone:` (page 2094)

Tasks

Creating an NSAttributedString Object

- `initWithString:` (page 153)
  Returns an `NSAttributedString` object initialized with the characters of a given string and no attribute information.
- `initWithAttributedString:` (page 153)
  Returns an `NSAttributedString` object initialized with the characters and attributes of another given attributed string.
- `initWithString:attributes:` (page 154)
  Returns an `NSAttributedString` object initialized with a given string and attributes.

Retrieving Character Information

- `string` (page 155)
  Returns the character contents of the receiver as an `NSString` object.
- `length` (page 155)
  Returns the length of the receiver's string object.

Retrieving Attribute Information

`- attributesAtIndex:effectiveRange:` (page 152)
Returns the attributes for the character at a given index.

`- attributesAtIndex:longestEffectiveRange:inRange:` (page 152)
Returns the attributes for the character at a given index, and by reference the range over which the attributes apply.

`- attribute:atIndex:effectiveRange:` (page 149)
Returns the value for an attribute with a given name of the character at a given index, and by reference the range over which the attribute applies.

`- attribute:atIndex:longestEffectiveRange:inRange:` (page 150)
Returns the value for the attribute with a given name of the character at a given index, and by reference the range over which the attribute applies.

Comparing Attributed Strings

`- isEqualToAttributedString:` (page 154)
Returns a Boolean value that indicates whether the receiver is equal to another given attributed string.

Extracting a Substring

`- attributedSubstringFromRange:` (page 151)
Returns an `NSAttributedString` object consisting of the characters and attributes within a given range in the receiver.

Instance Methods

attribute:atIndex:effectiveRange:

Returns the value for an attribute with a given name of the character at a given index, and by reference the range over which the attribute applies.

```
- (id)attribute:(NSString *)attributeName atIndex:(NSUInteger)index
    effectiveRange:(NSRangePointer)aRange
```

Parameters

attributeName
The name of an attribute.

index
The index for which to return attributes. This value must not exceed the bounds of the receiver.

aRange

If non-NULL:

- If the named attribute exists at *index*, upon return *aRange* contains a range over which the named attribute's value applies.
- If the named attribute does not exist at *index*, upon return *aRange* contains the range over which the attribute does not exist.

The range isn't necessarily the maximum range covered by *attributeName*, and its extent is implementation-dependent. If you need the maximum range, use `attribute:atIndex:longestEffectiveRange:inRange:` (page 150). If you don't need this value, pass NULL.

Return Value
The value for the attribute named *attributeName* of the character at index *index*, or nil if there is no such attribute.

Discussion
Raises an NSRangeException if *index* lies beyond the end of the receiver's characters.

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

See Also
`- attributesAtIndex:effectiveRange:` (page 152)

Related Sample Code
iSpend
TextLinks

Declared In
NSAttributedString.h

attribute:atIndex:longestEffectiveRange:inRange:

Returns the value for the attribute with a given name of the character at a given index, and by reference the range over which the attribute applies.

```
- (id)attribute:(NSString *)attributeName atIndex:(NSUInteger)index
    longestEffectiveRange:(NSRangePointer)aRange inRange:(NSRange)rangeLimit
```

Parameters

attributeName

The name of an attribute.

index

The index at which to test for *attributeName*.

aRange

If non-NULL:

- If the named attribute exists at *index*, upon return *aRange* contains the full range over which the value of the named attribute is the same as that at *index*, clipped to *rangeLimit*.
- If the named attribute does not exist at *index*, upon return *aRange* contains the full range over which the attribute does not exist, clipped to *rangeLimit*.

If you don't need this value, pass NULL.

rangeLimit

The range over which to search for continuous presence of *attributeName*. This value must not exceed the bounds of the receiver.

Return Value
The value for the attribute named *attributeName* of the character at *index*, or nil if there is no such attribute.

Discussion
Raises an NSRangeException if *index* or any part of *rangeLimit* lies beyond the end of the receiver's characters.

If you don't need the longest effective range, it's far more efficient to use the attribute:atIndex:effectiveRange: (page 149) method to retrieve the attribute value.

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

See Also
- attributesAtIndex:longestEffectiveRange:inRange: (page 152)

Declared In
NSAttributedString.h

attributedSubstringFromRange:

Returns an NSAttributedString object consisting of the characters and attributes within a given range in the receiver.

```
- (NSAttributedString *)attributedSubstringFromRange:(NSRange)aRange
```

Parameters

aRange

The range from which to create a new attributed string. *aRange* must lie within the bounds of the receiver.

Return Value
An NSAttributedString object consisting of the characters and attributes within *aRange* in the receiver.

Discussion
Raises an NSRangeException if any part of *aRange* lies beyond the end of the receiver's characters. This method treats the length of the string as a valid range value that returns an empty string.

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

Declared In
NSAttributedString.h

attributesAtIndex:effectiveRange:

Returns the attributes for the character at a given index.

```
- (NSDictionary *)attributesAtIndex:(NSUInteger)index
    effectiveRange:(NSRangePointer)aRange
```

Parameters

index

The index for which to return attributes. This value must lie within the bounds of the receiver.

aRange

Upon return, the range over which the attributes and values are the same as those at *index*. This range isn't necessarily the maximum range covered, and its extent is implementation-dependent. If you need the maximum range, use `attributesAtIndex:longestEffectiveRange:inRange:` (page 152). If you don't need this value, pass `NULL`.

Return Value
The attributes for the character at *index*.

Discussion
Raises an `NSRangeException` if *index* lies beyond the end of the receiver's characters.

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

See Also
`- attribute:atIndex:effectiveRange:` (page 149)

Declared In
NSAttributedString.h

attributesAtIndex:longestEffectiveRange:inRange:

Returns the attributes for the character at a given index, and by reference the range over which the attributes apply.

```
- (NSDictionary *)attributesAtIndex:(NSUInteger)index
    longestEffectiveRange:(NSRangePointer)aRange inRange:(NSRange)rangeLimit
```

Parameters

index

The index for which to return attributes. This value must not exceed the bounds of the receiver.

aRange

If non-`NULL`, upon return contains the maximum range over which the attributes and values are the same as those at *index*, clipped to *rangeLimit*.

rangeLimit
The range over which to search for continuous presence of the attributes at *index*. This value must not exceed the bounds of the receiver.

Discussion
Raises an `NSRangeException` if *index* or any part of *rangeLimit* lies beyond the end of the receiver's characters.

If you don't need the range information, it's far more efficient to use the `attributesAtIndex:effectiveRange:` (page 152) method to retrieve the attribute value.

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

See Also
`- attribute:atIndex:longestEffectiveRange:inRange:` (page 150)

Declared In
`NSAttributedString.h`

initWithAttributedString:

Returns an `NSAttributedString` object initialized with the characters and attributes of another given attributed string.

```
- (id)initWithAttributedString:(NSAttributedString *)attributedString
```

Parameters
attributedString
An attributed string.

Return Value
An `NSAttributedString` object initialized with the characters and attributes of *attributedString*. The returned object might be different than the original receiver.

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

See Also
`- initWithRTF:documentAttributes:` (NSAttributedString Additions)

Related Sample Code
Sketch-112

Declared In
`NSAttributedString.h`

initWithString:

Returns an `NSAttributedString` object initialized with the characters of a given string and no attribute information.

```
- (id)initWithString:(NSString *)aString
```

Parameters

aString

The characters for the new object.

Return Value

An `NSAttributedString` object initialized with the characters of *aString* and no attribute information The returned object might be different than the original receiver.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- initWithRTF:documentAttributes:` (NSAttributedString Additions)

Declared In

`NSAttributedString.h`

initWithString:attributes:

Returns an `NSAttributedString` object initialized with a given string and attributes.

```
- (id)initWithString:(NSString *)aString attributes:(NSDictionary *)attributes
```

Parameters

aString

The string for the new attributed string.

attributes

The attributes for the new attributed string. You can assign to a range of characters any attribute name/value pairs you wish, in addition to the standard attributes described in the "Constants" section of *NSAttributedString Application Kit Additions Reference*.

Discussion

Returns an `NSAttributedString` object initialized with the characters of *aString* and the attributes of *attributes*. The returned object might be different from the original receiver.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- initWithRTF:documentAttributes:` (NSAttributedString Additions)

Related Sample Code

Aperture Edit Plugin - Borders & Titles

CIAnnotation

OpenGL Screensaver

Declared In

`NSAttributedString.h`

isEqualToAttributedString:

Returns a Boolean value that indicates whether the receiver is equal to another given attributed string.

```
- (BOOL)isEqualToAttributedString:(NSAttributedString *)otherString
```

Parameters

otherString

The attributed string with which to compare the receiver.

Return Value

`YES` if the receiver is equal to *otherString*, otherwise `NO`.

Discussion

Attributed strings must match in both characters and attributes to be equal.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSAttributedString.h`

length

Returns the length of the receiver's string object.

```
- (NSUInteger)length
```

Availability

Available in Mac OS X v10.0 and later.

See Also

`length` (page 1580) (NSString)

`- size` (NSAttributedString Additions)

Related Sample Code

NumberInput_IMKit_Sample

VertexPerformanceTest

Declared In

`NSAttributedString.h`

string

Returns the character contents of the receiver as an `NSString` object.

```
- (NSString *)string
```

Return Value

The character contents of the receiver as an `NSString` object.

Discussion

This method doesn't strip out attachment characters; use `NSText`'s `string` method to extract just the linguistically significant characters.

For performance reasons, this method returns the current backing store of the attributed string object. If you want to maintain a snapshot of this as you manipulate the returned string, you should make a copy of the appropriate substring.

This primitive method must guarantee efficient access to an attributed string's characters; subclasses should implement it to execute in O(1) time.

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

Related Sample Code
iSpend
NumberInput_IMKit_Sample
Spotlight

Declared In
`NSAttributedString.h`

Constants

Standard attribute keys are described in the "Constants" section of *NSAttributedString Application Kit Additions Reference*.

NSAutoreleasePool Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSAutoreleasePool.h
Companion guide	Memory Management Programming Guide for Cocoa
Related sample code	CocoaSpeechSynthesisExample NumberInput_IMKit_Sample SpellingChecker CarbonCocoa Bundled SpellingChecker-CarbonCocoa SpellingChecker-CocoaCarbon

Overview

The `NSAutoreleasePool` class is used to support Cocoa's reference-counted memory management system. An autorelease pool stores objects that are sent a `release` message when the pool itself is drained.

In a reference-counted environment (as opposed to one which uses garbage collection), an `NSAutoreleasePool` object contains objects that have received an `autorelease` (page 2099) message and when drained it sends a `release` (page 2106) message to each of those objects. Thus, sending `autorelease` (page 2099) instead of `release` (page 2106) to an object extends the lifetime of that object at least until the pool itself is drained (it may be longer if the object is subsequently retained). An object can be put into the same pool several times, in which case it receives a `release` (page 2106) message for each time it was put into the pool.

In a reference counted environment, Cocoa expects there to be an autorelease pool always available. If a pool is not available, autoreleased objects do not get released and you leak memory. In this situation, your program will typically log suitable warning messages.

The Application Kit creates an autorelease pool on the main thread at the beginning of every cycle of the event loop, and drains it at the end, thereby releasing any autoreleased objects generated while processing an event. If you use the Application Kit, you therefore typically don't have to create your own pools. If your application creates a lot of temporary autoreleased objects within the event loop, however, it may be beneficial to create "local" autorelease pools to help to minimize the peak memory footprint.

You create an `NSAutoreleasePool` object with the usual `alloc` and `init` messages and dispose of it with `drain` (page 160) (or `release` (page 161)—to understand the difference, see "Garbage Collection" (page 158)). Since you cannot retain an autorelease pool (or autorelease it—see `retain` (page 161) and `autorelease` (page 160)), draining a pool ultimately has the effect of deallocating it. You should always drain an autorelease pool in the same context (invocation of a method or function, or body of a loop) that it was created. See Autorelease Pools for more details.

Each thread (including the main thread) maintains its own stack of `NSAutoreleasePool` objects (see "Threads" (page 158)). As new pools are created, they get added to the top of the stack. When pools are deallocated, they are removed from the stack. Autoreleased objects are placed into the top autorelease pool for the current thread. When a thread terminates, it automatically drains all of the autorelease pools associated with itself.

Threads

If you are making Cocoa calls outside of the Application Kit's main thread—for example if you create a Foundation-only application or if you detach a thread—you need to create your own autorelease pool.

If your application or thread is long-lived and potentially generates a lot of autoreleased objects, you should periodically drain and create autorelease pools (like the Application Kit does on the main thread); otherwise, autoreleased objects accumulate and your memory footprint grows. If, however, your detached thread does not make Cocoa calls, you do not need to create an autorelease pool.

Note: If you are creating secondary threads using the POSIX thread APIs instead of `NSThread` objects, you cannot use Cocoa, including `NSAutoreleasePool`, unless Cocoa is in multithreading mode. Cocoa enters multithreading mode only after detaching its first `NSThread` object. To use Cocoa on secondary POSIX threads, your application must first detach at least one `NSThread` object, which can immediately exit. You can test whether Cocoa is in multithreading mode with the `NSThread` class method `isMultiThreaded` (page 1642).

Garbage Collection

In a garbage-collected environment, there is no need for autorelease pools. You may, however, write a framework that is designed to work in both a garbage-collected and reference-counted environment. In this case, you can use autorelease pools to hint to the collector that collection may be appropriate. In a garbage-collected environment, sending a `drain` (page 160) message to a pool triggers garbage collection if necessary; `release` (page 161), however, is a no-op. In a reference-counted environment, `drain` (page 160) has the same effect as `release` (page 161). Typically, therefore, you should use `drain` (page 160) instead of `release` (page 161).

Tasks

Managing a Pool

- `release` (page 161)
  Releases and pops the receiver.

- `drain` (page 160)
  In a reference-counted environment, releases and pops the receiver; in a garbage-collected environment, triggers garbage collection if the memory allocated since the last collection is greater than the current threshold.
- `autorelease` (page 160)
  Raises an exception.
- `retain` (page 161)
  Raises an exception.

Adding an Object to a Pool

+ `addObject:` (page 159)
  Adds a given object to the active autorelease pool in the current thread.
- `addObject:` (page 160)
  Adds a given object to the receiver

Class Methods

addObject:

Adds a given object to the active autorelease pool in the current thread.

```
+ (void)addObject:(id)object
```

Parameters

object
  The object to add to the active autorelease pool in the current thread.

Discussion

The same object may be added several times to the active pool and, when the pool is deallocated, it will receive a `release` (page 2106) message for each time it was added.

Normally you don't invoke this method directly—you send `autorelease` (page 2099) to *object* instead.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `addObject:` (page 160)

Declared In

`NSAutoreleasePool.h`

Instance Methods

addObject:

Adds a given object to the receiver

```
- (void)addObject:(id)object
```

Parameters

object

The object to add to the receiver.

Discussion

The same object may be added several times to the same pool; when the pool is deallocated, the object will receive a `release` (page 2106) message for each time it was added.

Normally you don't invoke this method directly—you send `autorelease` (page 2099) to *object* instead.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ addObject:` (page 159)

Declared In

`NSAutoreleasePool.h`

autorelease

Raises an exception.

```
- (id)autorelease
```

Return Value

`self`.

Discussion

In a reference-counted environment, this method raises an exception.

drain

In a reference-counted environment, releases and pops the receiver; in a garbage-collected environment, triggers garbage collection if the memory allocated since the last collection is greater than the current threshold.

```
- (void)drain
```

Discussion
In a reference-counted environment, this method behaves the same as `release` (page 2106). Since an autorelease pool cannot be retained (see `retain` (page 161)), this therefore causes the receiver to be deallocated. When an autorelease pool is deallocated, it sends a `release` (page 2106) message to all its autoreleased objects. If an object is added several times to the same pool, when the pool is deallocated it receives a `release` (page 2106) message for each time it was added.

In a garbage-collected environment, this method ultimately calls `objc_collect_if_needed`.

Special Considerations

In a garbage-collected environment, `release` is a no-op, so unless you do not want to give the collector a hint it is important to use `drain` in any code that may be compiled for a garbage-collected environment.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
Core Data HTML Store
MassiveImage
VideoViewer

Declared In
`NSAutoreleasePool.h`

release

Releases and pops the receiver.

```
- (void)release
```

Discussion
In a reference-counted environment, since an autorelease pool cannot be retained (see `retain` (page 161)), this method causes the receiver to be deallocated. When an autorelease pool is deallocated, it sends a `release` (page 2106) message to all its autoreleased objects. If an object is added several times to the same pool, when the pool is deallocated it receives a `release` (page 2106) message for each time it was added.

In a garbage-collected environment, this method is a no-op.

Special Considerations

You should typically use `drain` (page 160) instead of `release`.

See Also
`- drain` (page 160)

retain

Raises an exception.

```
- (id)retain
```

Return Value
`self`.

Discussion
In a reference-counted environment, this method raises an exception.

NSBundle Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSBundle.h
Companion guides	Bundle Programming Guide Resource Programming Guide
Related sample code	CoreRecipes GLSLShowpiece NumberInput_IMKit_Sample Quartz Composer WWDC 2005 TextEdit TextEditPlus

Overview

An `NSBundle` object represents a location in the file system that groups code and resources that can be used in a program. `NSBundle` objects locate program resources, dynamically load and unload executable code, and assist in localization. You build a bundle in Xcode using one of these project types: Application, Framework, Loadable Bundle, Palette.

See also NSBundle Additions in the Application Kit framework, which defines methods for loading nib files and locating image resources.

Unlike some other Foundation classes with corresponding Core Foundation names (such as `NSString` and CFString), `NSBundle` objects cannot be cast ("toll-free bridged") to CFBundle references. If you need functionality provided in CFBundle, you can still create a CFBundle and use the CFBundle API. See Interchangeable Data Types for more information on toll-free bridging.

Tasks

Initializing an NSBundle

- `initWithPath:` (page 176)
  Returns an `NSBundle` object initialized to correspond to a given directory.

Getting an NSBundle

+ `bundleForClass:` (page 167)
  Returns the `NSBundle` object with which a given class is associated.

+ `bundleWithIdentifier:` (page 168)
  Returns the previously created `NSBundle` instance that has a given bundle identifier.

+ `bundleWithPath:` (page 169)
  Returns an `NSBundle` object that corresponds to the specified directory.

+ `mainBundle` (page 169)
  Returns the `NSBundle` object that corresponds to the directory where the current application executable is located.

+ `allBundles` (page 167)
  Returns an array of all the application's non-framework bundles.

+ `allFrameworks` (page 167)
  Returns an array of all of the application's bundles that represent frameworks.

Getting a Bundled Class

- `classNamed:` (page 174)
  Returns the `Class` object for the specified name.

- `principalClass` (page 187)
  Returns the receiver's principal class.

Finding a Resource

+ `pathForResource:ofType:inDirectory:` (page 170)
  Returns the full pathname for the resource file identified by a given name and extension and residing in a given bundle directory.

- `pathForResource:ofType:` (page 182)
  Returns the full pathname for the resource identified by a given name and specified file extension.

- `pathForResource:ofType:inDirectory:` (page 183)
  Returns the full pathname for the resource identified by the given name and file extension and located in the specified bundle subdirectory.

- `pathForResource:ofType:inDirectory:forLocalization:` (page 184)
  Returns the full pathname for the resource identified by the given name and file extension, located in the specified bundle subdirectory, and limited to global resources and those associated with the specified localization.
+ `pathsForResourcesOfType:inDirectory:` (page 171)
  Returns an array containing the pathnames for all bundle resources having a given extension and residing in the bundle directory specified by a given path.
- `pathsForResourcesOfType:inDirectory:` (page 185)
  Returns an array containing the pathnames for all bundle resources having the specified filename extension and residing in the resource subdirectory.
- `pathsForResourcesOfType:inDirectory:forLocalization:` (page 186)
  Returns an array containing the pathnames for all bundle resources having the specified filename extension, residing in the specified resource subdirectory, and limited to global resources and those associated with the specified localization.
- `resourcePath` (page 189)
  Returns the full pathname of the receiving bundle's subdirectory containing resources.

Getting the Bundle Directory

- `bundlePath` (page 174)
  Returns the full pathname of the receiver's bundle directory.

Getting Bundle Information

- `builtInPlugInsPath` (page 173)
  Returns the full pathname of the receiver's subdirectory containing plug-ins.
- `bundleIdentifier` (page 173)
  Returns the receiver's bundle identifier.
- `executablePath` (page 175)
  Returns the full pathname of the receiver's executable file.
- `infoDictionary` (page 176)
  Returns a dictionary that contains information about the receiver.
- `objectForInfoDictionaryKey:` (page 181)
  Returns the value associated with a given key in the receiver's property list.
- `pathForAuxiliaryExecutable:` (page 182)
  Returns the full pathname of the executable with a given name in the receiver's bundle.
- `privateFrameworksPath` (page 188)
  Returns the full pathname of the receiver's subdirectory containing private frameworks.
- `sharedFrameworksPath` (page 189)
  Returns the full pathname of the receiver's subdirectory containing shared frameworks.
- `sharedSupportPath` (page 190)
  Returns the full pathname of the receiver's subdirectory containing shared support files.

Managing Localized Resources

- `localizedStringForKey:value:table:` (page 180)
  Returns a localized version of the string designated by a given key in a given table.

Loading a Bundle's Code

- `executableArchitectures` (page 175)
  Returns an array of numbers indicating the architecture types supported by the bundle's executable.
- `preflightAndReturnError:` (page 187)
  Returns a Boolean value indicating whether the bundle's executable code could be loaded successfully.
- `load` (page 177)
  Dynamically loads the bundle's executable code into a running program, if the code has not already been loaded.
- `loadAndReturnError:` (page 178)
  Loads the bundle's executable code and returns any errors.
- `isLoaded` (page 177)
  Obtains information about the load status of a bundle.
- `unload` (page 190)
  Unloads the code associated with the receiver.

Managing Localizations

+ `preferredLocalizationsFromArray:` (page 172)
  Returns one or more localizations from the specified list that a bundle object would use to locate resources for the current user.
+ `preferredLocalizationsFromArray:forPreferences:` (page 172)
  Returns the localizations that a bundle object would prefer, given the specified bundle and user preference localizations.
- `localizations` (page 179)
  Returns a list of all the localizations contained within the receiver's bundle.
- `developmentLocalization` (page 175)
  Returns the localization used to create the bundle.
- `preferredLocalizations` (page 186)
  Returns one or more localizations contained in the receiver's bundle that the receiver uses to locate resources based on the user's preferences.
- `localizedInfoDictionary` (page 179)
  Returns a dictionary with the keys from the bundle's localized property list.

Class Methods

allBundles

Returns an array of all the application's non-framework bundles.

```
+ (NSArray *)allBundles
```

Return Value
An array of all the application's non-framework bundles.

Discussion
The returned array includes the main bundle and all bundles that have been dynamically created but doesn't contain any bundles that represent frameworks.

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

Declared In
`NSBundle.h`

allFrameworks

Returns an array of all of the application's bundles that represent frameworks.

```
+ (NSArray *)allFrameworks
```

Return Value
An array of all of the application's bundles that represent frameworks. Only frameworks with one or more Objective-C classes in them are included.

Discussion
The returned array includes frameworks that are linked into an application when the application is built and bundles for frameworks that have been dynamically created.

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

Related Sample Code
Core Data HTML Store
CoreRecipes

Declared In
`NSBundle.h`

bundleForClass:

Returns the `NSBundle` object with which a given class is associated.

```
+ (NSBundle *)bundleForClass:(Class)aClass
```

Parameters

aClass

A class.

Return Value

The `NSBundle` object that dynamically loaded *aClass* (a loadable bundle), the `NSBundle` object for the framework in which *aClass* is defined, or the main bundle object if *aClass* was not dynamically loaded or is not defined in a framework.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ mainBundle` (page 169)

`+ bundleWithPath:` (page 169)

Related Sample Code

BundleLoader

CIAnnotation

Core Data HTML Store

CoreRecipes

GLSLShowpiece

Declared In

`NSBundle.h`

bundleWithIdentifier:

Returns the previously created `NSBundle` instance that has a given bundle identifier.

```
+ (NSBundle *)bundleWithIdentifier:(NSString *)identifier
```

Parameters

identifier

The identifier for an existing `NSBundle` instance.

Return Value

The previously created `NSBundle` instance that has the bundle identifier *identifier*. Returns `nil` if the requested bundle is not found.

Discussion

This method is typically used by frameworks and plug-ins to locate their own bundle at runtime. This method may be somewhat more efficient than trying to locate the bundle using the `bundleForClass:` (page 167) method.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

JavaSplashScreen

PrefsPane

Declared In

`NSBundle.h`

bundleWithPath:

Returns an `NSBundle` object that corresponds to the specified directory.

```
+ (NSBundle *)bundleWithPath:(NSString *)fullPath
```

Parameters

fullPath

The path to a directory. This must be a full pathname for a directory; if it contains any symbolic links, they must be resolvable.

Return Value

The `NSBundle` object that corresponds to *fullPath*, or `nil` if *fullPath* does not identify an accessible bundle directory.

Discussion

This method allocates and initializes the returned object if there is no existing `NSBundle` associated with *fullPath*, in which case it returns the existing object.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ mainBundle` (page 169)
`+ bundleForClass:` (page 167)
`- initWithPath:` (page 176)

Related Sample Code

BundleLoader
Core Data HTML Store

Declared In

`NSBundle.h`

mainBundle

Returns the `NSBundle` object that corresponds to the directory where the current application executable is located.

```
+ (NSBundle *)mainBundle
```

Return Value

The `NSBundle` object that corresponds to the directory where the application executable is located, or `nil` if a bundle object could not be created.

Discussion

This method allocates and initializes a bundle object if one doesn't already exist. The new object corresponds to the directory where the application executable is located. Be sure to check the return value to make sure you have a valid bundle. This method may return a valid bundle object even for unbundled applications.

In general, the main bundle corresponds to an application file package or application wrapper: a directory that bears the name of the application and is marked by a "`.app`" extension.

Availability

Available in Mac OS X v10.0 and later.

See Also
+ bundleForClass: (page 167)
+ bundleWithPath: (page 169)

Related Sample Code
CITransitionSelectorSample2
CoreRecipes
NewsReader
NumberInput_IMKit_Sample
StickiesExample

Declared In
NSBundle.h

pathForResource:ofType:inDirectory:

Returns the full pathname for the resource file identified by a given name and extension and residing in a given bundle directory.

```
+ (NSString *)pathForResource:(NSString *)name ofType:(NSString *)extension
    inDirectory:(NSString *)bundlePath
```

Parameters

name

The name of a resource file contained in the bundle specified by *bundlePath*.

extension

If *extension* is an empty string or nil, the returned pathname is the first one encountered that exactly matches *name*.

bundlePath

The path of a top-level bundle directory. This must be a valid path. For example, to specify the bundle directory for an application, you might specify the path /Applications/MyApp.app.

Return Value
The full pathname for the resource file or nil if the file could not be located. This method also returns nil if the bundle specified by the bundlePath parameter does not exist or is not a readable directory.

Discussion
The method first looks for a matching resource file in the nonlocalized resource directory (typically Resources) of the specified bundle. If a matching resource file is not found, it then looks in the top level of any available language-specific ".lproj" directories. (The search order for the language-specific directories corresponds to the user's preferences.) It does not recurse through other subdirectories at any of these locations. For more details see Bundles and Localization.

Note: This method is best suited only for the occasional retrieval of resource files. In most cases where you need to retrieve bundle resources, it is preferable to use the NSBundle instance methods instead.

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

See Also
- localizedStringForKey:value:table: (page 180)

- pathForResource:ofType: (page 182)
- pathForResource:ofType:inDirectory: (page 183)
+ pathsForResourcesOfType:inDirectory: (page 171)
- pathsForResourcesOfType:inDirectory: (page 185)

Declared In
NSBundle.h

pathsForResourcesOfType:inDirectory:

Returns an array containing the pathnames for all bundle resources having a given extension and residing in the bundle directory specified by a given path.

```
+ (NSArray *)pathsForResourcesOfType:(NSString *)extension inDirectory:(NSString
    *)bundlePath
```

Parameters

extension

If *extension* is an empty string or nil, all bundle resources in the top-level resource directories are returned.

bundlePath

The top-level directory of a bundle. This must represent a valid path.

Return Value
An array containing the full pathnames for all bundle resources with the specified extension. This method returns an empty array of no matching resource files are found. It also returns an empty array if the bundle specified by the bundlePath parameter does not exist or is not a readable directory.

Discussion
This method provides a means for dynamically discovering multiple bundle resources of the same type.

The method first looks for matching resource files in the nonlocalized resource directory (typically Resources) of the specified bundle. It then looks in the top level of any available language-specific ".lproj" directories. It does not recurse through other subdirectories at any of these locations. For more details see Bundles and Localization.

Note: This method is best suited only for the occasional retrieval of resource files. In most cases where you need to retrieve bundle resources, it is preferable to use the NSBundle instance methods instead.

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

See Also
- localizedStringForKey:value:table: (page 180)
- pathForResource:ofType: (page 182)
- pathForResource:ofType:inDirectory: (page 183)
+ pathForResource:ofType:inDirectory: (page 170)

Declared In
NSBundle.h

preferredLocalizationsFromArray:

Returns one or more localizations from the specified list that a bundle object would use to locate resources for the current user.

```
+ (NSArray *)preferredLocalizationsFromArray:(NSArray *)localizationsArray
```

Parameters

localizationsArray

An array of `NSString` objects, each of which specifies the name of a localization that the bundle supports.

Return Value

An array of `NSString` objects containing the preferred localizations. These strings are ordered in the array according to the current user's language preferences and are taken from the strings in the *localizationsArray* parameter.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSBundle.h`

preferredLocalizationsFromArray:forPreferences:

Returns the localizations that a bundle object would prefer, given the specified bundle and user preference localizations.

```
+ (NSArray *)preferredLocalizationsFromArray:(NSArray *)localizationsArray
    forPreferences:(NSArray *)preferencesArray
```

Parameters

localizationsArray

An array of `NSString` objects, each of which specifies the name of a localization that the bundle supports.

preferencesArray

An array of `NSString` objects containing the user's preferred localizations. If this parameter is `nil`, the method uses the current user's localization preferences.

Return Value

An array of `NSString` objects containing the preferred localizations. These strings are ordered in the array according to the specified preferences and are taken from the strings in the *localizationsArray* parameter.

Discussion

Use the argument *localizationsArray* to specify the supported localizations of the bundle and use *preferencesArray* to specify the user's localization preferences.

If none of the user-preferred localizations are available in the bundle, this method chooses one of the bundle localizations and returns it.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`NSBundle.h`

Instance Methods

builtInPlugInsPath

Returns the full pathname of the receiver's subdirectory containing plug-ins.

```
- (NSString *)builtInPlugInsPath
```

Return Value
The full pathname of the receiving bundle's subdirectory containing plug-ins.

Discussion
This method returns the appropriate path for modern application and framework bundles. This method may not return a path for non-standard bundle formats or for some older bundle formats.

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

Related Sample Code
BundleLoader
CIAnnotation
Core Data HTML Store

Declared In
`NSBundle.h`

bundleIdentifier

Returns the receiver's bundle identifier.

```
- (NSString *)bundleIdentifier
```

Return Value
The receiver's bundle identifier, which is defined by the `CFBundleIdentifier` key in the bundle's information property list.

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

See Also
`- infoDictionary` (page 176)

Related Sample Code
CoreRecipes
MungSaver
NumberInput_IMKit_Sample

Declared In
`NSBundle.h`

bundlePath

Returns the full pathname of the receiver's bundle directory.

```
- (NSString *)bundlePath
```

Return Value
The full pathname of the receiver's bundle directory.

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

Related Sample Code
JavaSplashScreen
NSGLImage

Declared In
`NSBundle.h`

classNamed:

Returns the `Class` object for the specified name.

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

Parameters
`className`
The name of a class.

Return Value
The `Class` object for *`className`*. Returns `NIL` if *`className`* is not one of the classes associated with the receiver or if there is an error loading the executable code containing the class implementation.

Discussion
If the bundle's executable code is not yet loaded, this method dynamically loads it into memory. Classes (and categories) are loaded from just one file within the bundle directory; this code file has the same name as the directory, but without the extension ("`.bundle`", "`.app`", "`.framework`"). As a side effect of code loading, the receiver posts `NSBundleDidLoadNotification` (page 192) after all classes and categories have been loaded; see "Notifications" (page 192) for details.

The following example loads a bundle's executable code containing the class "`FaxWatcher`":

```
- (void)loadBundle:(id)sender
{
    Class exampleClass;
    id newInstance;
    NSString *str = @"~/BundleExamples/BundleExample.bundle";
    NSBundle *bundleToLoad = [NSBundle bundleWithPath:str];
    if (exampleClass = [bundleToLoad classNamed:@"FaxWatcher"])  {
        newInstance = [[exampleClass alloc] init];
    // [newInstance doSomething];
    }
}
```

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

See Also
- `principalClass` (page 187)
- `load` (page 177)

Declared In
`NSBundle.h`

developmentLocalization

Returns the localization used to create the bundle.

```
- (NSString *)developmentLocalization
```

Return Value
The localization used to create the bundle.

Discussion
The returned localization corresponds to the value in the `CFBundleDevelopmentRegion` key of the bundle's property list (`Info.plist`).

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

Declared In
`NSBundle.h`

executableArchitectures

Returns an array of numbers indicating the architecture types supported by the bundle's executable.

```
- (NSArray *)executableArchitectures
```

Return Value
An array of `NSNumber` objects, each of which contains an integer value corresponding to a supported processor architecture. For a list of common architecture types, see the constants in "Mach-O Architecture" (page 191). If the bundle does not contain a Mach-O executable, this method returns `nil`.

Discussion
This method scans the bundle's Mach-O executable and returns all of the architecture types it finds. Because they are taken directly from the executable, the returned values may not always correspond to one of the well-known CPU types defined in "Mach-O Architecture" (page 191).

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

Declared In
`NSBundle.h`

executablePath

Returns the full pathname of the receiver's executable file.

```
- (NSString *)executablePath
```

Return Value
The full pathname of the receiving bundle's executable file.

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

Declared In
`NSBundle.h`

infoDictionary

Returns a dictionary that contains information about the receiver.

```
- (NSDictionary *)infoDictionary
```

Return Value
A dictionary, constructed from the bundle's `Info.plist` file, that contains information about the receiver. If the bundle does not contain an `Info.plist` file, a valid dictionary is returned but this dictionary contains only private keys that are used internally by the `NSBundle` class.

Discussion
Common keys for accessing the values of the dictionary are `CFBundleIdentifier`, `NSMainNibFile`, and `NSPrincipalClass`.

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

See Also
`- principalClass` (page 187)

Related Sample Code
JavaSplashScreen
PrefsPane
VertexPerformanceTest

Declared In
`NSBundle.h`

initWithPath:

Returns an `NSBundle` object initialized to correspond to a given directory.

```
- (id)initWithPath:(NSString *)fullPath
```

Parameters
fullPath
The path to a directory. This must be a full pathname for a directory; if it contains any symbolic links, they must be resolvable.

Return Value
An `NSBundle` object initialized to correspond to *`fullPath`*. This method initializes and returns a new instance only if there is no existing bundle associated with *`fullPath`*, otherwise it deallocates `self` and returns the existing object. If *`fullPath`* doesn't exist or the user doesn't have access to it, returns `nil`.

Discussion
It's not necessary to allocate and initialize an instance for the main bundle; use the `mainBundle` (page 169) class method to get this instance. You can also use the `bundleWithPath:` (page 169) class method to obtain a bundle identified by its directory path.

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

See Also
`+ bundleForClass:` (page 167)

Declared In
`NSBundle.h`

isLoaded

Obtains information about the load status of a bundle.

```
- (BOOL)isLoaded
```

Return Value
`YES` if the bundle's code is currently loaded, otherwise `NO`.

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

See Also
`- load` (page 177)

Declared In
`NSBundle.h`

load

Dynamically loads the bundle's executable code into a running program, if the code has not already been loaded.

```
- (BOOL)load
```

Return Value
`YES` if the method successfully loads the bundle's code or if the code has already been loaded, otherwise `NO`.

Discussion
You can use this method to load the code associated with a dynamically loaded bundle, such as a plug-in or framework. Prior to Mac OS X version 10.5, a bundle would attempt to load its code—if it had any—only once. Once loaded, you could not unload that code. In Mac OS X version 10.5 and later, you can unload a bundle's executable code using the `unload` (page 190) method.

You don't need to load a bundle's executable code to search the bundle's resources.

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

See Also
- `- loadAndReturnError:` (page 178)
- `- isLoaded` (page 177)
- `- unload` (page 190)
- `- classNamed:` (page 174)
- `- principalClass` (page 187)

Related Sample Code
Core Data HTML Store

Declared In
`NSBundle.h`

loadAndReturnError:

Loads the bundle's executable code and returns any errors.

```
- (BOOL)loadAndReturnError:(NSError **)error
```

Parameters

error

On input, a pointer to an error object variable. On output, this variable may contain an error object indicating why the bundle's executable could not be loaded. If no error occurred, this parameter is left unmodified. You may specify `nil` for this parameter if you are not interested in the error information.

Return Value
`YES` if the bundle's executable code was loaded successfully or was already loaded; otherwise, `NO` if the code could not be loaded.

Discussion
If this method returns `NO` and you pass a value for the *error* parameter, a suitable error object is returned in that parameter. Potential errors returned are in the Cocoa error domain and include the types that follow. For a full list of error types, see `FoundationErrors.h`.

- `NSFileNoSuchFileError` - returned if the bundle's executable file was not located.
- `NSExecutableNotLoadableError` - returned if the bundle's executable file exists but could not be loaded. This error is returned if the executable is not recognized as a loadable executable. It can also be returned if the executable is a PEF/CFM executable but the current process does not support that type of executable.
- `NSExecutableArchitectureMismatchError` - returned if the bundle executable does not include code that matches the processor architecture of the current processor.
- `NSExecutableRuntimeMismatchError` - returned if the bundle's required Objective-C runtime information is not compatible with the runtime of the current process.

- `NSExecutableLoadError` - returned if the bundle's executable failed to load for some detectable reason prior to linking. This error might occur if the bundle depends on a framework or library that is missing or if the required framework or library is not compatible with the current architecture or runtime version.
- `NSExecutableLinkError` - returned if the executable failed to load due to link errors but is otherwise alright.

The error object may contain additional debugging information in its description that you can use to identify the cause of the error. (This debugging information should not be displayed to the user.) You can obtain the debugging information by invoking the error object's `description` method in your code or by using the `print-object` command on the error object in gdb.

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

See Also
- `load` (page 177)
- `unload` (page 190)

Declared In
`NSBundle.h`

localizations

Returns a list of all the localizations contained within the receiver's bundle.

```
- (NSArray *)localizations
```

Return Value
An array, containing `NSString` objects, that specifies all the localizations contained within the receiver's bundle.

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

Declared In
`NSBundle.h`

localizedInfoDictionary

Returns a dictionary with the keys from the bundle's localized property list.

```
- (NSDictionary *)localizedInfoDictionary
```

Return Value
A dictionary with the keys from the bundle's localized property list (`InfoPlist.strings`).

Discussion
This method uses the preferred localization for the current user when determining which resources to return. If the preferred localization is not available, this method chooses the most appropriate localization found in the bundle.

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

Related Sample Code
PrefsPane

Declared In
NSBundle.h

localizedStringForKey:value:table:

Returns a localized version of the string designated by a given key in a given table.

```
- (NSString *)localizedStringForKey:(NSString *)key value:(NSString *)value
    table:(NSString *)tableName
```

Parameters

key
The key for a string in the table identified by *tableName*.

value
The value to return if *key* is nil or if a localized string for *key* can't be found in the table.

tableName
The receiver's string table to search. If *tableName* is nil or is an empty string, the method attempts to use the table in Localizable.strings.

Return Value
A localized version of the string designated by *key* in table *tableName*. If *value* is nil or an empty string, and a localized string is not found in the table, returns *key*. If *key* and *value* are both nil, returns the empty string.

Discussion
For more details about string localization and the specification of a .strings file, see "Working With Localized Strings."

Using the user default NSShowNonLocalizedStrings, you can alter the behavior of localizedStringForKey:value:table: (page 180) to log a message when the method can't find a localized string. If you set this default to YES (in the global domain or in the application's domain), then when the method can't find a localized string in the table, it logs a message to the console and capitalizes *key* before returning it.

The following example cycles through a static array of keys when a button is clicked, gets the value for each key from a strings table named Buttons.strings, and sets the button title with the returned value:

```
- (void)changeTitle:(id)sender
{
    static int keyIndex = 0;
    NSBundle *thisBundle = [NSBundle bundleForClass:[self class]];

    NSString *locString = [thisBundle
        localizedStringForKey:assortedKeys[keyIndex++]
        value:@"No translation" table:@"Buttons"];
    [sender setTitle:locString];
    if (keyIndex == MAXSTRINGS) keyIndex=0;
}
```

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

See Also
- `- pathForResource:ofType:` (page 182)
- `- pathForResource:ofType:inDirectory:` (page 183)
- `- pathsForResourcesOfType:inDirectory:` (page 185)
- `+ pathForResource:ofType:inDirectory:` (page 170)
- `+ pathsForResourcesOfType:inDirectory:` (page 171)

Related Sample Code
BundleLoader
CocoaDVDPlayer
InstallerPluginSample
NewsReader
Sketch-112

Declared In
`NSBundle.h`

objectForInfoDictionaryKey:

Returns the value associated with a given key in the receiver's property list.

```
- (id)objectForInfoDictionaryKey:(NSString *)key
```

Parameters

key

A key in the receiver's property list.

Return Value
The value associated with *key* in the receiver's property list (`Info.plist`). The localized value of a key is returned when one is available.

Discussion
Use of this method is preferred over other access methods because it returns the localized value of a key when one is available.

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

Related Sample Code
AutoUpdater
BundleLoader
FancyAbout
GridCalendar

Declared In
`NSBundle.h`

pathForAuxiliaryExecutable:

Returns the full pathname of the executable with a given name in the receiver's bundle.

```
- (NSString *)pathForAuxiliaryExecutable:(NSString *)executableName
```

Parameters

executableName

The name of an executable file.

Return Value

The full pathname of the executable *executableName* in the receiver's bundle.

Discussion

This method returns the appropriate path for modern application and framework bundles. This method may not return a path for non-standard bundle formats or for some older bundle formats.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSBundle.h

pathForResource:ofType:

Returns the full pathname for the resource identified by a given name and specified file extension.

```
- (NSString *)pathForResource:(NSString *)name ofType:(NSString *)extension
```

Parameters

name

The name of the resource file.

extension

The file extension of a resource with the name *name*.

Return Value

The full pathname for the resource file or `nil` if the file could not be located.

Discussion

If *extension* is an empty string or `nil`, the returned pathname is the first one encountered where the file name exactly matches *name*.

The method first looks for a matching resource file in the nonlocalized resource directory (typically `Resources`) of the specified bundle. If a matching resource file is not found, it then looks in the top level of any available language-specific "`.lproj`" directories. (The search order for the language-specific directories corresponds to the user's preferences.) It does not recurse through other subdirectories at any of these locations. For more details see Bundles and Localization.

The following code fragment gets the path to a localized sound, creates an `NSSound` instance from it, and plays the sound.

```
NSString *soundPath;
NSSound *thisSound;
NSBundle *thisBundle = [NSBundle bundleForClass:[self class]];
if (soundPath = [thisBundle pathForResource:@"Hello" ofType:@"snd"])  {
```

```
    thisSound = [[[NSSound alloc] initFromSoundfile:soundPath] autorelease];
    [thisSound play];
}
```

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

See Also
- `- localizedStringForKey:value:table:` (page 180)
- `- pathForResource:ofType:` (page 182)
- `- pathForResource:ofType:inDirectory:` (page 183)
- `+ pathForResource:ofType:inDirectory:` (page 170)
- `+ pathsForResourcesOfType:inDirectory:` (page 171)

Related Sample Code
AttachAScript
CIAnnotation
CITransitionSelectorSample2
GLSLShowpiece
StickiesExample

Declared In
`NSBundle.h`

pathForResource:ofType:inDirectory:

Returns the full pathname for the resource identified by the given name and file extension and located in the specified bundle subdirectory.

```
- (NSString *)pathForResource:(NSString *)name ofType:(NSString *)extension
    inDirectory:(NSString *)subpath
```

Parameters

name
The name of the resource file.

extension
The file extension of the specified resource file.

subpath
The name of the bundle subdirectory.

Return Value
The full pathname for the resource file or `nil` if the file could not be located.

Discussion
If *extension* is an empty string or `nil`, the returned pathname is the first one encountered where the file name exactly matches *name*.

If *subpath* is `nil`, this method searches the top-level nonlocalized resource directory (typically `Resources`) and the top-level of any language-specific directories. For example, suppose you have a modern bundle and specify `@"Documentation"` for the *subpath* parameter. This method would first look in the `Contents/Resources/Documentation` directory of the bundle, followed by the `Documentation`

subdirectories of each language-specific `.lproj` directory. (The search order for the language-specific directories corresponds to the user's preferences.) This method does not recurse through any other subdirectories at any of these locations. For more details see Bundles and Localization.

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

See Also
- `localizedStringForKey:value:table:` (page 180)
- `pathForResource:ofType:` (page 182)
- `pathsForResourcesOfType:inDirectory:` (page 185)
+ `pathForResource:ofType:inDirectory:` (page 170)
+ `pathsForResourcesOfType:inDirectory:` (page 171)

Declared In
`NSBundle.h`

pathForResource:ofType:inDirectory:forLocalization:

Returns the full pathname for the resource identified by the given name and file extension, located in the specified bundle subdirectory, and limited to global resources and those associated with the specified localization.

```
- (NSString *)pathForResource:(NSString *)name ofType:(NSString *)extension
    inDirectory:(NSString *)subpath forLocalization:(NSString *)localizationName
```

Parameters

name
The name of the resource file.

extension
The file extension of the specified resource file.

subpath
The name of the bundle subdirectory to search.

localizationName
The name of the localization. This parameter should correspond to the name of one of the bundle's language-specific resource directories without the `.lproj` extension.

Return Value
The full pathname for the resource file or `nil` if the file could not be located.

Discussion
This method is equivalent to `pathForResource:ofType:inDirectory:` (page 183), except that only nonlocalized resources and those in the language-specific `.lproj` directory specified by *localizationName* are searched.

There should typically be little reason to use this method—see Getting the Current Language and Locale. See also preferredLocalizationsFromArray:forPreferences: (page 172) for how to determine what localizations are available.

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

Declared In
NSBundle.h

pathsForResourcesOfType:inDirectory:

Returns an array containing the pathnames for all bundle resources having the specified filename extension and residing in the resource subdirectory.

```
- (NSArray *)pathsForResourcesOfType:(NSString *)extension inDirectory:(NSString
    *)subpath
```

Parameters

extension
The file extension of the files to retrieve.

subpath
The name of the bundle subdirectory to search.

Return Value
An array containing the full pathnames for all bundle resources matching the specified criteria. This method returns an empty array of no matching resource files are found.

Discussion
This method provides a means for dynamically discovering multiple bundle resources of the same type. If *extension* is an empty string or nil, all bundle resources in the specified resource directory are returned.

The argument *subpath* specifies the name of a specific subdirectory to search within the current bundle's resource directory hierarchy. If *subpath* is nil, this method searches the top-level nonlocalized resource directory (typically Resources) and the top-level of any language-specific directories. For example, suppose you have a modern bundle and specify @"Documentation" for the *subpath* parameter. This method would first look in the Contents/Resources/Documentation directory of the bundle, followed by the Documentation subdirectories of each language-specific .lproj directory. (The search order for the language-specific directories corresponds to the user's preferences.) This method does not recurse through any other subdirectories at any of these locations. For more details see Bundles and Localization.

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

See Also
- localizedStringForKey:value:table: (page 180)
- pathForResource:ofType: (page 182)
- pathForResource:ofType:inDirectory: (page 183)
+ pathForResource:ofType:inDirectory: (page 170)
+ pathsForResourcesOfType:inDirectory: (page 171)

Related Sample Code
AutoSample
CocoaCreateMovie
QTKitCreateMovie

Declared In
NSBundle.h

pathsForResourcesOfType:inDirectory:forLocalization:

Returns an array containing the pathnames for all bundle resources having the specified filename extension, residing in the specified resource subdirectory, and limited to global resources and those associated with the specified localization.

```
- (NSArray *)pathsForResourcesOfType:(NSString *)extension inDirectory:(NSString
    *)subpath forLocalization:(NSString *)localizationName
```

Parameters

extension
The file extension of the files to retrieve.

subpath
The name of the bundle subdirectory to search.

localizationName
The name of the localization. This parameter should correspond to the name of one of the bundle's language-specific resource directories without the `.lproj` extension.

Return Value
An array containing the full pathnames for all bundle resources matching the specified criteria. This method returns an empty array of no matching resource files are found.

Discussion
This method is equivalent to `pathsForResourcesOfType:inDirectory:` (page 185), except that only nonlocalized resources and those in the language-specific `.lproj` directory specified by *localizationName* are searched.

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

Declared In
`NSBundle.h`

preferredLocalizations

Returns one or more localizations contained in the receiver's bundle that the receiver uses to locate resources based on the user's preferences.

```
- (NSArray *)preferredLocalizations
```

Return Value
One or more localizations contained in the receiver's bundle that the receiver uses to locate resources based on the user's preferences.

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

See Also
`+ preferredLocalizationsFromArray:` (page 172)
`- localizations` (page 179)

Declared In
`NSBundle.h`

preflightAndReturnError:

Returns a Boolean value indicating whether the bundle's executable code could be loaded successfully.

```
- (BOOL)preflightAndReturnError:(NSError **)error
```

Parameters

error

On input, a pointer to an error object variable. On output, this variable may contain an error object indicating why the bundle's executable could not be loaded. If no error would occur, this parameter is left unmodified. You may specify `nil` for this parameter if you are not interested in the error information.

Return Value

`YES` if the bundle's executable code could be loaded successfully or is already loaded; otherwise, `NO` if the code could not be loaded.

Discussion

This method does not actually load the bundle's executable code. Instead, it performs several checks to see if the code could be loaded and with one exception returns the same errors that would occur during an actual load operation. The one exception is the `NSExecutableLinkError` error, which requires the actual loading of the code to verify link errors.

For a list of possible load errors, see the discussion for the `loadAndReturnError:` (page 178) method.

Availability

Available in Mac OS X v10.5 and later.

See Also

`- loadAndReturnError:` (page 178)

Declared In

`NSBundle.h`

principalClass

Returns the receiver's principal class.

```
- (Class)principalClass
```

Return Value

The receiver's principal class—after ensuring that the code containing the definition of that class is dynamically loaded. If the receiver encounters errors in loading or if it can't find the executable code file in the bundle directory, returns `NIL`.

Discussion

The principal class typically controls all the other classes in the bundle; it should mediate between those classes and classes external to the bundle. Classes (and categories) are loaded from just one file within the bundle directory. `NSBundle` obtains the name of the code file to load from the dictionary returned from `infoDictionary` (page 176), using "`NSExecutable`" as the key. The bundle determines its principal class in one of two ways:

- It first looks in its own information dictionary, which extracts the information encoded in the bundle's property list (`Info.plist`). `NSBundle` obtains the principal class from the dictionary using the key `NSPrincipalClass`. For nonloadable bundles (applications and frameworks), if the principal class is not specified in the property list, the method returns `NIL`.
- If the principal class is not specified in the information dictionary, `NSBundle` identifies the first class loaded as the principal class. When several classes are linked into a dynamically loadable file, the default principal class is the first one listed on the `ld` command line. In the following example, Reporter would be the principal class:

```
ld -o myBundle -r Reporter.o NotePad.o QueryList.o
```

The order of classes in Xcode's project browser is the order in which they will be linked. To designate the principal class, control-drag the file containing its implementation to the top of the list.

As a side effect of code loading, the receiver posts `NSBundleDidLoadNotification` (page 192) after all classes and categories have been loaded; see "Notifications" (page 192) for details.

The following method obtains a bundle by specifying its path (`bundleWithPath:` (page 169)), then loads the bundle with `principalClass` (page 187) and uses the returned class object to allocate and initialize an instance of that class:

```
- (void)loadBundle:(id)sender
{
    Class exampleClass;
    id newInstance;
    NSString *path = @"/tmp/Projects/BundleExample/BundleExample.bundle";
    NSBundle *bundleToLoad = [NSBundle bundleWithPath:path];
    if (exampleClass = [bundleToLoad principalClass]) {
        newInstance = [[exampleClass alloc] init];
        [newInstance doSomething];
    }
}
```

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

See Also
- `- classNamed:` (page 174)
- `- infoDictionary` (page 176)
- `- load` (page 177)

Related Sample Code
BundleLoader

Declared In
`NSBundle.h`

privateFrameworksPath

Returns the full pathname of the receiver's subdirectory containing private frameworks.

```
- (NSString *)privateFrameworksPath
```

Return Value
The full pathname of the receiver's subdirectory containing private frameworks.

Discussion
This method returns the appropriate path for modern application and framework bundles. This method may not return a path for non-standard bundle formats or for some older bundle formats.

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

Related Sample Code
MP3 Player

Declared In
`NSBundle.h`

resourcePath

Returns the full pathname of the receiving bundle's subdirectory containing resources.

```
- (NSString *)resourcePath
```

Return Value
The full pathname of the receiving bundle's subdirectory containing resources.

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

See Also
`- bundlePath` (page 174)

Related Sample Code
NURBSSurfaceVertexProg
StickiesExample
SurfaceVertexProgram
TextureRange
VertexPerformanceDemo

Declared In
`NSBundle.h`

sharedFrameworksPath

Returns the full pathname of the receiver's subdirectory containing shared frameworks.

```
- (NSString *)sharedFrameworksPath
```

Return Value
The full pathname of the receiver's subdirectory containing shared frameworks.

Discussion
This method returns the appropriate path for modern application and framework bundles. This method may not return a path for non-standard bundle formats or for some older bundle formats.

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

Declared In
`NSBundle.h`

sharedSupportPath

Returns the full pathname of the receiver's subdirectory containing shared support files.

```
- (NSString *)sharedSupportPath
```

Return Value
The full pathname of the receiver's subdirectory containing shared support files.

Discussion
This method returns the appropriate path for modern application and framework bundles. This method may not return a path for non-standard bundle formats or for some older bundle formats.

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

Declared In
`NSBundle.h`

unload

Unloads the code associated with the receiver.

```
- (BOOL)unload
```

Return Value
`YES` if the bundle was successfully unloaded or was not already loaded; otherwise, `NO` if the bundle could not be unloaded.

Discussion
This method attempts to unload a bundle's executable code using the underlying dynamic loader (typically `dyld`). You may use this method to unload plug-in and framework bundles when you no longer need the code they contain. You should use this method to unload bundles that were loaded using the methods of the `NSBundle` class only. Do not use this method to unload bundles that were originally loaded using the bundle-manipulation functions in Core Foundation.

It is the responsibility of the caller to ensure that no in-memory objects or data structures refer to the code being unloaded. For example, if you have an object whose class is defined in a bundle, you must release that object prior to unloading the bundle. Similarly, your code should not attempt to access any symbols defined in an unloaded bundle.

Special Considerations

Prior to Mac OS X version 10.5, code could not be unloaded once loaded, and this method would always return `NO`. In Mac OS X version 10.5 and later, you can unload a bundle's executable code using this method.

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

See Also
- `loadAndReturnError:` (page 178)
- `load` (page 177)

Declared In
`NSBundle.h`

Constants

Mach-O Architecture

These constants describe the CPU types that a bundle's executable code may support.

```
enum {
    NSBundleExecutableArchitectureI386      = 0x00000007,
    NSBundleExecutableArchitecturePPC       = 0x00000012,
    NSBundleExecutableArchitectureX86_64    = 0x01000007,
    NSBundleExecutableArchitecturePPC64     = 0x01000012
};
```

Constants

`NSBundleExecutableArchitectureI386`
Specifies the 32-bit Intel architecture.
Available in Mac OS X v10.5 and later.
Declared in `NSBundle.h`.

`NSBundleExecutableArchitecturePPC`
Specifies the 32-bit PowerPC architecture.
Available in Mac OS X v10.5 and later.
Declared in `NSBundle.h`.

`NSBundleExecutableArchitectureX86_64`
Specifies the 64-bit Intel architecture.
Available in Mac OS X v10.5 and later.
Declared in `NSBundle.h`.

`NSBundleExecutableArchitecturePPC64`
Specifies the 64-bit PowerPC architecture.
Available in Mac OS X v10.5 and later.
Declared in `NSBundle.h`.

Declared In
`NSBundle.h`

Notifications

NSBundleDidLoadNotification

`NSBundle` posts `NSBundleDidLoadNotification` to notify observers which classes and categories have been dynamically loaded. When a request is made to an `NSBundle` object for a class (`classNamed:` (page 174) or `principalClass` (page 187)), the bundle dynamically loads the executable code file that contains the class implementation and all other class definitions contained in the file. After the module is loaded, the bundle posts the `NSBundleDidLoadNotification`.

The notification object is the `NSBundle` instance that dynamically loads classes. The *`userInfo`* dictionary contains the following information:

Key	Value
`@"NSLoadedClasses"`	An `NSArray` object containing the names (as `NSString` objects) of each class that was loaded

In a typical use of this notification, an object might want to enumerate the *`userInfo`* array to check if each loaded class conformed to a certain protocol (say, an protocol for a plug-and-play tool set); if a class does conform, the object would create an instance of that class and add the instance to another `NSArray` object.

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

Declared In
`NSBundle.h`

NSCachedURLResponse Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSURLCache.h
Availability	Available in Mac OS X v10.2 with Safari 1.0 installed. Available in Mac OS X v10.2.7 and later.
Companion guide	URL Loading System
Related sample code	URL CacheInfo

Overview

An `NSCachedURLResponse` object encapsulates an `NSURLResponse` object, an `NSData` object containing the content corresponding to the response, and an `NSDictionary` containing application specific information.

The `NSURLCache` system stores and retrieves instances of `NSCachedURLResponse`.

Tasks

Creating a Cached URL Response

- `initWithResponse:data:` (page 194)
  Initializes an `NSCachedURLResponse` object.
- `initWithResponse:data:userInfo:storagePolicy:` (page 195)
  Initializes an NSCachedURLResponse object.

Getting Cached URL Response Properties

- `data` (page 194)
  Returns the receiver's cached data.

- `response` (page 195)
  Returns the `NSURLResponse` object associated with the receiver.
- `storagePolicy` (page 196)
  Returns the receiver's cache storage policy.
- `userInfo` (page 196)
  Returns the receiver's user info dictionary.

Instance Methods

data

Returns the receiver's cached data.

```
- (NSData *)data
```

Return Value
The receiver's cached data.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Related Sample Code
URL CacheInfo

Declared In
`NSURLCache.h`

initWithResponse:data:

Initializes an `NSCachedURLResponse` object.

```
- (id)initWithResponse:(NSURLResponse *)response data:(NSData *)data
```

Parameters
response
  The response to cache.
data
  The data to cache.

Return Value
The `NSCachedURLResponse` object, initialized using the given data.

Discussion
The cache storage policy is set to the default, `NSURLCacheStorageAllowed`, and the user info dictionary is set to `nil`.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

See Also
- `initWithResponse:data:userInfo:storagePolicy:` (page 195)

Declared In
`NSURLCache.h`

initWithResponse:data:userInfo:storagePolicy:

Initializes an NSCachedURLResponse object.

```
- (id)initWithResponse:(NSURLResponse *)response data:(NSData *)data
    userInfo:(NSDictionary *)userInfo
    storagePolicy:(NSURLCacheStoragePolicy)storagePolicy
```

Parameters

response
The response to cache.

data
The data to cache.

userInfo
An optional dictionary of user information. May be `nil`.

storagePolicy
The storage policy for the cached response.

Return Value
The `NSCachedURLResponse` object, initialized using the given data.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- `initWithResponse:data:` (page 194)

Related Sample Code
URL CacheInfo

Declared In
`NSURLCache.h`

response

Returns the `NSURLResponse` object associated with the receiver.

```
- (NSURLResponse *)response
```

Return Value
The `NSURLResponse` object associated with the receiver.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Related Sample Code
URL CacheInfo

Declared In
`NSURLCache.h`

storagePolicy

Returns the receiver's cache storage policy.

```
- (NSURLCacheStoragePolicy)storagePolicy
```

Return Value
The receiver's cache storage policy.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Related Sample Code
URL CacheInfo

Declared In
`NSURLCache.h`

userInfo

Returns the receiver's user info dictionary.

```
- (NSDictionary *)userInfo
```

Return Value
An `NSDictionary` object containing the receiver's user info, or `nil` if there is no such object.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLCache.h`

Constants

NSURLCacheStoragePolicy

These constants specify the caching strategy used by an `NSCachedURLResponse` object.

```
typedef enum
{
    NSURLCacheStorageAllowed,
    NSURLCacheStorageAllowedInMemoryOnly,
    NSURLCacheStorageNotAllowed,
} NSURLCacheStoragePolicy;
```

Constants

`NSURLCacheStorageAllowed`

Specifies that storage in `NSURLCache` is allowed without restriction.

Important: iPhone OS ignores this cache policy, and instead treats it as `NSURLCacheStorageAllowedInMemoryOnly`.

Available in Mac OS X v10.2 and later.

Declared in `NSURLCache.h`.

`NSURLCacheStorageAllowedInMemoryOnly`

Specifies that storage in `NSURLCache` is allowed; however storage should be restricted to memory only.

Available in Mac OS X v10.2 and later.

Declared in `NSURLCache.h`.

`NSURLCacheStorageNotAllowed`

Specifies that storage in `NSURLCache` is not allowed in any fashion, either in memory or on disk.

Available in Mac OS X v10.2 and later.

Declared in `NSURLCache.h`.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

`NSURLCache.h`

NSCalendar Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.4 and later.
Declared in	Foundation/NSCalendar.h
Companion guides	Date and Time Programming Guide for Cocoa Data Formatting Programming Guide for Cocoa
Related sample code	Birthdays Mountains Reminders

Overview

Calendars encapsulate information about systems of reckoning time in which the beginning, length, and divisions of a year are defined. They provide information about the calendar and support for calendrical computations such as determining the range of a given calendrical unit and adding units to a given absolute time.

In a calendar, day, week, weekday, month, and year numbers are generally 1-based, but there may be calendar-specific exceptions. Ordinal numbers, where they occur, are 1-based. Some calendars represented by this API may have to map their basic unit concepts into year/month/week/day/… nomenclature. For example, a calendar composed of 4 quarters in a year instead of 12 months uses the month unit to represent quarters. The particular values of the unit are defined by each calendar, and are not necessarily consistent with values for that unit in another calendar.

To do calendar arithmetic, you use `NSDate` objects in conjunction with a calendar. For example, to convert between a decomposed date in one calendar and another calendar, you must first convert the decomposed elements into a date using the first calendar, then decompose it using the second. `NSDate` provides the absolute scale and epoch (reference point) for dates and times, which can then be rendered into a particular calendar, for calendrical computations or user display.

Two `NSCalendar` methods that return a date object, `dateFromComponents:` (page 206), `dateByAddingComponents:toDate:options:` (page 205), take as a parameter an `NSDateComponents` object that describes the calendrical components required for the computation. You can provide as many components as you need (or choose to). When there is incomplete information to compute an absolute time,

default values similar to `0` and `1` are usually chosen by a calendar, but this is a calendar-specific choice. If you provide inconsistent information, calendar-specific disambiguation is performed (which may involve ignoring one or more of the parameters). Related methods (`components:fromDate:` (page 203) and `components:fromDate:toDate:options:` (page 204)) take a bit mask parameter that specifies which components to calculate when returning an `NSDateComponents` object. The bit mask is composed of `NSCalendarUnit` constants (see "Constants" (page 213)).

`NSCalendar` is "toll-free bridged" with its Core Foundation counterpart, CFCalendar. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSCalendar *` parameter, you can pass in a `CFCalendarRef`, and in a function where you see a `CFCalendarRef` parameter, you can pass in an `NSCalendar` instance. See Interchangeable Data Types for more information on toll-free bridging.

Tasks

System Locale Information

`+ currentCalendar` (page 202)
Returns the logical calendar for the current user.

`+ autoupdatingCurrentCalendar` (page 201)
Returns the current logical calendar for the current user.

Initializing a Calendar

`- initWithCalendarIdentifier:` (page 207)
Initializes a newly-allocated `NSCalendar` object for the calendar specified by a given identifier.

`- setFirstWeekday:` (page 211)
Sets the index of the first weekday for the receiver.

`- setLocale:` (page 211)
Sets the locale for the receiver.

`- setMinimumDaysInFirstWeek:` (page 212)
Sets the minimum number of days in the first week of the receiver.

`- setTimeZone:` (page 212)
Sets the time zone for the receiver.

Getting Information About a Calendar

`- calendarIdentifier` (page 202)
Returns the identifier for the receiver.

`- firstWeekday` (page 207)
Returns the index of the first weekday of the receiver.

`- locale` (page 207)
Returns the locale for the receiver.

- `maximumRangeOfUnit:` (page 208)
  The maximum range limits of the values that a given unit can take on in the receive
- `minimumDaysInFirstWeek` (page 208)
  Returns the minimum number of days in the first week of the receiver.
- `minimumRangeOfUnit:` (page 209)
  Returns the minimum range limits of the values that a given unit can take on in the receiver.
- `ordinalityOfUnit:inUnit:forDate:` (page 209)
  Returns, for a given absolute time, the ordinal number of a smaller calendar unit (such as a day) within a specified larger calendar unit (such as a week).
- `rangeOfUnit:inUnit:forDate:` (page 210)
  Returns the range of absolute time values that a smaller calendar unit (such as a day) can take on in a larger calendar unit (such as a month) that includes a specified absolute time.
- `rangeOfUnit:startDate:interval:forDate:` (page 210)
  Returns by reference the starting time and duration of a given calendar unit that contains a given date.
- `timeZone` (page 213)
  Returns the time zone for the receiver.

Calendrical Calculations

- `components:fromDate:` (page 203)
  Returns a `NSDateComponents` object containing a given date decomposed into specified components.
- `components:fromDate:toDate:options:` (page 204)
  Returns, as an `NSDateComponents` object using specified components, the difference between two supplied dates.
- `dateByAddingComponents:toDate:options:` (page 205)
  Returns a new `NSDate` object representing the absolute time calculated by adding given components to a given date.
- `dateFromComponents:` (page 206)
  Returns a new `NSDate` object representing the absolute time calculated from given components.

Class Methods

autoupdatingCurrentCalendar

Returns the current logical calendar for the current user.

```
+ (id)autoupdatingCurrentCalendar
```

Return Value
The current logical calendar for the current user.

Discussion
Settings you get from this calendar do change as the user's settings change (contrast with `currentCalendar` (page 202)).

Note that if you cache values based on the calendar or related information those caches will of course not be automatically updated by the updating of the calendar object.

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

See Also
`+ currentCalendar` (page 202)
`- initWithCalendarIdentifier:` (page 207)
`- calendarIdentifier` (page 202)

Declared In
`NSCalendar.h`

currentCalendar

Returns the logical calendar for the current user.

```
+ (id)currentCalendar
```

Return Value
The logical calendar for the current user.

Discussion
The returned calendar is formed from the settings for the current user's chosen system locale overlaid with any custom settings the user has specified in System Preferences. Settings you get from this calendar do not change as System Preferences are changed, so that your operations are consistent (contrast with `autoupdatingCurrentCalendar` (page 201)).

Availability
Available in Mac OS X v10.4 and later.

See Also
`+ autoupdatingCurrentCalendar` (page 201)
`- initWithCalendarIdentifier:` (page 207)
`- calendarIdentifier` (page 202)

Declared In
`NSCalendar.h`

Instance Methods

calendarIdentifier

Returns the identifier for the receiver.

```
- (NSString *)calendarIdentifier
```

Return Value
The identifier for the receiver. For valid identifiers, see `NSLocale`.

Availability
Available in Mac OS X v10.4 and later.

See Also
`+ autoupdatingCurrentCalendar` (page 201)
`- initWithCalendarIdentifier:` (page 207)

Related Sample Code
Mountains

Declared In
`NSCalendar.h`

components:fromDate:

Returns a `NSDateComponents` object containing a given date decomposed into specified components.

```
- (NSDateComponents *)components:(NSUInteger)unitFlags fromDate:(NSDate *)date
```

Parameters

`unitFlags`
The components into which to decompose *`date`*—a bitwise `OR` of `NSCalendarUnit` constants.

`date`
The date for which to perform the calculation.

Return Value
An `NSDateComponents` object containing *`date`* decomposed into the components specified by *`unitFlags`*. Returns `nil` if *`date`* falls outside of the defined range of the receiver or if the computation cannot be performed

Discussion
The Weekday ordinality, when requested, refers to the next larger (than Week) of the requested units. Some computations can take a relatively long time.

The following example shows how to use this method to determine the current year, month, and day, using an existing calendar (`gregorian`):

```
unsigned unitFlags = NSYearCalendarUnit | NSMonthCalendarUnit |
NSDayCalendarUnit;
NSDate *date = [NSDate date];
NSDateComponents *comps = [gregorian components:unitFlags fromDate:date];
```

Availability
Available in Mac OS X v10.4 and later.

See Also
`- dateFromComponents:` (page 206)
`- components:fromDate:toDate:options:` (page 204)
`- dateByAddingComponents:toDate:options:` (page 205)

Related Sample Code
Birthdays

Declared In
NSCalendar.h

components:fromDate:toDate:options:

Returns, as an NSDateComponents object using specified components, the difference between two supplied dates.

```
- (NSDateComponents *)components:(NSUInteger)unitFlags fromDate:(NSDate
    *)startingDate toDate:(NSDate *)resultDate options:(NSUInteger)opts
```

Parameters

unitFlags
Specifies the components for the returned NSDateComponents object—a bitwise OR of NSCalendarUnit constants.

startingDate
The start date for the calculation.

resultDate
The end date for the calculation.

opts
Options for the calculation.

If you specify a "wrap" option (NSWrapCalendarComponents), the specified components are incremented and wrap around to zero/one on overflow, but do not cause higher units to be incremented. When the wrap option is false, overflow in a unit carries into the higher units, as in typical addition.

Return Value
An NSDateComponents object whose components are specified by *unitFlags* and calculated from the difference between the *resultDate* and *startDate* using the options specified by *opts*. Returns nil if either date falls outside the defined range of the receiver or if the computation cannot be performed.

Discussion
The result is lossy if there is not a small enough unit requested to hold the full precision of the difference. Some operations can be ambiguous, and the behavior of the computation is calendar-specific, but generally larger components will be computed before smaller components; for example, in the Gregorian calendar a result might be 1 month and 5 days instead of, for example, 0 months and 35 days. The resulting component values may be negative if *resultDate* is before *startDate*.

The following example shows how to get the approximate number of months and days between two dates using an existing calendar (gregorian):

```
NSDate *startDate = ...;
NSDate *endDate = ...;
unsigned int unitFlags = NSMonthCalendarUnit | NSDayCalendarUnit;
NSDateComponents *comps = [gregorian components:unitFlags fromDate:startDate
toDate:endDate  options:0];
int months = [comps month];
int days = [comps day];
```

Note that some computations can take a relatively long time.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `dateByAddingComponents:toDate:options:` (page 205)
- `dateFromComponents:` (page 206)

Declared In
`NSCalendar.h`

dateByAddingComponents:toDate:options:

Returns a new `NSDate` object representing the absolute time calculated by adding given components to a given date.

```
- (NSDate *)dateByAddingComponents:(NSDateComponents *)comps toDate:(NSDate *)date
    options:(NSUInteger)opts
```

Parameters

comps

The components to add to *date*.

date

The date to which *comps* are added.

opts

Options for the calculation. See "`NSDateComponents wrapping behavior`" (page 215) for possible values. Pass `0` to specify no options.

If you specify no options (you pass `0`), overflow in a unit carries into the higher units (as in typical addition).

Return Value
A new `NSDate` object representing the absolute time calculated by adding to *date* the calendrical components specified by *comps* using the options specified by *opts*. Returns `nil` if *date* falls outside the defined range of the receiver or if the computation cannot be performed.

Discussion
Some operations can be ambiguous, and the behavior of the computation is calendar-specific, but generally components are added in the order specified.

The following example shows how to add 2 months and 3 days to the current date and time using an existing calendar (`gregorian`):

```
NSDate *currentDate = [NSDate date];
NSDateComponents *comps = [[NSDateComponents alloc] init];
[comps setMonth:2];
[comps setDay:3];
NSDate *date = [gregorian dateByAddingComponents:comps toDate:currentDate
options:0];
[comps release];
```

Note that some computations can take a relatively long time.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `dateFromComponents:` (page 206)

- components:fromDate:toDate:options: (page 204)

Declared In
NSCalendar.h

dateFromComponents:

Returns a new NSDate object representing the absolute time calculated from given components.

- (NSDate *)**dateFromComponents:**(NSDateComponents *)*comps*

Parameters

comps

The components from which to calculate the returned date.

Return Value
A new NSDate object representing the absolute time calculated from *comps*. Returns nil if the receiver cannot convert the components given in *comps* into an absolute time. The method also returns nil and for out-of-range values.

Discussion
When there are insufficient components provided to completely specify an absolute time, a calendar uses default values of its choice. When there is inconsistent information, a calendar may ignore some of the components parameters or the method may return nil. Unnecessary components are ignored (for example, Day takes precedence over Weekday and Weekday ordinals).

The following example shows how to use this method to create a date object to represent 14:10:00 on 6 January 1965, for a given calendar (gregorian).

```
NSDateComponents *comps = [[NSDateComponents alloc] init];
[comps setYear:1965];
[comps setMonth:1];
[comps setDay:6];
[comps setHour:14];
[comps setMinute:10];
[comps setSecond:0];
NSDate *date = [gregorian dateFromComponents:comps];
[comps release];
```

Note that some computations can take a relatively long time to perform.

Availability
Available in Mac OS X v10.4 and later.

See Also
- components:fromDate: (page 203)
- dateFromComponents: (page 206)

Related Sample Code
Reminders

Declared In
NSCalendar.h

firstWeekday

Returns the index of the first weekday of the receiver.

```
- (NSUInteger)firstWeekday
```

Return Value
The index of the first weekday of the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setFirstWeekday:` (page 211)

Declared In
`NSCalendar.h`

initWithCalendarIdentifier:

Initializes a newly-allocated `NSCalendar` object for the calendar specified by a given identifier.

```
- (id)initWithCalendarIdentifier:(NSString *)string
```

Parameters
string
The identifier for the new calendar. For valid identifiers, see `NSLocale`.

Return Value
The initialized calendar, or `nil` if the identifier is unknown (if, for example, it is either an unrecognized string or the calendar is not supported by the current version of the operating system).

Availability
Available in Mac OS X v10.4 and later.

See Also
- `+ autoupdatingCurrentCalendar` (page 201)
- `- calendarIdentifier` (page 202)

Related Sample Code
Birthdays
Mountains
Reminders

Declared In
`NSCalendar.h`

locale

Returns the locale for the receiver.

```
- (NSLocale *)locale
```

Return Value
The locale for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setLocale:` (page 211)

Declared In
`NSCalendar.h`

maximumRangeOfUnit:

The maximum range limits of the values that a given unit can take on in the receive

```
- (NSRange)maximumRangeOfUnit:(NSCalendarUnit)unit
```

Parameters

unit

The unit for which the maximum range is returned.

Return Value
The maximum range limits of the values that the unit specified by *unit* can take on in the receiver.

Discussion
As an example, in the Gregorian calendar the maximum range of values for the Day unit is 1-31.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- minimumRangeOfUnit:` (page 209)

Declared In
`NSCalendar.h`

minimumDaysInFirstWeek

Returns the minimum number of days in the first week of the receiver.

```
- (NSUInteger)minimumDaysInFirstWeek
```

Return Value
The minimum number of days in the first week of the receiver

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setMinimumDaysInFirstWeek:` (page 212)

Declared In
`NSCalendar.h`

minimumRangeOfUnit:

Returns the minimum range limits of the values that a given unit can take on in the receiver.

```
- (NSRange)minimumRangeOfUnit:(NSCalendarUnit)unit
```

Parameters

unit

The unit for which the maximum range is returned.

Return Value

The minimum range limits of the values that the unit specified by *unit* can take on in the receiver.

Discussion

As an example, in the Gregorian calendar the minimum range of values for the Day unit is 1-28.

Availability

Available in Mac OS X v10.4 and later.

See Also

- maximumRangeOfUnit: (page 208)

Declared In

NSCalendar.h

ordinalityOfUnit:inUnit:forDate:

Returns, for a given absolute time, the ordinal number of a smaller calendar unit (such as a day) within a specified larger calendar unit (such as a week).

```
- (NSUInteger)ordinalityOfUnit:(NSCalendarUnit)smaller inUnit:(NSCalendarUnit)larger
    forDate:(NSDate *)date
```

Parameters

smaller

The smaller calendar unit

larger

The larger calendar unit

date

The absolute time for which the calculation is performed

Return Value

The ordinal number of *smaller* within *larger* at the time specified by *date*. Returns NSNotFound if *larger* is not logically bigger than *smaller* in the calendar, or the given combination of units does not make sense (or is a computation which is undefined).

Discussion

The ordinality is in most cases not the same as the decomposed value of the unit. Typically return values are 1 and greater. For example, the time 00:45 is in the first hour of the day, and for units Hour and Day respectively, the result would be 1. An exception is the week-in-month calculation, which returns 0 for days before the first week in the month containing the date.

Note that some computations can take a relatively long time.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `rangeOfUnit:inUnit:forDate:` (page 210)
- `rangeOfUnit:startDate:interval:forDate:` (page 210)

Declared In
`NSCalendar.h`

rangeOfUnit:inUnit:forDate:

Returns the range of absolute time values that a smaller calendar unit (such as a day) can take on in a larger calendar unit (such as a month) that includes a specified absolute time.

```
- (NSRange)rangeOfUnit:(NSCalendarUnit)smaller inUnit:(NSCalendarUnit)larger
    forDate:(NSDate *)date
```

Parameters

smaller
The smaller calendar unit.

larger
The larger calendar unit.

date
The absolute time for which the calculation is performed.

Return Value
The range of absolute time values *smaller* can take on in *larger* at the time specified by *date*. Returns {NSNotFound, NSNotFound} if *larger* is not logically bigger than *smaller* in the calendar, or the given combination of units does not make sense (or is a computation which is undefined).

Discussion
You can use this method to calculate, for example, the range the Day unit can take on in the Month in which *date* lies.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `rangeOfUnit:startDate:interval:forDate:` (page 210)
- `ordinalityOfUnit:inUnit:forDate:` (page 209)

Related Sample Code
Birthdays

Declared In
`NSCalendar.h`

rangeOfUnit:startDate:interval:forDate:

Returns by reference the starting time and duration of a given calendar unit that contains a given date.

```
- (BOOL)rangeOfUnit:(NSCalendarUnit)unit startDate:(NSDate **)datep
    interval:(NSTimeInterval *)tip forDate:(NSDate *)date
```

Parameters

unit

A calendar unit (see "Calendar Units" (page 213) for possible values).

datep

Upon return, contains the starting time of the calendar unit *unit* that contains the date *date*

tip

Upon return, contains the duration of the calendar unit *unit* that contains the date *date*

date

A date.

Return Value

`YES` if the starting time and duration of a unit could be calculated, otherwise `NO`.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `rangeOfUnit:inUnit:forDate:` (page 210)
- `ordinalityOfUnit:inUnit:forDate:` (page 209)

Declared In

`NSCalendar.h`

setFirstWeekday:

Sets the index of the first weekday for the receiver.

```
- (void)setFirstWeekday:(NSUInteger)weekday
```

Parameters

weekday

The first weekday for the receiver.

Availability

Available in Mac OS X v10.4 and later.

See Also

- `firstWeekday` (page 207)

Declared In

`NSCalendar.h`

setLocale:

Sets the locale for the receiver.

```
- (void)setLocale:(NSLocale *)locale
```

Parameters

locale

The locale for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- locale (page 207)

Declared In
NSCalendar.h

setMinimumDaysInFirstWeek:

Sets the minimum number of days in the first week of the receiver.

```
- (void)setMinimumDaysInFirstWeek:(NSUInteger)mdw
```

Parameters

mdw

The minimum number of days in the first week of the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- minimumDaysInFirstWeek (page 208)

Declared In
NSCalendar.h

setTimeZone:

Sets the time zone for the receiver.

```
- (void)setTimeZone:(NSTimeZone *)tz
```

Parameters

tz

The time zone for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- timeZone (page 213)

Declared In
NSCalendar.h

timeZone

Returns the time zone for the receiver.

```
- (NSTimeZone *)timeZone
```

Return Value
The time zone for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setTimeZone:` (page 212)

Declared In
`NSCalendar.h`

Constants

NSCalendarUnit

Defines a type used to specify calendrical units such as day and month.

```
typedef NSUInteger NSCalendarUnit;
```

Discussion
See "Calendar Units" (page 213) for possible values.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSCalendar.h`

Calendar Units

Specify calendrical units such as day and month.

```
enum {
    NSEraCalendarUnit = kCFCalendarUnitEra,
    NSYearCalendarUnit = kCFCalendarUnitYear,
    NSMonthCalendarUnit = kCFCalendarUnitMonth,
    NSDayCalendarUnit = kCFCalendarUnitDay,
    NSHourCalendarUnit = kCFCalendarUnitHour,
    NSMinuteCalendarUnit = kCFCalendarUnitMinute,
    NSSecondCalendarUnit = kCFCalendarUnitSecond,
    NSWeekCalendarUnit = kCFCalendarUnitWeek,
    NSWeekdayCalendarUnit = kCFCalendarUnitWeekday,
    NSWeekdayOrdinalCalendarUnit = kCFCalendarUnitWeekdayOrdinal
};
```

Constants

`NSEraCalendarUnit`

Specifies the era unit.

The corresponding value is an `int`. Equal to `kCFCalendarUnitEra`.

Available in Mac OS X v10.4 and later.

Declared in `NSCalendar.h`.

`NSYearCalendarUnit`

Specifies the year unit.

The corresponding value is an `int`. Equal to `kCFCalendarUnitYear`.

Available in Mac OS X v10.4 and later.

Declared in `NSCalendar.h`.

`NSMonthCalendarUnit`

Specifies the month unit.

The corresponding value is an `int`. Equal to `kCFCalendarUnitMonth`.

Available in Mac OS X v10.4 and later.

Declared in `NSCalendar.h`.

`NSDayCalendarUnit`

Specifies the day unit.

The corresponding value is an `int`. Equal to `kCFCalendarUnitDay`.

Available in Mac OS X v10.4 and later.

Declared in `NSCalendar.h`.

`NSHourCalendarUnit`

Specifies the hour unit.

The corresponding value is an `int`. Equal to `kCFCalendarUnitHour`.

Available in Mac OS X v10.4 and later.

Declared in `NSCalendar.h`.

`NSMinuteCalendarUnit`

Specifies the minute unit.

The corresponding value is an `int`. Equal to `kCFCalendarUnitMinute`.

Available in Mac OS X v10.4 and later.

Declared in `NSCalendar.h`.

`NSSecondCalendarUnit`
Specifies the second unit.

The corresponding value is a `double`. Equal to `kCFCalendarUnitSecond`.

Available in Mac OS X v10.4 and later.

Declared in `NSCalendar.h`.

`NSWeekCalendarUnit`
Specifies the week unit.

The corresponding value is an `int`. Equal to `kCFCalendarUnitWeek`.

Available in Mac OS X v10.4 and later.

Declared in `NSCalendar.h`.

`NSWeekdayCalendarUnit`
Specifies the weekday unit.

The corresponding value is an `int`. Equal to `kCFCalendarUnitWeekday`. The weekday units are the numbers 1 through N (where for the Gregorian calendar N=7 and 1 is Sunday).

Available in Mac OS X v10.4 and later.

Declared in `NSCalendar.h`.

`NSWeekdayOrdinalCalendarUnit`
Specifies the ordinal weekday unit.

The corresponding value is an `int`. Equal to `kCFCalendarUnitWeekdayOrdinal`. The weekday ordinal unit describes ordinal position within the month unit of the corresponding weekday unit. For example, in the Gregorian calendar a weekday ordinal unit of 2 for a weekday unit 3 indicates "the second Tuesday in the month".

Available in Mac OS X v10.4 and later.

Declared in `NSCalendar.h`.

Discussion
Calendar units may be used as a bit mask to specify a combination of units. Values in this `enum` are equal to the corresponding constants in the `CFCalendarUnit enum`.

Declared In
`NSCalendar.h`

NSDateComponents wrapping behavior

The wrapping option specifies wrapping behavior for calculations involving `NSDateComponents` objects.

```
enum
{
    NSWrapCalendarComponents = kCFCalendarComponentsWrap,
};
```

Constants
`NSWrapCalendarComponents`
Specifies that the components specified for an `NSDateComponents` object should be incremented and wrap around to zero/one on overflow, but should not cause higher units to be incremented.

Available in Mac OS X v10.4 and later.

Declared in `NSCalendar.h`.

Declared In
`NSCalendar.h`

NSCalendarDate Class Reference

Inherits from	NSDate : NSObject
Conforms to	NSCoding (NSDate) NSCopying (NSDate) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSCalendarDate.h
Companion guides	Date and Time Programming Guide for Cocoa Data Formatting Programming Guide for Cocoa
Related sample code	Clock Control CoreRecipes GridCalendar NewsReader SimpleCalendar

Overview

`NSCalendarDate` is a public subclass of `NSDate` that represents concrete date objects and performs date computations based on the Gregorian calendar. These objects associate a time interval with a time zone and are especially suited for representing and manipulating dates according to western calendrical systems.

Important: Use of `NSCalendarDate` strongly discouraged. It is not deprecated yet, however it may be in the next major OS release after Mac OS X v10.5. For calendrical calculations, you should use suitable combinations of `NSCalendar`, `NSDate`, and `NSDateComponents`, as described in Calendars in *Date and Time Programming Guide for Cocoa*.

An `NSCalendarDate` object stores a date as the number of seconds relative to the absolute reference date (the first instance of 1 January 2001, GMT). Use the associated time zone to change how the `NSCalendarDate` object prints its time interval. The time zone does not change how the time interval is stored. Because the value is stored independently of the time zone, you can accurately compare `NSCalendarDate` objects with any other `NSDate` objects or use them to create other `NSDate` objects. It also means that you can track a date across different time zones; that is, you can create a new `NSCalendarDate` object with a different time zone to see how the particular date is represented in that time zone.

Important: `NSCalendarDate` uses the Gregorian calendar for all of time, even before it was actually adopted. `NSCalendar`'s version of the Gregorian calendar uses the Julian calendar before October 4, 1582. If you need to accurately deal with dates prior to October 4, 1582, you should use `NSCalendar`.

`NSCalendarDate` provides both class and instance methods for creating objects. Some of these methods allow you to initialize `NSCalendarDate` objects from strings, while others create objects from sets of integers corresponding to the standard time values (months, hours, seconds, and so on).

To retrieve conventional elements of an `NSCalendarDate` object, use the `...Of...` methods. For example, `dayOfWeek` (page 227) returns a number that indicates the day of the week (0 is Sunday). The `monthOfYear` (page 234) method returns a number from 1 through 12 that indicates the month.

The Calendar Format

Each `NSCalendarDate` object has a calendar format associated with it. This format is a string that contains date conversion specifiers that are very similar to those used in the standard C library function `strftime()`. `NSCalendarDate` interprets dates that are represented as strings conforming to this format. You can set the default format for an `NSCalendarDate` object at initialization time or using the `setCalendarFormat:` (page 235) method. Several methods allow you to specify formats other than the one bound to the object.

The date conversion specifiers cover a range of date conventions. See Converting Dates to Strings in *Date and Time Programming Guide for Cocoa* for the list of specifiers.

Locales and String Representations of Calendar Dates

`NSCalendarDate` provides several `description...` methods for representing dates as strings. These methods—`description` (page 228), `descriptionWithLocale:` (page 230), `descriptionWithCalendarFormat:` (page 228), and `descriptionWithCalendarFormat:locale:` (page 229)—take an implicit or explicit calendar format. The locale information affects the returned string. If you use `descriptionWithLocale:` or `descriptionWithCalendarFormat:locale:`, you may specify a locale dictionary. `NSCalendarDate` accesses the locale information as an `NSDictionary` object. The following keys in the locale dictionary affect `NSCalendarDate`:

`NSTimeDateFormatString`	A format string that specifies how dates with times are printed. The default is to use full month names and days with a 24-hour clock, as in "Sunday, January 01, 2001 23:00:00 Pacific Standard Time."
`NSAMPMDesignation`	An array of strings that specify how the morning and afternoon designations are printed. The defaults are AM and PM.
`NSMonthNameArray`	An array that specifies the full names for the months.
`NSShortMonthNameArray`	An array that specifies the abbreviations for the months.
`NSWeekDayNameArray`	An array that gives the names for the days of the week. Sunday should be the first day of the week.

`NSShortWeekDayNameArray`	An array that specifies the abbreviations for the days of the week. Sunday should be the first day of the week.

If a `description...` method does not have a locale parameter or if you pass `nil` as the locale to a method that takes a locale argument, `NSCalendarDate` uses the system default locale. The default locale—sometimes called the "root" locale—is a generic English-like locale. Typically you should instead use the user's preferences. You can obtain a dictionary representation of the user's standard user defaults using the `NSUserDefaults` method `dictionaryRepresentation` (page 1849), as illustrated in the following example:

```
NSCalendarDate *calendarDate = [[NSCalendarDate alloc]
initWithTimeIntervalSinceReferenceDate:uploadedTime];
[calendarDate descriptionWithLocale:[[NSUserDefaults standardUserDefaults]
dictionaryRepresentation]];
// ...
[calendarDate release];
```

Subclassing Notes

If you subclass `NSCalendarDate` and override `description` (page 228), you should also override `descriptionWithLocale:` (page 230). The `stringWithFormat:` (page 1536) method of `NSString` uses `descriptionWithLocale:` (page 230) instead of `description` when you use the `%@` conversion specifier to get a string representation of an `NSCalendarDate` object. That is, this message:

```
[NSString stringWithFormat:@"The current date and time are  %@",
        [MyNSCalendarDateSubclass date]]
```

invokes `descriptionWithLocale:` (page 230).

Tasks

Creating an NSCalendarDate Instance

`+ calendarDate` (page 221)
Creates and returns a calendar date initialized to the current date and time.

`+ dateWithString:calendarFormat:` (page 222)
Creates and returns a calendar date initialized with the date given as a string in a specified format.

`+ dateWithString:calendarFormat:locale:` (page 222)
Creates and returns a calendar date initialized with the date given as a string in a specified format and interpreted using a given locale.

`+ dateWithYear:month:day:hour:minute:second:timeZone:` (page 223)
Creates and returns a calendar date initialized with specified values for year, month, day, hour, minute, second, and time zone.

Initializing an NSCalendarDate Instance

- `initWithString:` (page 231)
  Returns a calendar date initialized with the date specified as a string in the default calendar format.
- `initWithString:calendarFormat:` (page 231)
  Returns a calendar date initialized with the date given as a string in a specified format.
- `initWithString:calendarFormat:locale:` (page 232)
  Returns a calendar date initialized with the date given as a string in a specified format and interpreted using a given locale.
- `initWithYear:month:day:hour:minute:second:timeZone:` (page 233)
  Returns a calendar date initialized with specified values for year, month, day, hour, minute, second, and time zone.

Retrieving Date Elements

- `dayOfCommonEra` (page 226)
  Returns the number of days between the receiver and the beginning of the Common Era.
- `dayOfMonth` (page 226)
  Returns the day of the month (1 through 31) of the receiver.
- `dayOfWeek` (page 227)
  Returns the day of the week (0 through 6) of the receiver.
- `dayOfYear` (page 227)
  Returns the day of the year (1 through 366) of the receiver.
- `hourOfDay` (page 231)
  Returns the hour (0 through 23) of the receiver.
- `minuteOfHour` (page 234)
  Returns the minute (0 through 59) of the receiver.
- `monthOfYear` (page 234)
  Returns the month of the year (1 through 12) of the receiver.
- `secondOfMinute` (page 235)
  Returns the second (0 through 59) of the receiver.
- `yearOfCommonEra` (page 237)
  Returns the year, including the century, of the receiver.

Adjusting a Date

- `dateByAddingYears:months:days:hours:minutes:seconds:` (page 225)
  Returns a new calendar date that represents the date of the receiver updated with given offsets.

Computing Date Intervals

- `years:months:days:hours:minutes:seconds:sinceDate:` (page 237)
  Computes the calendrical time difference between the receiver and a given date.

Representing Dates as Strings

- `description` (page 228)
  Returns a string representation of the receiver formatted as specified by the receiver's default calendar format.
- `descriptionWithCalendarFormat:` (page 228)
  Returns a string representation of the receiver.
- `descriptionWithCalendarFormat:locale:` (page 229)
  Returns a string representation of the receiver formatted according to given conversion specifiers and represented according to given locale information.
- `descriptionWithLocale:` (page 230)
  Returns a string representation of the receiver formatted as specified by the receiver's default calendar format and represented according to the given locale information.

Getting and Setting Calendar Formats

- `calendarFormat` (page 224)
  Returns the receiver's default calendar format.
- `setCalendarFormat:` (page 235)
  Sets the default calendar format for the receiver.

Managing the Time Zone

- `setTimeZone:` (page 236)
  Sets the time zone for the receiver.
- `timeZone` (page 236)
  Returns the time zone object associated with the receiver.

Class Methods

calendarDate

Creates and returns a calendar date initialized to the current date and time.

```
+ (id)calendarDate
```

Return Value
A new calendar date initialized to the current date and time.

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

See Also
`+ date` (page 393) (`NSDate`)

Related Sample Code
SimpleCalendar

Declared In
`NSCalendarDate.h`

dateWithString:calendarFormat:

Creates and returns a calendar date initialized with the date given as a string in a specified format.

```
+ (id)dateWithString:(NSString *)description calendarFormat:(NSString *)format
```

Parameters

description
: A string containing a description of a date in the format specified by *format*.

format
: A string used to interpret *description* and as the default calendar format for the new object. *format* consists of conversion specifiers similar to those used in `strftime()`. See Converting Dates to Strings, in *Date and Time Programming Guide for Cocoa* for more details.

Return Value
A new calendar date initialized with the date specified in *description*. Returns `nil` if *description* does not match *format* exactly.

Discussion
The following example shows how to get a calendar date with a temporal value corresponding to the form "Friday, 1 July 2001, 11:45 AM.":

```
NSCalendarDate *today = [NSCalendarDate
    dateWithString:@"Friday, 1 July 2001, 11:45 AM"
    calendarFormat:@"%A, %d %B %Y, %I:%M %p"];
```

If you include a time zone in the *description* parameter, this method verifies it and can substitute an alternative time zone. If the method does supply a new time zone, it applies the difference in offsets-from-GMT values between the substituted and the original time zones to the calendar date being created.

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

See Also
`+ dateWithString:calendarFormat:locale:` (page 222)
`- calendarFormat` (page 224)
`- initWithString:calendarFormat:` (page 231)

Declared In
`NSCalendarDate.h`

dateWithString:calendarFormat:locale:

Creates and returns a calendar date initialized with the date given as a string in a specified format and interpreted using a given locale.

```
+ (id)dateWithString:(NSString *)description calendarFormat:(NSString *)format
    locale:(id)localeDictionary
```

Parameters

description

A string containing a description of a date in the format specified by *format*.

format

A string used to interpret *description* and as the default calendar format for the new object. *format* consists of conversion specifiers similar to those used in `strftime()`. See Converting Dates to Strings, in *Date and Time Programming Guide for Cocoa* for more details.

localeDictionary

A dictionary that contains keys and values to represent the locale data to use when parsing *description*. See "Locales and String Representations of Calendar Dates" (page 218) for a list of the appropriate keys.

Return Value

A new calendar date initialized with the date specified by *description* and interpreted using the locale data in *localeDictionary*. Returns `nil` if *description* does not exactly match *format*.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ dateWithString:calendarFormat:` (page 222)
`- calendarFormat` (page 224)
`- initWithString:calendarFormat:locale:` (page 232)

Declared In

`NSCalendarDate.h`

dateWithYear:month:day:hour:minute:second:timeZone:

Creates and returns a calendar date initialized with specified values for year, month, day, hour, minute, second, and time zone.

```
+ (id)dateWithYear:(NSInteger)year month:(NSUInteger)month day:(NSUInteger)day
    hour:(NSUInteger)hour minute:(NSUInteger)minute second:(NSUInteger)second
    timeZone:(NSTimeZone *)aTimeZone
```

Parameters

year

The year for the new date. The value must include the century (for example, 1999 instead of 99).

month

The month for the new date. Valid values are `1` through `12`.

day

The day for the new date. Valid values are `1` through `31`.

hour

The hour for the new date. Valid values are `0` through `23`.

minute

The minute for the new date. Valid values are `0` through `59`.

second
The second for the new date. Valid values are `0` through `59`.

aTimeZone
The time zone for the new date.

Return Value
A new calendar date initialized with the specified values for year, month, day, hour, minute, second, and time zone.

Discussion
On days when daylight savings time "falls back," there are two 1:30 AMs. If you use this method, there is no way to create the *second* 1:30 AM. Instead, you should create the first and then use `dateByAddingYears:months:days:hours:minutes:seconds:` (page 225) to add an hour.

The following code fragment shows a calendar date created for 4 July 2001, 9 PM, Eastern Standard Time (`timeZoneWithName:` (page 1670) returns the `NSTimeZone` object that represents the time zone with the specified name):

```
NSCalendarDate *fireworks = [NSCalendarDate dateWithYear:2001
    month:7 day:4 hour:21 minute:0 second:0
    timeZone:[NSTimeZone timeZoneWithAbbreviation:@"EST"]];
```

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

See Also
`- initWithYear:month:day:hour:minute:second:timeZone:` (page 233)

Related Sample Code
SimpleCalendar

Declared In
`NSCalendarDate.h`

Instance Methods

calendarFormat

Returns the receiver's default calendar format.

```
- (NSString *)calendarFormat
```

Return Value
The receiver's default calendar format (used when the format is unspecified).

Discussion
You can set this format when you create the calendar date using one of the class methods `dateWithString:calendarFormat:` (page 222) or `dateWithString:calendarFormat:locale:` (page 222), or you can change the format using the instance method `setCalendarFormat:` (page 235). If you do not specify a default calendar format, `NSCalendarDate` substitutes its own default: an international format

of "%Y-%m-%d %H:%M:%S %z" (for example, 2001-03-24 16:45:12 +0900). See Converting Dates to Strings, in *Date and Time Programming Guide for Cocoa* for more information on what a calendar format contains.

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

See Also
- descriptionWithLocale: (page 230)

Declared In
NSCalendarDate.h

dateByAddingYears:months:days:hours:minutes:seconds:

Returns a new calendar date that represents the date of the receiver updated with given offsets.

```
- (NSCalendarDate *)dateByAddingYears:(NSInteger)year months:(NSInteger)month
    days:(NSInteger)day hours:(NSInteger)hour minutes:(NSInteger)minute
    seconds:(NSInteger)second
```

Parameters

year
The number of years to add to the receiver. The value may be negative to indicate a time in the past.

month
The number of months to add to the receiver. The value may be negative to indicate a time in the past.

day
The number of days to add to the receiver. The value may be negative to indicate a time in the past.

hour
The number of hours to add to the receiver. The value may be negative to indicate a time in the past.

minute
The number of minutes to add to the receiver. The value may be negative to indicate a time in the past.

second
The number of seconds to add to the receiver. The value may be negative to indicate a time in the past.

Return Value
A new calendar date that represents the date of the receiver updated with the year, month, day, hour, minute, and second offsets specified in the parameters.

Discussion
The parameter values are applied in a left-to-right order: *year* first, then *month*, then *day*, and so on. So, adding one month, four days to 27 April results in 31 May, not 1 June.

This method preserves "clock time" across changes in daylight saving time zones and leap years. If you add one day to 2:30 AM on the day before daylight saving time "springs ahead," it will actually result in 1:30 AM on the next day (which is one day, or 24 hours, later).

The following code fragment shows a calendar date created with a date a week later than an existing calendar date:

```
NSCalendarDate *now = [NSCalendarDate calendarDate];
NSCalendarDate *nextWeek = [now dateByAddingYears:0 months:0
    days:7 hours:0 minutes:0 seconds:0];
```

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

See Also
- `years:months:days:hours:minutes:seconds:sinceDate:` (page 237)

Related Sample Code
SimpleCalendar

Declared In
`NSCalendarDate.h`

dayOfCommonEra

Returns the number of days between the receiver and the beginning of the Common Era.

```
- (NSInteger)dayOfCommonEra
```

Return Value
The number of days between the receiver and the beginning of the Common Era.

Discussion
The base year of the Common Era is 1 C.E. (which is the same as 1 A.D.).

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

See Also
- `yearOfCommonEra` (page 237)

Related Sample Code
NewsReader

Declared In
`NSCalendarDate.h`

dayOfMonth

Returns the day of the month (1 through 31) of the receiver.

```
- (NSInteger)dayOfMonth
```

Return Value
The day of the month (1 through 31) of the receiver.

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

See Also
- `dayOfWeek` (page 227)

- dayOfYear (page 227)
- hourOfDay (page 231)
- minuteOfHour (page 234)
- monthOfYear (page 234)
- secondOfMinute (page 235)

Related Sample Code
Birthdays
SimpleCalendar

Declared In
NSCalendarDate.h

dayOfWeek

Returns the day of the week (0 through 6) of the receiver.

```
- (NSInteger)dayOfWeek
```

Return Value
The day of the week (0 through 6) of the receiver. 0 indicates Sunday.

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

See Also
- dayOfMonth (page 226)
- dayOfYear (page 227)
- hourOfDay (page 231)
- minuteOfHour (page 234)
- monthOfYear (page 234)
- secondOfMinute (page 235)

Related Sample Code
SimpleCalendar

Declared In
NSCalendarDate.h

dayOfYear

Returns the day of the year (1 through 366) of the receiver.

```
- (NSInteger)dayOfYear
```

Return Value
The day of the year (1 through 366) of the receiver.

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

See Also
- `dayOfMonth` (page 226)
- `dayOfWeek` (page 227)
- `hourOfDay` (page 231)
- `minuteOfHour` (page 234)
- `monthOfYear` (page 234)
- `secondOfMinute` (page 235)

Declared In
`NSCalendarDate.h`

description

Returns a string representation of the receiver formatted as specified by the receiver's default calendar format.

```
- (NSString *)description
```

Return Value
A string representation of the receiver, formatted as specified by the receiver's default calendar format.

Discussion
You can find out what the default calendar format is using the method `calendarFormat` (page 224). See "Locales and String Representations of Calendar Dates" (page 218) for information on locales and this method.

Because `NSCalendarDate` implements `descriptionWithLocale:` (page 230), `descriptionWithLocale:` is used to print the date when you use the `%@` conversion specifier. That is, the following statement invokes `descriptionWithLocale:`, not `description`:

```
NSLog(@"The current date and time is %@", [NSCalendarDate  date]);
```

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

See Also
- `descriptionWithCalendarFormat:` (page 228)
- `descriptionWithCalendarFormat:locale:` (page 229)
- `descriptionWithLocale:` (page 230)
- `setCalendarFormat:` (page 235)

Declared In
`NSCalendarDate.h`

descriptionWithCalendarFormat:

Returns a string representation of the receiver.

```
- (NSString *)descriptionWithCalendarFormat:(NSString *)format
```

Parameters

format

The format for the description. See Converting Dates to Strings, in *Date and Time Programming Guide for Cocoa* for a listing of specifiers.

Return Value

A string representation of the receiver, formatted as specified by the conversion specifiers in the calendar format string *format*.

Discussion

See "Locales and String Representations of Calendar Dates" (page 218) for information on locales and this method.

The following example shows how to create a description of the current date in the same format as "Tues 3/24/01 3:30 PM":

```
NSCalendarDate *now = [NSCalendarDate calendarDate];
NSString *nowAsString =
    [now descriptionWithCalendarFormat:@"%a %m/%d/%y %I:%M %p"];
```

Availability

Available in Mac OS X v10.0 and later.

See Also

- `- description` (page 228)
- `- descriptionWithCalendarFormat:locale:` (page 229)
- `- descriptionWithLocale:` (page 230)

Related Sample Code

Clock Control

SimpleCalendar

Declared In

`NSCalendarDate.h`

descriptionWithCalendarFormat:locale:

Returns a string representation of the receiver formatted according to given conversion specifiers and represented according to given locale information.

```
- (NSString *)descriptionWithCalendarFormat:(NSString *)format
    locale:(id)localeDictionary
```

Parameters

format

The format for the description. See Converting Dates to Strings, in *Date and Time Programming Guide for Cocoa* for a list of specifiers.

localeDictionary

A dictionary that contains keys and values to represent the locale data to use when creating the description. See "Locales and String Representations of Calendar Dates" (page 218) for further details.

Return Value

A string representation of the receiver, formatted according to the conversion specifiers in *format* and represented according to the locale information in *localeDictionary*.

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

See Also
- `- description` (page 228)
- `- descriptionWithCalendarFormat:` (page 228)
- `- descriptionWithLocale:` (page 230)

Related Sample Code
NewsReader

Declared In
`NSCalendarDate.h`

descriptionWithLocale:

Returns a string representation of the receiver formatted as specified by the receiver's default calendar format and represented according to the given locale information.

```
- (NSString *)descriptionWithLocale:(id)localeDictionary
```

Parameters

localeDictionary

A dictionary that contains keys and values to represent the locale data to use when creating the description. See "Locales and String Representations of Calendar Dates" (page 218) for further details.

Return Value
A string representation of the receiver formatted as specified by the receiver's default calendar format and represented according to the locale information in *localeDictionary*.

Discussion
You can find out what the default calendar format is using the method `calendarFormat` (page 224).

This method is used to print an `NSCalendarDate` object when the `%@` conversion specifier is used. That is, this statement invokes `descriptionWithLocale::`

```
NSLog(@"The current date and time is %@", [NSCalendarDate  date]);
```

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

See Also
- `- description` (page 228)
- `- descriptionWithCalendarFormat:` (page 228)
- `- descriptionWithCalendarFormat:locale:` (page 229)
- `- setCalendarFormat:` (page 235)

Declared In
`NSCalendarDate.h`

hourOfDay

Returns the hour (0 through 23) of the receiver.

```
- (NSInteger)hourOfDay
```

Return Value
The hour (0 through 23) of the receiver.

Discussion
On daylight saving time "fall back" days, a value of `1` is returned for two consecutive hours, but with a different time zone (the first in daylight saving time and the second in standard time).

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

See Also
- `dayOfMonth` (page 226)
- `dayOfWeek` (page 227)
- `dayOfYear` (page 227)
- `minuteOfHour` (page 234)
- `monthOfYear` (page 234)
- `secondOfMinute` (page 235)

Declared In
`NSCalendarDate.h`

initWithString:

Returns a calendar date initialized with the date specified as a string in the default calendar format.

```
- (id)initWithString:(NSString *)description
```

Parameters
`description`
The description of the new date. The string must conform to the default calendar format "`%Y-%m-%d %H:%M:%S %z`" (for example, `2001-03-24 16:45:12 +0900`). See Converting Dates to Strings, in *Date and Time Programming Guide for Cocoa* for a discussion of date conversion specifiers.

Return Value
A calendar date initialized with the date specified by *`description`*. Returns `nil` if *`description`* does not exactly match the default calendar format.

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

Declared In
`NSCalendarDate.h`

initWithString:calendarFormat:

Returns a calendar date initialized with the date given as a string in a specified format.

```
- (id)initWithString:(NSString *)description calendarFormat:(NSString *)format
```

Parameters

description

A string containing a description of a date in the format specified by *format*.

format

A string used to interpret *description* and as the default calendar format for the new object. *format* consists of conversion specifiers similar to those used in `strftime()`. See Converting Dates to Strings, in *Date and Time Programming Guide for Cocoa* for more details.

Discussion

The following example shows how to initialize a calendar date with a string of the form "03.24.01 22:00 PST":

```
NSCalendarDate *newDate = [[NSCalendarDate alloc]
    initWithString:@"03.24.01 22:00 PST"
    calendarFormat:@"%m.%d.%y %H:%M %Z"];
```

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ dateWithString:calendarFormat:` (page 222)

`- calendarFormat` (page 224)

Related Sample Code

Clock Control

Declared In

`NSCalendarDate.h`

initWithString:calendarFormat:locale:

Returns a calendar date initialized with the date given as a string in a specified format and interpreted using a given locale.

```
- (id)initWithString:(NSString *)description calendarFormat:(NSString *)format
    locale:(id)localeDictionary
```

Parameters

description

A string containing a description of a date in the format specified by *format*.

format

A string used to interpret *description* and as the default calendar format for the new object. *format* consists of conversion specifiers similar to those used in `strftime()`. See Converting Dates to Strings, in *Date and Time Programming Guide for Cocoa* for more details.

localeDictionary

A dictionary that contains keys and values to represent the locale data to use when parsing *description*. See "Locales and String Representations of Calendar Dates" (page 218) for a list of the appropriate keys.

Return Value
A calendar date initialized with the date specified in the string *description*. Returns `nil` if you specify a locale dictionary that has a month name array with more than 12 elements or a day name array with more than 7 arguments.

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

See Also
`+ dateWithString:calendarFormat:locale:` (page 222)
`- calendarFormat` (page 224)

Declared In
`NSCalendarDate.h`

initWithYear:month:day:hour:minute:second:timeZone:

Returns a calendar date initialized with specified values for year, month, day, hour, minute, second, and time zone.

```
- (id)initWithYear:(NSInteger)year month:(NSUInteger)month day:(NSUInteger)day
    hour:(NSUInteger)hour minute:(NSUInteger)minute second:(NSUInteger)second
    timeZone:(NSTimeZone *)aTimeZone
```

Parameters

year
The year for the new date. The value must include the century (for example, 1999 instead of 99).

month
The month for the new date. Valid values are `1` through `12`.

day
The day for the new date. Valid values are `1` through `31`.

hour
The hour for the new date. Valid values are `0` through `23`.

minute
The minute for the new date. Valid values are `0` through `59`.

second
The second for the new date. Valid values are `0` through `59`.

aTimeZone
The time zone for the new date.

Return Value
A calendar date initialized with the specified values for year, month, day, hour, minute, second, and time zone.

Discussion
On days when daylight saving time "falls back," there are two 1:30 AMs. If you use this method there is no way to create the *second* 1:30 AM. Instead, you should create the first and then use `dateByAddingYears:months:days:hours:minutes:seconds:` (page 225) to add an hour.

The following code fragment shows a calendar date created with a date of 4 July 2001, 9 PM, Eastern Standard Time (`timeZoneWithName:` (page 1670) returns the `NSTimeZone` object that represents the time zone with the specified name):

```
NSCalendarDate *fireworks = [[[NSCalendarDate alloc] initWithYear:2001
    month:7 day:4 hour:21 minute:0 second:0
    timeZone:[NSTimeZone timeZoneWithAbbreviation:@"EST"]]  autorelease];
```

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

See Also
`+ dateWithYear:month:day:hour:minute:second:timeZone:` (page 223)

Related Sample Code
GridCalendar

Declared In
`NSCalendarDate.h`

minuteOfHour

Returns the minute (0 through 59) of the receiver.

```
- (NSInteger)minuteOfHour
```

Return Value
The minute (0 through 59) of the receiver.

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

See Also
- `dayOfMonth` (page 226)
- `dayOfWeek` (page 227)
- `dayOfYear` (page 227)
- `hourOfDay` (page 231)
- `monthOfYear` (page 234)
- `secondOfMinute` (page 235)

Declared In
`NSCalendarDate.h`

monthOfYear

Returns the month of the year (1 through 12) of the receiver.

```
- (NSInteger)monthOfYear
```

Return Value
The month of the year (1 through 12) of the receiver.

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

See Also
- `dayOfMonth` (page 226)
- `dayOfWeek` (page 227)
- `dayOfYear` (page 227)
- `hourOfDay` (page 231)
- `minuteOfHour` (page 234)
- `secondOfMinute` (page 235)

Related Sample Code
Birthdays
GridCalendar
SimpleCalendar

Declared In
`NSCalendarDate.h`

secondOfMinute

Returns the second (0 through 59) of the receiver.

```
- (NSInteger)secondOfMinute
```

Return Value
The seconds value (0 through 59) of the receiver.

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

See Also
- `dayOfMonth` (page 226)
- `dayOfWeek` (page 227)
- `dayOfYear` (page 227)
- `hourOfDay` (page 231)
- `minuteOfHour` (page 234)
- `monthOfYear` (page 234)

Declared In
`NSCalendarDate.h`

setCalendarFormat:

Sets the default calendar format for the receiver.

```
- (void)setCalendarFormat:(NSString *)format
```

Parameters

format

The default calendar format for the receiver. See Converting Dates to Strings, in *Date and Time Programming Guide for Cocoa* for a list of the date conversion specifiers.

Discussion

A calendar format is a string formatted with date conversion specifiers. If you do not specify a calendar format for an object, `NSCalendarDate` substitutes its own default. The default is the international format of "`%Y-%m-%d %H:%M:%S %z`" (for example, `2001-03-24 16:45:12 +0900`).

Availability

Available in Mac OS X v10.0 and later.

See Also

- `calendarFormat` (page 224)
- `description` (page 228)
- `descriptionWithLocale:` (page 230)

Declared In

`NSCalendarDate.h`

setTimeZone:

Sets the time zone for the receiver.

```
- (void)setTimeZone:(NSTimeZone *)aTimeZone
```

Parameters

aTimeZone

The time zone for the receiver.

Discussion

If you do not specify a time zone for an object at initialization time, `NSCalendarDate` uses the default time zone for the locale.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `timeZone` (page 236)

Declared In

`NSCalendarDate.h`

timeZone

Returns the time zone object associated with the receiver.

```
- (NSTimeZone *)timeZone
```

Return Value

The time zone object associated with the receiver.

Discussion
You can set the time zone when you create the calendar date using the class methods `dateWithString:calendarFormat:` (page 222) or `dateWithString:calendarFormat:locale:` (page 222) by including the time zone in the description and format parameters. Or you can explicitly set the time zone to an `NSTimeZone` object using `dateWithYear:month:day:hour:minute:second:timeZone:` (page 223). If you do not specify a time zone for an object at initialization time, `NSCalendarDate` uses the default time zone for the locale.

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

See Also
`- setTimeZone:` (page 236)

Related Sample Code
SimpleCalendar

Declared In
`NSCalendarDate.h`

yearOfCommonEra

Returns the year, including the century, of the receiver.

```
- (NSInteger)yearOfCommonEra
```

Return Value
The year, including the century, of the receiver (for example, 1995). The base year of the Common Era is 1 C.E. (which is the same as 1 A.D.).

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

See Also
`- dayOfCommonEra` (page 226)

Related Sample Code
GridCalendar
Reminders
SimpleCalendar

Declared In
`NSCalendarDate.h`

years:months:days:hours:minutes:seconds:sinceDate:

Computes the calendrical time difference between the receiver and a given date.

```
- (void)years:(NSInteger *)yearsPointer months:(NSInteger *)monthsPointer
    days:(NSInteger *)daysPointer hours:(NSInteger *)hoursPointer minutes:(NSInteger
     *)minutesPointer seconds:(NSInteger *)secondsPointer sinceDate:(NSCalendarDate
     *)date
```

Parameters

yearsPointer

Upon return, contains the number of years between the receiver and *date*. Pass NULL to ignore this component.

monthsPointer

Upon return, contains the number of months between the receiver and *date*. Pass NULL to ignore this component.

daysPointer

Upon return, contains the number of days between the receiver and *date*. Pass NULL to ignore this component.

hoursPointer

Upon return, contains the number of hours between the receiver and *date*. Pass NULL to ignore this component.

minutesPointer

Upon return, contains the number of minutes between the receiver and *date*. Pass NULL to ignore this component.

secondsPointer

Upon return, contains the number of seconds between the receiver and *date*. Pass NULL to ignore this component.

date

The date with which to compare the receiver. The value must not be nil, otherwise an exception is raised.

Discussion

You can choose any representation you wish for the time difference by passing NULL for arguments you want to ignore, other than *date*. The following example illustrates how to compute the difference in months, days, and years between two dates.

```
NSCalendarDate *momsBDay = [NSCalendarDate dateWithYear:1936
    month:1 day:8 hour:7 minute:30 second:0
    timeZone:[NSTimeZone timeZoneWithAbbreviation:@"EST"]];
NSCalendarDate *dateOfBirth = [NSCalendarDate dateWithYear:1965
    month:12 day:7 hour:17 minute:25 second:0
    timeZone:[NSTimeZone timeZoneWithAbbreviation:@"EST"]];
int years, months, days;

[dateOfBirth years:&years months:&months days:&days  hours:NULL
    minutes:NULL seconds:NULL sinceDate:momsBDay];
```

This returns 29 years, 10 months, and 29 days. To express the years in terms of months, pass NULL for the years argument:

```
[dateOfBirth years:NULL months:&months days:&days hours:NULL
    minutes:NULL seconds:NULL sinceDate:momsBDay];
```

This returns 358 months and 29 days.

Availability

Available in Mac OS X v10.0 and later.

See Also

- dateByAddingYears:months:days:hours:minutes:seconds: (page 225)

Related Sample Code
SimpleCalendar

Declared In
`NSCalendarDate.h`

NSCharacterSet Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSMutableCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSCharacterSet.h
Companion guide	String Programming Guide for Cocoa
Related sample code	ImageClient iSpend Quartz Composer WWDC 2005 TextEdit TextEditPlus VertexPerformanceTest

Overview

An `NSCharacterSet` object represents a set of Unicode-compliant characters. `NSString` and `NSScanner` objects use `NSCharacterSet` objects to group characters together for searching operations, so that they can find any of a particular set of characters during a search. The cluster's two public classes, `NSCharacterSet` and `NSMutableCharacterSet`, declare the programmatic interface for static and dynamic character sets, respectively.

The objects you create using these classes are referred to as character set objects (and when no confusion will result, merely as character sets). Because of the nature of class clusters, character set objects aren't actual instances of the `NSCharacterSet` or `NSMutableCharacterSet` classes but of one of their private subclasses. Although a character set object's class is private, its interface is public, as declared by these abstract superclasses, `NSCharacterSet` and `NSMutableCharacterSet`. The character set classes adopt the `NSCopying` and `NSMutableCopying` protocols, making it convenient to convert a character set of one type to the other.

The `NSCharacterSet` class declares the programmatic interface for an object that manages a set of Unicode characters (see the `NSString` class cluster specification for information on Unicode). `NSCharacterSet`'s principal primitive method, `characterIsMember:` (page 253), provides the basis for all other instance methods in its interface. A subclass of `NSCharacterSet` needs only to implement this method, plus

`mutableCopyWithZone:` (page 2094), for proper behavior. For optimal performance, a subclass should also override `bitmapRepresentation` (page 253), which otherwise works by invoking `characterIsMember:` (page 253) for every possible Unicode value.

`NSCharacterSet` is "toll-free bridged" with its Cocoa Foundation counterpart, *CFCharacterSet Reference*. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSCharacterSet *` parameter, you can pass a `CFCharacterSetRef`, and in a function where you see a `CFCharacterSetRef` parameter, you can pass an `NSCharacterSet` instance (you cast one type to the other to suppress compiler warnings). See Interchangeable Data Types for more information on toll-free bridging.

The mutable subclass of `NSCharacterSet` is `NSMutableCharacterSet`.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 2034)
- `initWithCoder:` (page 2034)

NSCopying
- `copyWithZone:` (page 2042)

NSMutableCopying
- `mutableCopyWithZone:` (page 2094)

Tasks

Creating a Standard Character Set

`+ alphanumericCharacterSet` (page 244)
Returns a character set containing the characters in the categories Letters, Marks, and Numbers.

`+ capitalizedLetterCharacterSet` (page 244)
Returns a character set containing the characters in the category of Titlecase Letters.

`+ controlCharacterSet` (page 247)
Returns a character set containing the characters in the categories of Control or Format Characters.

`+ decimalDigitCharacterSet` (page 247)
Returns a character set containing the characters in the category of Decimal Numbers.

`+ decomposableCharacterSet` (page 248)
Returns a character set containing all individual Unicode characters that can also be represented as composed character sequences.

`+ illegalCharacterSet` (page 248)
Returns a character set containing values in the category of Non-Characters or that have not yet been defined in version 3.2 of the Unicode standard.

+ `letterCharacterSet` (page 249)
  Returns a character set containing the characters in the categories Letters and Marks.

+ `lowercaseLetterCharacterSet` (page 249)
  Returns a character set containing the characters in the category of Lowercase Letters.

+ `newlineCharacterSet` (page 250)
  Returns a character set containing the newline characters.

+ `nonBaseCharacterSet` (page 250)
  Returns a character set containing the characters in the category of Marks.

+ `punctuationCharacterSet` (page 250)
  Returns a character set containing the characters in the category of Punctuation.

+ `symbolCharacterSet` (page 251)
  Returns a character set containing the characters in the category of Symbols.

+ `uppercaseLetterCharacterSet` (page 251)
  Returns a character set containing the characters in the categories of Uppercase Letters and Titlecase Letters.

+ `whitespaceAndNewlineCharacterSet` (page 252)
  Returns a character set containing only the whitespace characters space (`U+0020`) and tab (`U+0009`) and the newline and nextline characters (`U+000A`–`U+000D`, `U+0085`).

+ `whitespaceCharacterSet` (page 252)
  Returns a character set containing only the in-line whitespace characters space (`U+0020`) and tab (`U+0009`).

Creating a Custom Character Set

+ `characterSetWithCharactersInString:` (page 245)
  Returns a character set containing the characters in a given string.

+ `characterSetWithRange:` (page 246)
  Returns a character set containing characters with Unicode values in a given range.

- `invertedSet` (page 254)
  Returns a character set containing only characters that don't exist in the receiver.

Creating and Managing Character Sets as Bitmap Representations

+ `characterSetWithBitmapRepresentation:` (page 245)
  Returns a character set containing characters determined by a given bitmap representation.

+ `characterSetWithContentsOfFile:` (page 246)
  Returns a character set read from the bitmap representation stored in the file a given path.

- `bitmapRepresentation` (page 253)
  Returns an `NSData` object encoding the receiver in binary format.

Testing Set Membership

- `characterIsMember:` (page 253)
  Returns a Boolean value that indicates whether a given character is in the receiver.
- `hasMemberInPlane:` (page 254)
  Returns a Boolean value that indicates whether the receiver has at least one member in a given character plane.
- `isSupersetOfSet:` (page 254)
  Returns a Boolean value that indicates whether the receiver is a superset of another given character set.
- `longCharacterIsMember:` (page 255)
  Returns a Boolean value that indicates whether a given long character is a member of the receiver.

Class Methods

alphanumericCharacterSet

Returns a character set containing the characters in the categories Letters, Marks, and Numbers.

```
+ (id)alphanumericCharacterSet
```

Return Value
A character set containing the characters in the categories Letters, Marks, and Numbers.

Discussion
Informally, this set is the set of all characters used as basic units of alphabets, syllabaries, ideographs, and digits.

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

See Also
`+ letterCharacterSet` (page 249)
`+ decimalDigitCharacterSet` (page 247)

Declared In
`NSCharacterSet.h`

capitalizedLetterCharacterSet

Returns a character set containing the characters in the category of Titlecase Letters.

```
+ (id)capitalizedLetterCharacterSet
```

Return Value
A character set containing the characters in the category of Titlecase Letters.

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

See Also
+ letterCharacterSet (page 249)
+ uppercaseLetterCharacterSet (page 251)

Declared In
NSCharacterSet.h

characterSetWithBitmapRepresentation:

Returns a character set containing characters determined by a given bitmap representation.

```
+ (id)characterSetWithBitmapRepresentation:(NSData *)data
```

Parameters
data
A bitmap representation of a character set.

Return Value
A character set containing characters determined by *data*.

Discussion
This method is useful for creating a character set object with data from a file or other external data source.

A raw bitmap representation of a character set is a byte array of 2^16 bits (that is, 8192 bytes). The value of the bit at position *n* represents the presence in the character set of the character with decimal Unicode value *n*. To add a character with decimal Unicode value *n* to a raw bitmap representation, use a statement such as the following:

```
unsigned char bitmapRep[8192];
bitmapRep[n >> 3] |= (((unsigned int)1) << (n & 7));
```

To remove that character:

```
bitmapRep[n >> 3] &= ~(((unsigned int)1) << (n & 7));
```

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

See Also
- bitmapRepresentation (page 253)
+ characterSetWithContentsOfFile: (page 246)

Declared In
NSCharacterSet.h

characterSetWithCharactersInString:

Returns a character set containing the characters in a given string.

```
+ (id)characterSetWithCharactersInString:(NSString *)aString
```

Parameters

aString

A string containing characters for the new character set.

Return Value

A character set containing the characters in *aString*. Returns an empty character set if *aString* is empty.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

iSpend

QTAudioExtractionPanel

Quartz Composer QCTV

Declared In

NSCharacterSet.h

characterSetWithContentsOfFile:

Returns a character set read from the bitmap representation stored in the file a given path.

```
+ (id)characterSetWithContentsOfFile:(NSString *)path
```

Parameters

path

A path to a file containing a bitmap representation of a character set. The path name must end with the extension `.bitmap`.

Return Value

A character set read from the bitmap representation stored in the file at *path*.

Discussion

To read a bitmap representation from any file, use the `NSData` method`dataWithContentsOfFile:options:error:` (page 373) and pass the result to `characterSetWithBitmapRepresentation:` (page 245).

This method doesn't use filenames to check for the uniqueness of the character sets it creates. To prevent duplication of character sets in memory, cache them and make them available through an API that checks whether the requested set has already been loaded.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSCharacterSet.h

characterSetWithRange:

Returns a character set containing characters with Unicode values in a given range.

```
+ (id)characterSetWithRange:(NSRange)aRange
```

Parameters

aRange

A range of Unicode values.

`aRange.location` is the value of the first character to return; `aRange.location + aRange.length- 1` is the value of the last.

Return Value
A character set containing characters whose Unicode values are given by *aRange*. If `aRange.length` is `0`, returns an empty character set.

Discussion
This code excerpt creates a character set object containing the lowercase English alphabetic characters:

```
NSRange lcEnglishRange;
NSCharacterSet *lcEnglishLetters;

lcEnglishRange.location = (unsigned int)'a';
lcEnglishRange.length = 26;
lcEnglishLetters = [NSCharacterSet characterSetWithRange:lcEnglishRange];
```

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

Declared In
`NSCharacterSet.h`

controlCharacterSet

Returns a character set containing the characters in the categories of Control or Format Characters.

```
+ (id)controlCharacterSet
```

Return Value
A character set containing the characters in the categories of Control or Format Characters.

Discussion
These characters are specifically the Unicode values `U+0000` to `U+001F` and `U+007F` to `U+009F`.

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

See Also
`+ illegalCharacterSet` (page 248)

Related Sample Code
Link Snoop

Declared In
`NSCharacterSet.h`

decimalDigitCharacterSet

Returns a character set containing the characters in the category of Decimal Numbers.

```
+ (id)decimalDigitCharacterSet
```

Return Value
A character set containing the characters in the category of Decimal Numbers.

Discussion
Informally, this set is the set of all characters used to represent the decimal values `0` through `9`. These characters include, for example, the decimal digits of the Indic scripts and Arabic.

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

See Also
`+ alphanumericCharacterSet` (page 244)

Declared In
`NSCharacterSet.h`

decomposableCharacterSet

Returns a character set containing all individual Unicode characters that can also be represented as composed character sequences.

```
+ (id)decomposableCharacterSet
```

Return Value
A character set containing all individual Unicode characters that can also be represented as composed character sequences (such as for letters with accents), by the definition of "standard decomposition" in version 3.2 of the Unicode character encoding standard.

Discussion
These characters include compatibility characters as well as pre-composed characters.

> **Note:** This character set doesn't currently include the Hangul characters defined in version 2.0 of the Unicode standard.

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

See Also
`+ nonBaseCharacterSet` (page 250)

Declared In
`NSCharacterSet.h`

illegalCharacterSet

Returns a character set containing values in the category of Non-Characters or that have not yet been defined in version 3.2 of the Unicode standard.

```
+ (id)illegalCharacterSet
```

Return Value
A character set containing values in the category of Non-Characters or that have not yet been defined in version 3.2 of the Unicode standard.

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

See Also
`+ controlCharacterSet` (page 247)

Declared In
`NSCharacterSet.h`

letterCharacterSet

Returns a character set containing the characters in the categories Letters and Marks.

```
+ (id)letterCharacterSet
```

Return Value
A character set containing the characters in the categories Letters and Marks.

Discussion
Informally, this set is the set of all characters used as letters of alphabets and ideographs.

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

See Also
`+ alphanumericCharacterSet` (page 244)
`+ lowercaseLetterCharacterSet` (page 249)
`+ uppercaseLetterCharacterSet` (page 251)

Declared In
`NSCharacterSet.h`

lowercaseLetterCharacterSet

Returns a character set containing the characters in the category of Lowercase Letters.

```
+ (id)lowercaseLetterCharacterSet
```

Return Value
A character set containing the characters in the category of Lowercase Letters.

Discussion
Informally, this set is the set of all characters used as lowercase letters in alphabets that make case distinctions.

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

See Also
`+ uppercaseLetterCharacterSet` (page 251)

+ letterCharacterSet (page 249)

Declared In
NSCharacterSet.h

newlineCharacterSet

Returns a character set containing the newline characters.

```
+ (id)newlineCharacterSet
```

Return Value
A character set containing the newline characters (U+000A–U+000D, U+0085).

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

See Also
+ whitespaceAndNewlineCharacterSet (page 252)
+ whitespaceCharacterSet (page 252)

Declared In
NSCharacterSet.h

nonBaseCharacterSet

Returns a character set containing the characters in the category of Marks.

```
+ (id)nonBaseCharacterSet
```

Return Value
A character set containing the characters in the category of Marks.

Discussion
This set is also defined as all legal Unicode characters with a non-spacing priority greater than 0. Informally, this set is the set of all characters used as modifiers of base characters.

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

See Also
+ decomposableCharacterSet (page 248)

Declared In
NSCharacterSet.h

punctuationCharacterSet

Returns a character set containing the characters in the category of Punctuation.

```
+ (id)punctuationCharacterSet
```

Return Value
A character set containing the characters in the category of Punctuation.

Discussion
Informally, this set is the set of all non-whitespace characters used to separate linguistic units in scripts, such as periods, dashes, parentheses, and so on.

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

Declared In
`NSCharacterSet.h`

symbolCharacterSet

Returns a character set containing the characters in the category of Symbols.

```
+ (id)symbolCharacterSet
```

Return Value
A character set containing the characters in the category of Symbols.

Discussion
These characters include, for example, the dollar sign ($) and the plus (+) sign.

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

Declared In
`NSCharacterSet.h`

uppercaseLetterCharacterSet

Returns a character set containing the characters in the categories of Uppercase Letters and Titlecase Letters.

```
+ (id)uppercaseLetterCharacterSet
```

Return Value
A character set containing the characters in the categories of Uppercase Letters and Titlecase Letters.

Discussion
Informally, this set is the set of all characters used as uppercase letters in alphabets that make case distinctions.

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

See Also
`+ capitalizedLetterCharacterSet` (page 244)
`+ lowercaseLetterCharacterSet` (page 249)
`+ letterCharacterSet` (page 249)

Declared In
`NSCharacterSet.h`

whitespaceAndNewlineCharacterSet

Returns a character set containing only the whitespace characters space (U+0020) and tab (U+0009) and the newline and nextline characters (U+000A–U+000D, U+0085).

```
+ (id)whitespaceAndNewlineCharacterSet
```

Return Value
A character set containing only the whitespace characters space (U+0020) and tab (U+0009) and the newline and nextline characters (U+000A–U+000D, U+0085).

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

See Also
+ newlineCharacterSet (page 250)
+ whitespaceCharacterSet (page 252)

Related Sample Code
ImageMapExample
Quartz Composer WWDC 2005 TextEdit
TextEditPlus
TextLinks
VertexPerformanceTest

Declared In
NSCharacterSet.h

whitespaceCharacterSet

Returns a character set containing only the in-line whitespace characters space (U+0020) and tab (U+0009).

```
+ (id)whitespaceCharacterSet
```

Return Value
A character set containing only the in-line whitespace characters space (U+0020) and tab (U+0009).

Discussion
This set doesn't contain the newline or carriage return characters.

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

See Also
+ whitespaceAndNewlineCharacterSet (page 252)
+ newlineCharacterSet (page 250)

Related Sample Code
CoreRecipes
ImageClient

Declared In
NSCharacterSet.h

Instance Methods

bitmapRepresentation

Returns an `NSData` object encoding the receiver in binary format.

```
- (NSData *)bitmapRepresentation
```

Return Value
An `NSData` object encoding the receiver in binary format.

Discussion
This format is suitable for saving to a file or otherwise transmitting or archiving.

A raw bitmap representation of a character set is a byte array of 2^16 bits (that is, 8192 bytes). The value of the bit at position *n* represents the presence in the character set of the character with decimal Unicode value *n*. To test for the presence of a character with decimal Unicode value *n* in a raw bitmap representation, use an expression such as the following:

```
unsigned char bitmapRep[8192];
if (bitmapRep[n >> 3] & (((unsigned int)1) << (n  & 7))) {
    /* Character is present. */
}
```

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

See Also
`+ characterSetWithBitmapRepresentation:` (page 245)

Declared In
`NSCharacterSet.h`

characterIsMember:

Returns a Boolean value that indicates whether a given character is in the receiver.

```
- (BOOL)characterIsMember:(unichar)aCharacter
```

Parameters
aCharacter
The character to test for membership of the receiver.

Return Value
`YES` if *aCharacter* is in the receiving character set, otherwise `NO`.

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

See Also
`- longCharacterIsMember:` (page 255)

Declared In
NSCharacterSet.h

hasMemberInPlane:

Returns a Boolean value that indicates whether the receiver has at least one member in a given character plane.

```
- (BOOL)hasMemberInPlane:(uint8_t)thePlane
```

Parameters

thePlane
A character plane.

Return Value
YES if the receiver has at least one member in *thePlane*, otherwise NO.

Discussion
This method makes it easier to find the plane containing the members of the current character set. The Basic Multilingual Plane is plane 0.

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

Declared In
NSCharacterSet.h

invertedSet

Returns a character set containing only characters that don't exist in the receiver.

```
- (NSCharacterSet *)invertedSet
```

Return Value
A character set containing only characters that don't exist in the receiver.

Discussion
Inverting an immutable character set is much more efficient than inverting a mutable character set.

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

See Also
invert (page 942) (NSMutableCharacterSet)

Declared In
NSCharacterSet.h

isSupersetOfSet:

Returns a Boolean value that indicates whether the receiver is a superset of another given character set.

```
- (BOOL)isSupersetOfSet:(NSCharacterSet *)theOtherSet
```

Parameters

theOtherSet

A character set.

Return Value

YES if the receiver is a superset of *theOtherSet*, otherwise NO.

Availability

Available in Mac OS X v10.2 and later.

Declared In

NSCharacterSet.h

longCharacterIsMember:

Returns a Boolean value that indicates whether a given long character is a member of the receiver.

```
- (BOOL)longCharacterIsMember:(UTF32Char)theLongChar
```

Parameters

theLongChar

A UTF32 character.

Return Value

YES if *theLongChar* is in the receiver, otherwise NO.

Discussion

This method supports the specification of 32-bit characters.

Availability

Available in Mac OS X v10.2 and later.

See Also

- characterIsMember: (page 253)

Declared In

NSCharacterSet.h

Constants

NSOpenStepUnicodeReservedBase

Specifies lower bound for a Unicode character range reserved for Apple's corporate use.

```
enum {
   NSOpenStepUnicodeReservedBase = 0xF400
};
```

Constants

`NSOpenStepUnicodeReservedBase`

Specifies lower bound for a Unicode character range reserved for Apple's corporate use (the range is `0xF400-0xF8FF`).

Available in Mac OS X v10.0 and later.

Declared in `NSCharacterSet.h`.

Declared In

`NSCharacterSet.h`

NSClassDescription Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSClassDescription.h
Companion guides	Cocoa Scripting Guide Key-Value Coding Programming Guide

Overview

`NSClassDescription` is an abstract class that provides the interface for querying the relationships and properties of a class. Concrete subclasses of `NSClassDescription` provide the available attributes of objects of a particular class and the relationships between that class and other classes. Defining these relationships between classes allows for more intelligent and flexible manipulation of objects with key-value coding.

It is important to note that there are no class descriptions by default. To use `NSClassDescription` objects in your code you have to implement them for your model classes. For all concrete subclasses, you must provide implementations for all instance methods of `NSClassDescription`. (`NSClassDescription` provides only the implementation for the class methods that maintain the cache of registered class descriptions.) Once created, you must register a class description with the `NSClassDescription` method `registerClassDescription:forClass:` (page 259).

You can use the `NSString` objects in the arrays returned by methods such as attributeKeys (page 260) and toManyRelationshipKeys (page 261) to access—using key-value coding—the properties of an instance of the class to which a class description object corresponds. For more about attributes and relationships, see *Cocoa Fundamentals Guide*. For more about key-value coding, see *Key-Value Coding Programming Guide*.

`NSScriptClassDescription`, which is used to map the relationships between scriptable classes, is the only concrete subclass of `NSClassDescription` provided as part of the Cocoa framework.

Tasks

Working with Class Descriptions

`+ classDescriptionForClass:` (page 258)
Returns the class description for a given class.

`+ invalidateClassDescriptionCache` (page 259)
Removes all `NSClassDescription` objects from the cache.

`+ registerClassDescription:forClass:` (page 259)
Registers an `NSClassDescription` object for a given class in the `NSClassDescription` cache.

Attribute Keys

`- attributeKeys` (page 260)
Overridden by subclasses to return the names of attributes of instances of the described class.

Relationship Keys

`- inverseForRelationshipKey:` (page 260)
Overridden by subclasses to return the name of the inverse relationship from a relationship specified by a given key.

`- toManyRelationshipKeys` (page 261)
Overridden by subclasses to return the keys for the to-many relationship properties of instances of the described class.

`- toOneRelationshipKeys` (page 261)
Overridden by subclasses to return the keys for the to-one relationship properties of instances of the described class.

Class Methods

classDescriptionForClass:

Returns the class description for a given class.

```
+ (NSClassDescription *)classDescriptionForClass:(Class)aClass
```

Parameters

aClass
The class for which to return a class description.

Return Value
The class description for *aClass*, or `nil` if a class description cannot be found.

Discussion
If a class description for *aClass* is not found, the method posts an `NSClassDescriptionNeededForClassNotification` on behalf of *aClass*, allowing an observer to register a class description. The method then checks for a class description again. Returns `nil` if a class description is still not found.

If you have an instance of the receiver's class, you can use the `NSObject` instance method `classDescription` (page 1170) instead.

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

Declared In
`NSClassDescription.h`

invalidateClassDescriptionCache

Removes all `NSClassDescription` objects from the cache.

```
+ (void)invalidateClassDescriptionCache
```

Discussion
You should rarely need to invoke this method. Use it whenever a registered `NSClassDescription` object might be replaced by a different version, such as when you have loaded a new provider of `NSClassDescription` objects, or when you are about to remove a provider of `NSClassDescription` objects.

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

Declared In
`NSClassDescription.h`

registerClassDescription:forClass:

Registers an `NSClassDescription` object for a given class in the `NSClassDescription` cache.

```
+ (void)registerClassDescription:(NSClassDescription *)description
    forClass:(Class)aClass
```

Parameters

description
The class description to register.

aClass
The class for which to register *description*.

Discussion
You should rarely need to directly invoke this method.

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

Declared In
NSClassDescription.h

Instance Methods

attributeKeys

Overridden by subclasses to return the names of attributes of instances of the described class.

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

Return Value
An array of `NSString` objects containing the names of attributes of instances of the described class.

Discussion
For example, a class description that describes Movie objects could return the attribute keys `title`, `dateReleased`, and `rating`.

If you have an instance of the class the receiver describes, you can use the `NSObject` instance method `attributeKeys` (page 1168) instead.

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

See Also
- `toManyRelationshipKeys` (page 261)
- `toOneRelationshipKeys` (page 261)

Declared In
NSClassDescription.h

inverseForRelationshipKey:

Overridden by subclasses to return the name of the inverse relationship from a relationship specified by a given key.

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

Return Value
The name of the inverse relationship from the relationship specified by *relationshipKey*.

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

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

returns the string `employees`.

If you have an instance of the class the receiver describes, you can use the `NSObject` instance method `inverseForRelationshipKey:` (page 1180) instead.

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

Declared In
`NSClassDescription.h`

toManyRelationshipKeys

Overridden by subclasses to return the keys for the to-many relationship properties of instances of the described class.

```
- (NSArray *)toManyRelationshipKeys
```

Return Value
An array of `NSString` objects containing the names of the to-many relationship properties of instances of the described class.

Discussion
To-many relationship properties are arrays of objects.

If you have an instance of the class the receiver describes, you can use the `NSObject` instance method `toManyRelationshipKeys` (page 1194) instead.

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

See Also
- `attributeKeys` (page 260)
- `toOneRelationshipKeys` (page 261)

Declared In
`NSClassDescription.h`

toOneRelationshipKeys

Overridden by subclasses to return the keys for the to-one relationship properties of instances of the described class.

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

Return Value
An array of `NSString` objects containing the names of the to-one relationship properties of instances of the described class.

Discussion
To-one relationship properties are single objects.

If you have an instance of the class the receiver describes, you can use the `NSObject` instance method `toOneRelationshipKeys` (page 1195) instead.

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

See Also
- `attributeKeys` (page 260)
- `toManyRelationshipKeys` (page 261)

Declared In
`NSClassDescription.h`

Notifications

NSClassDescriptionNeededForClassNotification

Posted by `classDescriptionForClass:` (page 258) when a class description cannot be found for a class.

After the notification is processed, `classDescriptionForClass:` (page 258) checks for a class description again. This checking allows an observer to register class descriptions lazily. The notification is posted only once for any given class, even if the class description remains undefined.

The notification object is the class object for which the class description is requested. This notification does not contain a *`userInfo`* dictionary.

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

Declared In
`NSClassDescription.h`

NSCloneCommand Class Reference

Inherits from	NSScriptCommand : NSObject
Conforms to	NSCoding (NSScriptCommand) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptStandardSuiteCommands.h
Companion guide	Cocoa Scripting Guide

Overview

An instance of `NSCloneCommand` clones the specified scriptable object or objects (such as words, paragraphs, images, and so on) and inserts them in the specified location, or the default location if no location is specified. The cloned scriptable objects typically correspond to objects in the application, but aren't required to. This command corresponds to AppleScript's `duplicate` command.

`NSCloneCommand` is part of Cocoa's built-in scripting support. It works automatically to support the `duplicate` command through key-value coding. Most applications don't need to subclass `NSCloneCommand` or invoke its methods.

When an instance of `NSCloneCommand` is executed, it clones the specified objects by sending them `copyWithZone:` (page 1157) messages.

Tasks

Working with Specifiers

- `keySpecifier` (page 264)
  Returns a specifier for the object or objects to be cloned.
- `setReceiversSpecifier:` (page 264)
  Sets the receiver's object specifier;.

Instance Methods

keySpecifier

Returns a specifier for the object or objects to be cloned.

```
- (NSScriptObjectSpecifier *)keySpecifier
```

Return Value
A specifier for the object or objects to be cloned.

Discussion
For example, the specifier may indicate that a document's third rectangle should be cloned. The returned specifier is valid only in the context of the `NSCloneCommand` object; for example, if you send the specifier a `containerSpecifier` (page 1416) message, the result is `nil`.

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

Declared In
`NSScriptStandardSuiteCommands.h`

setReceiversSpecifier:

Sets the receiver's object specifier;.

```
- (void)setReceiversSpecifier:(NSScriptObjectSpecifier *)receiversRef
```

Parameters
`receiversRef`
The object specifier for the receiver.

Discussion
When evaluated, the specifier indicates the receiver or receivers of the `clone` command.

This method overrides `setReceiversSpecifier:` (page 1390) in `NSScriptCommand`. It performs the same function as the overridden method, with a critical difference: it causes the container specifier part of the passed-in object specifier to become the receiver specifier of the command, and the key part of the passed-in object specifier to become the key specifier. If, for example, *`receiversRef`* is a specifier for `the third rectangle of the first document`, the receiver specifier is `the first document` while the key specifier is `the third rectangle`.

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

Declared In
`NSScriptStandardSuiteCommands.h`

NSCloseCommand Class Reference

Inherits from	NSScriptCommand : NSObject
Conforms to	NSCoding (NSScriptCommand) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptStandardSuiteCommands.h
Companion guide	Cocoa Scripting Guide
Related sample code	Quartz Composer WWDC 2005 TextEdit TextEditPlus

Overview

An instance of `NSCloseCommand` closes the specified scriptable object or objects—typically a document or window (and its associated document, if any). The command may optionally specify a location to save in and how to handle modified documents (by automatically saving changes, not saving them, or asking the user).

`NSCloseCommand` is part of Cocoa's built-in scripting support. It works automatically to support the `close` command through key-value coding. Most applications don't need to subclass `NSCloseCommand` or call its methods.

Tasks

Accessing Save Options

`- saveOptions` (page 266)
Returns a constant indicating how to deal with closing any modified documents.

Instance Methods

saveOptions

Returns a constant indicating how to deal with closing any modified documents.

```
- (NSSaveOptions)saveOptions
```

Return Value
A constant indicating how to deal with closing any modified documents. The default value returned is `NSSaveOptionsAsk`. See "Constants" (page 266) for a list of possible return values.

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

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSScriptStandardSuiteCommands.h`

Constants

NSSaveOptions

The `saveOptions` (page 266) method returns one of the following constants to indicate how to deal with saving any modified documents:

```
typedef enum {
    NSSaveOptionsYes = 0,
    NSSaveOptionsNo,
    NSSaveOptionsAsk
} NSSaveOptions;
```

Constants

`NSSaveOptionsYes`
Indicates a modified document should be saved on closing without asking the user.
Available in Mac OS X v10.0 and later.
Declared in `NSScriptStandardSuiteCommands.h`.

`NSSaveOptionsNo`
Indicates a modified document should not be saved on closing.
Available in Mac OS X v10.0 and later.
Declared in `NSScriptStandardSuiteCommands.h`.

`NSSaveOptionsAsk`

Indicates the user should be asked before saving any modified documents on closing. When no option is specified, this is the default.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptStandardSuiteCommands.h`.

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

Declared In
`NSScriptStandardSuiteCommands.h`

NSCoder Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSCoder.h Foundation/NSKeyedArchiver.h Foundation/NSGeometry.h
Companion guide	Archives and Serializations Programming Guide for Cocoa
Related sample code	bMoviePaletteCocoa iSpend Mountains Reducer StickiesExample

Overview

The `NSCoder` abstract class declares the interface used by concrete subclasses to transfer objects and other Objective-C data items between memory and some other format. This capability provides the basis for archiving (where objects and data items are stored on disk) and distribution (where objects and data items are copied between different processes or threads). The concrete subclasses provided by Foundation for these purposes are `NSArchiver`, `NSUnarchiver`, `NSKeyedArchiver`, `NSKeyedUnarchiver`, and `NSPortCoder`. Concrete subclasses of `NSCoder` are referred to in general as coder classes, and instances of these classes as coder objects (or simply coders). A coder object that can only encode values is referred to as an encoder object, and one that can only decode values as a decoder object.

`NSCoder` operates on objects, scalars, C arrays, structures, and strings, and on pointers to these types. It does not handle types whose implementation varies across platforms, such as `union`, `void *`, function pointers, and long chains of pointers. A coder object stores object type information along with the data, so an object decoded from a stream of bytes is normally of the same class as the object that was originally encoded into the stream. An object can change its class when encoded, however; this is described in *Archives and Serializations Programming Guide for Cocoa*.

Tasks

Testing Coder

- `allowsKeyedCoding` (page 273)
  Returns a Boolean value that indicates whether the receiver supports keyed coding of objects.
- `containsValueForKey:` (page 274)
  Returns a Boolean value that indicates whether an encoded value is available for a string.

Encoding Data

- `encodeArrayOfObjCType:count:at:` (page 283)
  Encodes an array of *count* items, whose Objective-C type is given by *itemType*.
- `encodeBool:forKey:` (page 283)
  Encodes *boolv* and associates it with the string *key*.
- `encodeBycopyObject:` (page 284)
  Can be overridden by subclasses to encode *object* so that a copy, rather than a proxy, is created upon decoding.
- `encodeByrefObject:` (page 284)
  Can be overridden by subclasses to encode *object* so that a proxy, rather than a copy, is created upon decoding.
- `encodeBytes:length:` (page 284)
  Encodes a buffer of data whose types are unspecified.
- `encodeBytes:length:forKey:` (page 285)
  Encodes a buffer of data, *bytesp*, whose length is specified by *lenv*, and associates it with the string *key*.
- `encodeConditionalObject:` (page 285)
  Can be overridden by subclasses to conditionally encode *object*, preserving common references to that object.
- `encodeConditionalObject:forKey:` (page 286)
  Conditionally encodes a reference to *objv* and associates it with the string *key* only if *objv* has been unconditionally encoded with `encodeObject:forKey:` (page 290).
- `encodeDataObject:` (page 286)
  Encodes a given `NSData` object.
- `encodeDouble:forKey:` (page 287)
  Encodes *realv* and associates it with the string *key*.
- `encodeFloat:forKey:` (page 287)
  Encodes *realv* and associates it with the string *key*.
- `encodeInt:forKey:` (page 288)
  Encodes *intv* and associates it with the string *key*.
- `encodeInteger:forKey:` (page 289)
  Encodes a given `NSInteger` and associates it with a given key.

- `encodeInt32:forKey:` (page 287)
  Encodes the 32-bit integer *intv* and associates it with the string *key*.
- `encodeInt64:forKey:` (page 288)
  Encodes the 64-bit integer *intv* and associates it with the string *key*.
- `encodeObject:` (page 289)
  Encodes *object*.
- `encodeObject:forKey:` (page 290)
  Encodes the object *objv* and associates it with the string *key*.
- `encodePoint:` (page 291)
  Encodes *point*.
- `encodePoint:forKey:` (page 291)
  Encodes *point* and associates it with the string *key*.
- `encodePropertyList:` (page 291)
  Encodes the property list *aPropertyList*.
- `encodeRect:` (page 291)
  Encodes *rect*.
- `encodeRect:forKey:` (page 292)
  Encodes *rect* and associates it with the string *key*.
- `encodeRootObject:` (page 292)
  Can be overridden by subclasses to encode an interconnected group of Objective-C objects, starting with *rootObject*.
- `encodeSize:` (page 293)
  Encodes *size*.
- `encodeSize:forKey:` (page 293)
  Encodes *size* and associates it with the string *key*.
- `encodeValueOfObjCType:at:` (page 293)
  Must be overridden by subclasses to encode a single value residing at *address*, whose Objective-C type is given by *valueType*.
- `encodeValuesOfObjCTypes:` (page 294)
  Encodes a series of values of potentially differing Objective-C types.
- `encodeNXObject:` (page 289) Deprecated in Mac OS X v10.5
  Encodes an old-style object onto the coder.

Decoding Data

- `decodeArrayOfObjCType:count:at:` (page 274)
  Decodes an array of *count* items, whose Objective-C type is given by *itemType*.
- `decodeBoolForKey:` (page 274)
  Decodes and returns a boolean value that was previously encoded with `encodeBool:forKey:` (page 283) and associated with the string *key*.
- `decodeBytesForKey:returnedLength:` (page 275)
  Decodes a buffer of data that was previously encoded with `encodeBytes:length:forKey:` (page 285) and associated with the string *key*.

- `decodeBytesWithReturnedLength:` (page 275)
  Decodes a buffer of data whose types are unspecified.
- `decodeDataObject` (page 276)
  Decodes and returns an NSData object that was previously encoded with `encodeDataObject:` (page 286). Subclasses must override this method.
- `decodeDoubleForKey:` (page 276)
  Decodes and returns a `double` value that was previously encoded with either `encodeFloat:forKey:` (page 287) or `encodeDouble:forKey:` (page 287) and associated with the string *key*.
- `decodeFloatForKey:` (page 276)
  Decodes and returns a `float` value that was previously encoded with `encodeFloat:forKey:` (page 287) or `encodeDouble:forKey:` (page 287) and associated with the string *key*.
- `decodeIntForKey:` (page 278)
  Decodes and returns an `int` value that was previously encoded with `encodeInt:forKey:` (page 288), `encodeInteger:forKey:` (page 289), `encodeInt32:forKey:` (page 287), or `encodeInt64:forKey:` (page 288) and associated with the string *key*.
- `decodeIntegerForKey:` (page 278)
  Decodes and returns an `NSInteger` value that was previously encoded with `encodeInt:forKey:` (page 288), `encodeInteger:forKey:` (page 289), `encodeInt32:forKey:` (page 287), or `encodeInt64:forKey:` (page 288) and associated with the string *key*.
- `decodeInt32ForKey:` (page 277)
  Decodes and returns a 32-bit integer value that was previously encoded with `encodeInt:forKey:` (page 288), `encodeInteger:forKey:` (page 289), `encodeInt32:forKey:` (page 287), or `encodeInt64:forKey:` (page 288) and associated with the string *key*.
- `decodeInt64ForKey:` (page 277)
  Decodes and returns a 64-bit integer value that was previously encoded with `encodeInt:forKey:` (page 288), `encodeInteger:forKey:` (page 289), `encodeInt32:forKey:` (page 287), or `encodeInt64:forKey:` (page 288) and associated with the string *key*.
- `decodeObject` (page 279)
  Decodes an Objective-C object that was previously encoded with any of the `encode...Object:` methods.
- `decodeObjectForKey:` (page 279)
  Decodes and returns an autoreleased Objective-C object that was previously encoded with `encodeObject:forKey:` (page 290) or `encodeConditionalObject:forKey:` (page 286) and associated with the string *key*.
- `decodePoint` (page 280)
  Decodes and returns an `NSPoint` structure that was previously encoded with `encodePoint:` (page 291).
- `decodePointForKey:` (page 280)
  Decodes and returns an `NSPoint` structure that was previously encoded with `encodePoint:forKey:` (page 291).
- `decodePropertyList` (page 280)
  Decodes a property list that was previously encoded with `encodePropertyList:` (page 291).
- `decodeRect` (page 280)
  Decodes and returns an `NSRect` structure that was previously encoded with `encodeRect:` (page 291).

- `decodeRectForKey:` (page 281)
  Decodes and returns an `NSRect` structure that was previously encoded with `encodeRect:forKey:` (page 292).
- `decodeSize` (page 281)
  Decodes and returns an `NSSize` structure that was previously encoded with `encodeSize:` (page 293).
- `decodeSizeForKey:` (page 281)
  Decodes and returns an `NSSize` structure that was previously encoded with `encodeSize:forKey:` (page 293).
- `decodeValueOfObjCType:at:` (page 282)
  Decodes a single value, whose Objective-C type is given by *valueType*.
- `decodeValuesOfObjCTypes:` (page 282)
  Decodes a series of potentially different Objective-C types.
- `decodeNXObject` (page 278) Deprecated in Mac OS X v10.5
  Decodes an object previously written with `encodeNXObject:` (page 289).

Managing Zones

- `objectZone` (page 294)
  Returns the memory zone used to allocate decoded objects.
- `setObjectZone:` (page 295)
  `NSCoder`'s implementation of this method does nothing.

Getting Version Information

- `systemVersion` (page 295)
  During encoding, this method should return the system version currently in effect.
- `versionForClassName:` (page 295)
  Returns the version in effect for the class with a given name.

Instance Methods

allowsKeyedCoding

Returns a Boolean value that indicates whether the receiver supports keyed coding of objects.

```
- (BOOL)allowsKeyedCoding
```

Discussion
The default implementation returns `NO`. Concrete subclasses that support keyed coding, such as `NSKeyedArchiver`, need to override this method to return `YES`.

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

Declared In
NSCoder.h

containsValueForKey:

Returns a Boolean value that indicates whether an encoded value is available for a string.

```
- (BOOL)containsValueForKey:(NSString *)key
```

Discussion
The string is passed as *key*. Subclasses must override this method if they perform keyed coding.

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

Declared In
NSCoder.h

decodeArrayOfObjCType:count:at:

Decodes an array of *count* items, whose Objective-C type is given by *itemType*.

```
- (void)decodeArrayOfObjCType:(const char *)itemType count:(NSUInteger)count at:(void
    *)address
```

Discussion
The items are decoded into the buffer beginning at *address*, which must be large enough to contain them all. *itemType* must contain exactly one type code. NSCoder's implementation invokes decodeValueOfObjCType:at: (page 282) to decode the entire array of items. If you use this method to decode an array of Objective-C objects, you are responsible for releasing each object.

This method matches an encodeArrayOfObjCType:count:at: (page 283) message used during encoding.

For information on creating an Objective-C type code suitable for *itemType*, see the "Type Encodings" section in the "The Objective-C Runtime System" chapter of *The Objective-C 2.0 Programming Language*.

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

See Also
- decodeValuesOfObjCTypes: (page 282)

Declared In
NSCoder.h

decodeBoolForKey:

Decodes and returns a boolean value that was previously encoded with encodeBool:forKey: (page 283) and associated with the string *key*.

```
- (BOOL)decodeBoolForKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

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

Related Sample Code
iSpend
Reducer

Declared In
NSCoder.h

decodeBytesForKey:returnedLength:

Decodes a buffer of data that was previously encoded with `encodeBytes:length:forKey:` (page 285) and associated with the string *key*.

```
- (const uint8_t *)decodeBytesForKey:(NSString *)key returnedLength:(NSUInteger
    *)lengthp
```

Discussion
The buffer's length is returned by reference in *lengthp*. The returned bytes are immutable. Subclasses must override this method if they perform keyed coding.

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

See Also
- `encodeBytes:length:forKey:` (page 285)

Declared In
NSCoder.h

decodeBytesWithReturnedLength:

Decodes a buffer of data whose types are unspecified.

```
- (void *)decodeBytesWithReturnedLength:(NSUInteger *)numBytes
```

Discussion
NSCoder's implementation invokes `decodeValueOfObjCType:at:` (page 282) to decode the data as a series of bytes, which this method then places into a buffer and returns. The buffer's length is returned by reference in *numBytes*. If you need the bytes beyond the scope of the current autorelease pool, you must copy them.

This method matches an `encodeBytes:length:` (page 284) message used during encoding.

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

See Also
- `encodeArrayOfObjCType:count:at:` (page 283)

Declared In
NSCoder.h

decodeDataObject

Decodes and returns an NSData object that was previously encoded with `encodeDataObject:` (page 286). Subclasses must override this method.

```
- (NSData *)decodeDataObject
```

Discussion
The implementation of your overriding method must match the implementation of your `encodeDataObject:` (page 286) method. For example, a typical `encodeDataObject:` (page 286) method encodes the number of bytes of data followed by the bytes themselves. Your override of this method must read the number of bytes, create an NSData object of the appropriate size, and decode the bytes into the new NSData object. Your overriding method should return an autoreleased NSData object.

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

Declared In
NSCoder.h

decodeDoubleForKey:

Decodes and returns a `double` value that was previously encoded with either `encodeFloat:forKey:` (page 287) or `encodeDouble:forKey:` (page 287) and associated with the string *key*.

```
- (double)decodeDoubleForKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

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

Related Sample Code
QTQuartzPlayer
Squiggles

Declared In
NSCoder.h

decodeFloatForKey:

Decodes and returns a `float` value that was previously encoded with `encodeFloat:forKey:` (page 287) or `encodeDouble:forKey:` (page 287) and associated with the string *key*.

```
- (float)decodeFloatForKey:(NSString *)key
```

Discussion
If the value was encoded as a `double`, the extra precision is lost. Also, if the encoded real value does not fit into a `float`, the method raises an `NSRangeException`. Subclasses must override this method if they perform keyed coding.

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

Related Sample Code
iSpend

Declared In
`NSCoder.h`

decodeInt32ForKey:

Decodes and returns a 32-bit integer value that was previously encoded with `encodeInt:forKey:` (page 288), `encodeInteger:forKey:` (page 289), `encodeInt32:forKey:` (page 287), or `encodeInt64:forKey:` (page 288) and associated with the string *key*.

```
- (int32_t)decodeInt32ForKey:(NSString *)key
```

Discussion
If the encoded integer does not fit into a 32-bit integer, the method raises an `NSRangeException`. Subclasses must override this method if they perform keyed coding.

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

Declared In
`NSCoder.h`

decodeInt64ForKey:

Decodes and returns a 64-bit integer value that was previously encoded with `encodeInt:forKey:` (page 288), `encodeInteger:forKey:` (page 289), `encodeInt32:forKey:` (page 287), or `encodeInt64:forKey:` (page 288) and associated with the string *key*.

```
- (int64_t)decodeInt64ForKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

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

Declared In
`NSCoder.h`

decodeIntegerForKey:

Decodes and returns an `NSInteger` value that was previously encoded with `encodeInt:forKey:` (page 288), `encodeInteger:forKey:` (page 289), `encodeInt32:forKey:` (page 287), or `encodeInt64:forKey:` (page 288) and associated with the string *key*.

```
- (NSInteger)decodeIntegerForKey:(NSString *)key
```

Discussion
If the encoded integer does not fit into the `NSInteger` size, the method raises an `NSRangeException`. Subclasses must override this method if they perform keyed coding.

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

Declared In
`NSCoder.h`

decodeIntForKey:

Decodes and returns an `int` value that was previously encoded with `encodeInt:forKey:` (page 288), `encodeInteger:forKey:` (page 289), `encodeInt32:forKey:` (page 287), or `encodeInt64:forKey:` (page 288) and associated with the string *key*.

```
- (int)decodeIntForKey:(NSString *)key
```

Discussion
If the encoded integer does not fit into the default integer size, the method raises an `NSRangeException`. Subclasses must override this method if they perform keyed coding.

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

Related Sample Code
Reducer

Declared In
`NSCoder.h`

decodeNXObject

Decodes an object previously written with `encodeNXObject:` (page 289). (Deprecated in Mac OS X v10.5.)

```
- (id)decodeNXObject
```

Discussion
No sharing is done across separate `decodeNXObject` invocations. Callers must have implemented an `initWithCoder:` (page 2034), which parallels the `read:` methods, on all of their classes that may be touched by this operation. The returned object is autoreleased.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
NSCoder.h

decodeObject

Decodes an Objective-C object that was previously encoded with any of the `encode...Object:` methods.

```
- (id)decodeObject
```

Discussion
`NSCoder`'s implementation invokes `decodeValueOfObjCType:at:` (page 282) to decode the object data.

Subclasses may need to override this method if they override any of the corresponding `encode...Object:` methods. For example, if an object was encoded conditionally using the `encodeConditionalObject:` (page 285) method, this method needs to check whether the object had actually been encoded.

The implementation for the concrete subclass `NSUnarchiver` returns an object that is retained by the unarchiver and is released when the unarchiver is deallocated. Therefore, you must retain the returned object before releasing the unarchiver. `NSKeyedUnarchiver`'s implementation, however, returns an autoreleased object, so its life is the same as the current autorelease pool instead of the keyed unarchiver.

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

See Also
- `encodeBycopyObject:` (page 284)
- `encodeByrefObject:` (page 284)
- `encodeObject:` (page 289)

Related Sample Code
bMoviePalette
bMoviePaletteCocoa
Clock Control
StickiesExample

Declared In
NSCoder.h

decodeObjectForKey:

Decodes and returns an autoreleased Objective-C object that was previously encoded with `encodeObject:forKey:` (page 290) or `encodeConditionalObject:forKey:` (page 286) and associated with the string *key*.

```
- (id)decodeObjectForKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

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

Related Sample Code
IBFragmentView
iSpend
Mountains
Reducer
StickiesExample

Declared In
NSCoder.h

decodePoint

Decodes and returns an NSPoint structure that was previously encoded with encodePoint: (page 291).

```
- (NSPoint)decodePoint
```

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

Declared In
NSGeometry.h

decodePointForKey:

Decodes and returns an NSPoint structure that was previously encoded with encodePoint:forKey: (page 291).

```
- (NSPoint)decodePointForKey:(NSString *)key
```

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

Declared In
NSKeyedArchiver.h

decodePropertyList

Decodes a property list that was previously encoded with encodePropertyList: (page 291).

```
- (id)decodePropertyList
```

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

Declared In
NSCoder.h

decodeRect

Decodes and returns an NSRect structure that was previously encoded with encodeRect: (page 291).

```
- (NSRect)decodeRect
```

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

Declared In
NSGeometry.h

decodeRectForKey:

Decodes and returns an `NSRect` structure that was previously encoded with `encodeRect:forKey:` (page 292).

```
- (NSRect)decodeRectForKey:(NSString *)key
```

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

Declared In
NSKeyedArchiver.h

decodeSize

Decodes and returns an `NSSize` structure that was previously encoded with `encodeSize:` (page 293).

```
- (NSSize)decodeSize
```

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

Declared In
NSGeometry.h

decodeSizeForKey:

Decodes and returns an `NSSize` structure that was previously encoded with `encodeSize:forKey:` (page 293).

```
- (NSSize)decodeSizeForKey:(NSString *)key
```

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

Related Sample Code
Reducer

Declared In
NSKeyedArchiver.h

decodeValueOfObjCType:at:

Decodes a single value, whose Objective-C type is given by *valueType*.

```
- (void)decodeValueOfObjCType:(const char *)valueType at:(void *)data
```

Discussion
valueType must contain exactly one type code, and the buffer specified by *data* must be large enough to hold the value corresponding to that type code. For information on creating an Objective-C type code suitable for *valueType*, see the "Type Encodings" section in "The Objective-C Runtime System" chapter of *The Objective-C 2.0 Programming Language*.

Subclasses must override this method and provide an implementation to decode the value. In your overriding implementation, decode the value into the buffer beginning at *data*. If your overriding method is capable of decoding an Objective-C object, your method must also retain that object. Clients of this method are then responsible for releasing the object.

This method matches an `encodeValueOfObjCType:at:` (page 293) message used during encoding.

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

See Also
- `decodeArrayOfObjCType:count:at:` (page 274)
- `decodeValuesOfObjCTypes:` (page 282)
- `decodeObject` (page 279)

Declared In
`NSCoder.h`

decodeValuesOfObjCTypes:

Decodes a series of potentially different Objective-C types.

```
- (void)decodeValuesOfObjCTypes:(const char *)valueTypes, ...
```

Discussion
valueTypes is a single string containing any number of type codes. The variable arguments to this method consist of one or more pointer arguments, each of which specifies the buffer in which to place a single decoded value. For each type code in *valueTypes*, you must specify a corresponding pointer argument whose buffer is large enough to hold the decoded value. If you use this method to decode Objective-C objects, you are responsible for releasing them.

This method matches an `encodeValuesOfObjCTypes:` (page 294) message used during encoding.

`NSCoder`'s implementation invokes `decodeValueOfObjCType:at:` (page 282) to decode individual types. Subclasses that implement the `decodeValueOfObjCType:at:` (page 282) method do not need to override this method.

For information on creating Objective-C type codes suitable for *valueTypes*, see the "Type Encodings" section in "The Objective-C Runtime System" chapter of *The Objective-C 2.0 Programming Language*.

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

See Also
- `decodeArrayOfObjCType:count:at:` (page 274)

Declared In
`NSCoder.h`

encodeArrayOfObjCType:count:at:

Encodes an array of *count* items, whose Objective-C type is given by *itemType*.

```
- (void)encodeArrayOfObjCType:(const char *)itemType count:(NSUInteger)count
    at:(const void *)address
```

Discussion
The values are encoded from the buffer beginning at *address*. *itemType* must contain exactly one type code. `NSCoder`'s implementation invokes `encodeValueOfObjCType:at:` (page 293) to encode the entire array of items. Subclasses that implement the `encodeValueOfObjCType:at:` (page 293) method do not need to override this method.

This method must be matched by a subsequent `decodeArrayOfObjCType:count:at:` (page 274) message.

For information on creating an Objective-C type code suitable for *itemType*, see the "Type Encodings" section in "The Objective-C Runtime System" chapter of *The Objective-C Programming Language*.

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

See Also
- `encodeValueOfObjCType:at:` (page 293)
- `encodeValuesOfObjCTypes:` (page 294)
- `encodeBytes:length:` (page 284)

Declared In
`NSCoder.h`

encodeBool:forKey:

Encodes *boolv* and associates it with the string *key*.

```
- (void)encodeBool:(BOOL)boolv forKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

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

See Also
- `decodeBoolForKey:` (page 274)

Related Sample Code
iSpend
Reducer

Declared In
NSCoder.h

encodeBycopyObject:

Can be overridden by subclasses to encode *object* so that a copy, rather than a proxy, is created upon decoding.

```
- (void)encodeBycopyObject:(id)object
```

Discussion
NSCoder's implementation simply invokes encodeObject: (page 289).

This method must be matched by a corresponding decodeObject (page 279) message.

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

See Also
- encodeRootObject: (page 292)
- encodeConditionalObject: (page 285)
- encodeByrefObject: (page 284)

Declared In
NSCoder.h

encodeByrefObject:

Can be overridden by subclasses to encode *object* so that a proxy, rather than a copy, is created upon decoding.

```
- (void)encodeByrefObject:(id)object
```

Discussion
NSCoder's implementation simply invokes encodeObject: (page 289).

This method must be matched by a corresponding decodeObject (page 279) message.

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

See Also
- encodeBycopyObject: (page 284)

Declared In
NSCoder.h

encodeBytes:length:

Encodes a buffer of data whose types are unspecified.

```
- (void)encodeBytes:(const void *)address length:(NSUInteger)numBytes
```

Discussion
The buffer to be encoded begins at *address*, and its length in bytes is given by *numBytes*.

This method must be matched by a corresponding `decodeBytesWithReturnedLength:` (page 275) message.

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

See Also
`- encodeArrayOfObjCType:count:at:` (page 283)

Declared In
`NSCoder.h`

encodeBytes:length:forKey:

Encodes a buffer of data, *bytesp*, whose length is specified by *lenv*, and associates it with the string *key*.

```
- (void)encodeBytes:(const uint8_t *)bytesp length:(NSUInteger)lenv forKey:(NSString
    *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

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

See Also
`- decodeBytesForKey:returnedLength:` (page 275)

Declared In
`NSCoder.h`

encodeConditionalObject:

Can be overridden by subclasses to conditionally encode *object*, preserving common references to that object.

```
- (void)encodeConditionalObject:(id)object
```

Discussion
In the overriding method, *object* should be encoded only if it's unconditionally encoded elsewhere (with any other `encode...Object:` method).

This method must be matched by a subsequent `decodeObject` (page 279) message. Upon decoding, if *object* was never encoded unconditionally, `decodeObject` returns `nil` in place of *object*. However, if *object* was encoded unconditionally, all references to *object* must be resolved.

`NSCoder`'s implementation simply invokes `encodeObject:` (page 289).

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

See Also
- `encodeRootObject:` (page 292)
- `encodeObject:` (page 289)
- `encodeBycopyObject:` (page 284)
- `encodeConditionalObject:` (page 101) (`NSArchiver`)

Declared In
`NSCoder.h`

encodeConditionalObject:forKey:

Conditionally encodes a reference to *objv* and associates it with the string *key* only if *objv* has been unconditionally encoded with `encodeObject:forKey:` (page 290).

```
- (void)encodeConditionalObject:(id)objv forKey:(NSString *)key
```

Discussion
Subclasses must override this method if they support keyed coding.

The encoded object is decoded with the `decodeObjectForKey:` (page 279) method. If *objv* was never encoded unconditionally, `decodeObjectForKey:` (page 279) returns `nil` in place of *objv*.

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

Related Sample Code
IBFragmentView
Reducer

Declared In
`NSCoder.h`

encodeDataObject:

Encodes a given `NSData` object.

```
- (void)encodeDataObject:(NSData *)data
```

Discussion
Subclasses must override this method.

This method must be matched by a subsequent `decodeDataObject` (page 276) message.

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

See Also
- `encodeObject:` (page 289)

Declared In
`NSCoder.h`

encodeDouble:forKey:

Encodes *realv* and associates it with the string *key*.

```
- (void)encodeDouble:(double)realv forKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

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

See Also
- `decodeDoubleForKey:` (page 276)
- `decodeFloatForKey:` (page 276)

Related Sample Code
QTQuartzPlayer
Squiggles

Declared In
`NSCoder.h`

encodeFloat:forKey:

Encodes *realv* and associates it with the string *key*.

```
- (void)encodeFloat:(float)realv forKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

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

See Also
- `decodeFloatForKey:` (page 276)
- `decodeDoubleForKey:` (page 276)

Related Sample Code
iSpend

Declared In
`NSCoder.h`

encodeInt32:forKey:

Encodes the 32-bit integer *intv* and associates it with the string *key*.

```
- (void)encodeInt32:(int32_t)intv forKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

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

See Also
- `decodeIntForKey:` (page 278)
- `decodeIntegerForKey:` (page 278)
- `decodeInt32ForKey:` (page 277)
- `decodeInt64ForKey:` (page 277)

Declared In
`NSCoder.h`

encodeInt64:forKey:

Encodes the 64-bit integer *intv* and associates it with the string *key*.

```
- (void)encodeInt64:(int64_t)intv forKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

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

See Also
- `decodeIntForKey:` (page 278)
- `decodeIntegerForKey:` (page 278)
- `decodeInt32ForKey:` (page 277)
- `decodeInt64ForKey:` (page 277)

Declared In
`NSCoder.h`

encodeInt:forKey:

Encodes *intv* and associates it with the string *key*.

```
- (void)encodeInt:(int)intv forKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

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

See Also
- `decodeIntForKey:` (page 278)
- `decodeIntegerForKey:` (page 278)
- `decodeInt32ForKey:` (page 277)
- `decodeInt64ForKey:` (page 277)

Related Sample Code
Reducer

Declared In
NSCoder.h

encodeInteger:forKey:

Encodes a given NSInteger and associates it with a given key.

```
- (void)encodeInteger:(NSInteger)intv forKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

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

See Also
- decodeIntForKey: (page 278)
- decodeIntegerForKey: (page 278)
- decodeInt32ForKey: (page 277)
- decodeInt64ForKey: (page 277)

Declared In
NSCoder.h

encodeNXObject:

Encodes an old-style object onto the coder. (Deprecated in Mac OS X v10.5.)

```
- (void)encodeNXObject:(id)object
```

Discussion
No sharing is done across separate encodeNXObject: invocations. Callers must have implemented an encodeWithCoder: (page 2034), which parallels the write: methods, on all of their classes that may be touched by this operation.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
NSCoder.h

encodeObject:

Encodes *object*.

```
- (void)encodeObject:(id)object
```

Discussion
`NSCoder`'s implementation simply invokes `encodeValueOfObjCType:at:` (page 293) to encode *`object`*. Subclasses can override this method to encode a reference to *`object`* instead of *`object`* itself. For example, `NSArchiver` detects duplicate objects and encodes a reference to the original object rather than encode the same object twice.

This method must be matched by a subsequent `decodeObject` (page 279) message.

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

See Also
- `encodeRootObject:` (page 292)
- `encodeConditionalObject:` (page 285)
- `encodeBycopyObject:` (page 284)

Related Sample Code
bMoviePalette
bMoviePaletteCocoa
Clock Control
StickiesExample

Declared In
`NSCoder.h`

encodeObject:forKey:

Encodes the object *`objv`* and associates it with the string *`key`*.

```
- (void)encodeObject:(id)objv forKey:(NSString *)key
```

Discussion
Subclasses must override this method to identify multiple encodings of *`objv`* and encode a reference to *`objv`* instead. For example, `NSKeyedArchiver` detects duplicate objects and encodes a reference to the original object rather than encode the same object twice.

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

See Also
- `decodeObjectForKey:` (page 279)

Related Sample Code
IBFragmentView
iSpend
Mountains
Squiggles
StickiesExample

Declared In
`NSCoder.h`

encodePoint:

Encodes *point*.

```
- (void)encodePoint:(NSPoint)point
```

Discussion
NSCoder's implementation invokes encodeValueOfObjCType:at: (page 293) to encode *point*.

This method must be matched by a subsequent decodePoint (page 280) message.

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

Declared In
NSGeometry.h

encodePoint:forKey:

Encodes *point* and associates it with the string *key*.

```
- (void)encodePoint:(NSPoint)point forKey:(NSString *)key
```

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

See Also
- decodePointForKey: (page 280)

Declared In
NSKeyedArchiver.h

encodePropertyList:

Encodes the property list *aPropertyList*.

```
- (void)encodePropertyList:(id)aPropertyList
```

Discussion
NSCoder's implementation invokes encodeValueOfObjCType:at: (page 293) to encode *aPropertyList*.

This method must be matched by a subsequent decodePropertyList (page 280) message.

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

Declared In
NSCoder.h

encodeRect:

Encodes *rect*.

```
- (void)encodeRect:(NSRect)rect
```

Discussion
`NSCoder`'s implementation invokes `encodeValueOfObjCType:at:` (page 293) to encode *rect*.

This method must be matched by a subsequent `decodeRect` (page 280) message.

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

Declared In
`NSGeometry.h`

encodeRect:forKey:

Encodes *rect* and associates it with the string *key*.

```
- (void)encodeRect:(NSRect)rect forKey:(NSString *)key
```

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

See Also
- `decodeRectForKey:` (page 281)

Declared In
`NSKeyedArchiver.h`

encodeRootObject:

Can be overridden by subclasses to encode an interconnected group of Objective-C objects, starting with *rootObject*.

```
- (void)encodeRootObject:(id)rootObject
```

Discussion
`NSCoder`'s implementation simply invokes `encodeObject:` (page 289).

This method must be matched by a subsequent `decodeObject` (page 279) message.

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

See Also
- `encodeObject:` (page 289)
- `encodeConditionalObject:` (page 285)
- `encodeBycopyObject:` (page 284)
- `encodeRootObject:` (page 101) (`NSArchiver`)

Declared In
`NSCoder.h`

encodeSize:

Encodes *size*.

```
- (void)encodeSize:(NSSize)size
```

Discussion
`NSCoder`'s implementation invokes `encodeValueOfObjCType:at:` (page 293) to encode *size*.

This method must be matched by a subsequent `decodeSize` (page 281) message.

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

Declared In
`NSGeometry.h`

encodeSize:forKey:

Encodes *size* and associates it with the string *key*.

```
- (void)encodeSize:(NSSize)size forKey:(NSString *)key
```

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

See Also
`- decodeSizeForKey:` (page 281)

Related Sample Code
Reducer

Declared In
`NSKeyedArchiver.h`

encodeValueOfObjCType:at:

Must be overridden by subclasses to encode a single value residing at *address*, whose Objective-C type is given by *valueType*.

```
- (void)encodeValueOfObjCType:(const char *)valueType at:(const void *)address
```

Discussion
valueType must contain exactly one type code.

This method must be matched by a subsequent `decodeValueOfObjCType:at:` (page 282) message.

For information on creating an Objective-C type code suitable for *valueType*, see the "Type Encodings" section in "The Objective-C Runtime System" chapter of *The Objective-C 2.0 Programming Language*.

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

See Also
- `encodeArrayOfObjCType:count:at:` (page 283)
- `encodeValuesOfObjCTypes:` (page 294)

Declared In
`NSCoder.h`

encodeValuesOfObjCTypes:

Encodes a series of values of potentially differing Objective-C types.

```
- (void)encodeValuesOfObjCTypes:(const char *)valueTypes, ...
```

Discussion
`valueTypes` is a single string containing any number of type codes. The variable arguments to this method consist of one or more pointer arguments, each of which specifies a buffer containing the value to be encoded. For each type code in *`valueTypes`*, you must specify a corresponding pointer argument.

This method must be matched by a subsequent `decodeValuesOfObjCTypes:` (page 282) message.

`NSCoder`'s implementation invokes `encodeValueOfObjCType:at:` (page 293) to encode individual types. Subclasses that implement the `encodeValueOfObjCType:at:` (page 293) method do not need to override this method. However, subclasses that provide a more efficient approach for encoding a series of values may override this method to implement that approach.

For information on creating Objective-C type codes suitable for valueTypes, see the "Type Encodings" section in "The Objective-C Runtime System" chapter of *The Objective-C 2.0 Programming Language*.

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

See Also
- `encodeArrayOfObjCType:count:at:` (page 283)
- `encodeValueOfObjCType:at:` (page 293)

Declared In
`NSCoder.h`

objectZone

Returns the memory zone used to allocate decoded objects.

```
- (NSZone *)objectZone
```

Discussion
`NSCoder`'s implementation simply returns the default memory zone, as given by `NSDefaultMallocZone()`.

Subclasses must override this method and the `setObjectZone:` (page 295) method to allow objects to be decoded into a zone other than the default zone. In its overriding implementation of this method, your subclass should return the current memory zone (if one has been set) or the default zone (if no other zone has been set).

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

Declared In
NSCoder.h

setObjectZone:

NSCoder's implementation of this method does nothing.

```
- (void)setObjectZone:(NSZone *)zone
```

Discussion
Can be overridden by subclasses to set the memory zone used to allocate decoded objects.

Subclasses must override this method and objectZone (page 294) to allow objects to be decoded into a zone other than the default zone. In its overriding implementation of this method, your subclass should store a reference to the current memory zone.

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

Declared In
NSCoder.h

systemVersion

During encoding, this method should return the system version currently in effect.

```
- (unsigned)systemVersion
```

Discussion
During decoding, this method should return the version that was in effect when the data was encoded.

By default, this method returns the current system version, which is appropriate for encoding but not for decoding. Subclasses that implement decoding must override this method to return the system version of the data being decoded.

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

Declared In
NSCoder.h

versionForClassName:

Returns the version in effect for the class with a given name.

```
- (NSInteger)versionForClassName:(NSString *)className
```

Return Value
The version in effect for the class named *className* or NSNotFound if no class named *className* exists.

Discussion

When encoding, this method returns the current version number of the class. When decoding, this method returns the version number of the class being decoded. Subclasses must override this method.

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 Mac OS X v10.0 and later.

See Also

`+ setVersion:` (page 1166) (`NSObject`)

`+ version` (page 1167) (`NSObject`)

Declared In

`NSCoder.h`

NSComparisonPredicate Class Reference

Inherits from	NSPredicate : NSObject
Conforms to	NSCoding (NSPredicate) NSCopying (NSPredicate) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.4 and later.
Declared in	Foundation/NSComparisonPredicate.h
Companion guide	Predicate Programming Guide
Related sample code	CoreRecipes iSpend PredicateEditorSample

Overview

`NSComparisonPredicate` is a subclass of `NSPredicate` used to compare expressions.

Comparison predicates are predicates used to compare the results of two expressions. Comparison predicates take an operator, a left expression, and a right expression, and return as a `BOOL` the result of invoking the operator with the results of evaluating the expressions. Expressions are represented by instances of the `NSExpression` class.

Tasks

Constructors

`+ predicateWithLeftExpression:rightExpression:customSelector:` (page 298)
Returns a new predicate formed by combining the left and right expressions using a given selector.

`+ predicateWithLeftExpression:rightExpression:modifier:type:options:` (page 299)
Creates and returns a predicate of a given type formed by combining given left and right expressions using a given modifier and options.

`- initWithLeftExpression:rightExpression:customSelector:` (page 300)
Initializes a predicate formed by combining given left and right expressions using a given selector.

- `initWithLeftExpression:rightExpression:modifier:type:options:` (page 301)
  Initializes a predicate to a given type formed by combining given left and right expressions using a given modifier and options.

Getting Information About a Comparison Predicate

- `comparisonPredicateModifier` (page 299)
  Returns the comparison predicate modifier for the receiver.
- `customSelector` (page 300)
  Returns the selector for the receiver.
- `leftExpression` (page 301)
  Returns the left expression for the receiver.
- `options` (page 301)
  Returns the options that are set for the receiver.
- `predicateOperatorType` (page 302)
  Returns the predicate type for the receiver.
- `rightExpression` (page 302)
  Returns the right expression for the receiver.

Class Methods

predicateWithLeftExpression:rightExpression:customSelector:

Returns a new predicate formed by combining the left and right expressions using a given selector.

```
+ (NSPredicate *)predicateWithLeftExpression:(NSExpression *)lhs
    rightExpression:(NSExpression *)rhs customSelector:(SEL)selector
```

Parameters

lhs
The left hand side expression.

rhs
The right hand side expression.

selector
The selector to use for comparison. The method defined by the selector must take a single argument and return a `BOOL` value.

Return Value
A new predicate formed by combining the left and right expressions using `selector`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSComparisonPredicate.h`

predicateWithLeftExpression:rightExpression:modifier:type:options:

Creates and returns a predicate of a given type formed by combining given left and right expressions using a given modifier and options.

```
+ (NSPredicate *)predicateWithLeftExpression:(NSExpression *)lhs
    rightExpression:(NSExpression *)rhs
    modifier:(NSComparisonPredicateModifier)modifier
    type:(NSPredicateOperatorType)type options:(NSUInteger)options
```

Parameters

lhs
The left hand expression.

rhs
The right hand expression.

modifier
The modifier to apply.

type
The predicate operator type.

options
The options to apply (see NSComparisonPredicate Options (page 303)).

Return Value
A new predicate of type *type* formed by combining the given left and right expressions using the *modifier* and *options*.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
iSpend
PredicateEditorSample

Declared In
NSComparisonPredicate.h

Instance Methods

comparisonPredicateModifier

Returns the comparison predicate modifier for the receiver.

```
- (NSComparisonPredicateModifier)comparisonPredicateModifier
```

Return Value
The comparison predicate modifier for the receiver.

Discussion
The default value is `NSDirectPredicateModifier` (page 303).

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
PredicateEditorSample

Declared In
`NSComparisonPredicate.h`

customSelector

Returns the selector for the receiver.

```
- (SEL)customSelector
```

Return Value
The selector for the receiver, or `NULL` if there is none.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSComparisonPredicate.h`

initWithLeftExpression:rightExpression:customSelector:

Initializes a predicate formed by combining given left and right expressions using a given selector.

```
- (id)initWithLeftExpression:(NSExpression *)lhs rightExpression:(NSExpression
    *)rhs customSelector:(SEL)selector
```

Parameters

lhs
The left hand expression.

rhs
The right hand expression.

selector
The selector to use. The method defined by the selector must take a single argument and return a `BOOL` value.

Return Value
The receiver, initialized by combining the left and right expressions using *selector*.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSComparisonPredicate.h`

initWithLeftExpression:rightExpression:modifier:type:options:

Initializes a predicate to a given type formed by combining given left and right expressions using a given modifier and options.

```
- (id)initWithLeftExpression:(NSExpression *)lhs rightExpression:(NSExpression
    *)rhs modifier:(NSComparisonPredicateModifier)modifier
    type:(NSPredicateOperatorType)type options:(NSUInteger)options
```

Parameters

lhs
The left hand expression.

rhs
The right hand expression.

modifier
The modifier to apply.

type
The predicate operator type.

options
The options to apply (see NSComparisonPredicate Options (page 303)).

Return Value
The receiver, initialized to a predicate of type *type* formed by combining the left and right expressions using the *modifier* and *options*.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSComparisonPredicate.h

leftExpression

Returns the left expression for the receiver.

```
- (NSExpression *)leftExpression
```

Return Value
The left expression for the receiver, or `nil` if there is none.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
PredicateEditorSample

Declared In
NSComparisonPredicate.h

options

Returns the options that are set for the receiver.

```
- (NSUInteger)options
```

Return Value
The options that are set for the receiver.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
PredicateEditorSample

Declared In
`NSComparisonPredicate.h`

predicateOperatorType

Returns the predicate type for the receiver.

```
- (NSPredicateOperatorType)predicateOperatorType
```

Return Value
The predicate type for the receiver.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
CoreRecipes
PredicateEditorSample

Declared In
`NSComparisonPredicate.h`

rightExpression

Returns the right expression for the receiver.

```
- (NSExpression *)rightExpression
```

Return Value
The right expression for the receiver, or `nil` if there is none.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
PredicateEditorSample

Declared In
`NSComparisonPredicate.h`

Constants

NSComparisonPredicateModifier

These constants describe the possible types of modifier for `NSComparisonPredicate`.

```
typedef enum {
    NSDirectPredicateModifier = 0,
    NSAllPredicateModifier,
    NSAnyPredicateModifier,
} NSComparisonPredicateModifier;
```

Constants

`NSDirectPredicateModifier`

A predicate to compare directly the left and right hand sides.

Available in Mac OS X v10.4 and later.

Declared in `NSComparisonPredicate.h`.

`NSAllPredicateModifier`

A predicate to compare all entries in the destination of a to-many relationship.

The left hand side must be a collection. The corresponding predicate compares each value in the left hand side with the right hand side, and returns `NO` when it finds the first mismatch—or `YES` if all match.

Available in Mac OS X v10.4 and later.

Declared in `NSComparisonPredicate.h`.

`NSAnyPredicateModifier`

A predicate to match with any entry in the destination of a to-many relationship.

The left hand side must be a collection. The corresponding predicate compares each value in the left hand side against the right hand side and returns `YES` when it finds the first match—or `NO` if no match is found

Available in Mac OS X v10.4 and later.

Declared in `NSComparisonPredicate.h`.

Declared In

`NSComparisonPredicate.h`

NSComparisonPredicate Options

These constants describe the possible types of string comparison for `NSComparisonPredicate`.

```
enum {
    NSCaseInsensitivePredicateOption = 0x01,
    NSDiacriticInsensitivePredicateOption = 0x02,
};
```

Constants

`NSCaseInsensitivePredicateOption`

A case-insensitive predicate.

Available in Mac OS X v10.4 and later.

Declared in `NSComparisonPredicate.h`.

`NSDiacriticInsensitivePredicateOption`

A diacritic-insensitive predicate.

Available in Mac OS X v10.4 and later.

Declared in `NSComparisonPredicate.h`.

Declared In

`NSComparisonPredicate.h`

NSPredicateOperatorType

Defines the type of comparison for `NSComparisonPredicate`.

```
typedef enum {
    NSLessThanPredicateOperatorType = 0,
    NSLessThanOrEqualToPredicateOperatorType,
    NSGreaterThanPredicateOperatorType,
    NSGreaterThanOrEqualToPredicateOperatorType,
    NSEqualToPredicateOperatorType,
    NSNotEqualToPredicateOperatorType,
    NSMatchesPredicateOperatorType,
    NSLikePredicateOperatorType,
    NSBeginsWithPredicateOperatorType,
    NSEndsWithPredicateOperatorType,
    NSInPredicateOperatorType,
    NSCustomSelectorPredicateOperatorType,
    NSContainsPredicateOperatorType,
    NSBetweenPredicateOperatorType
} NSPredicateOperatorType;
```

Constants

`NSLessThanPredicateOperatorType`

A less-than predicate.

Available in Mac OS X v10.4 and later.

Declared in `NSComparisonPredicate.h`.

`NSLessThanOrEqualToPredicateOperatorType`

A less-than-or-equal-to predicate.

Available in Mac OS X v10.4 and later.

Declared in `NSComparisonPredicate.h`.

`NSGreaterThanPredicateOperatorType`
A greater-than predicate.

Available in Mac OS X v10.4 and later.

Declared in `NSComparisonPredicate.h`.

`NSGreaterThanOrEqualToPredicateOperatorType`
A greater-than-or-equal-to predicate.

Available in Mac OS X v10.4 and later.

Declared in `NSComparisonPredicate.h`.

`NSEqualToPredicateOperatorType`
An equal-to predicate.

Available in Mac OS X v10.4 and later.

Declared in `NSComparisonPredicate.h`.

`NSNotEqualToPredicateOperatorType`
A not-equal-to predicate.

Available in Mac OS X v10.4 and later.

Declared in `NSComparisonPredicate.h`.

`NSMatchesPredicateOperatorType`
A full regular expression matching predicate.

Available in Mac OS X v10.4 and later.

Declared in `NSComparisonPredicate.h`.

`NSLikePredicateOperatorType`
A simple subset of the matches predicate, similar in behavior to SQL `LIKE`.

Available in Mac OS X v10.4 and later.

Declared in `NSComparisonPredicate.h`.

`NSBeginsWithPredicateOperatorType`
A begins-with predicate.

Available in Mac OS X v10.4 and later.

Declared in `NSComparisonPredicate.h`.

`NSEndsWithPredicateOperatorType`
An ends-with predicate.

Available in Mac OS X v10.4 and later.

Declared in `NSComparisonPredicate.h`.

`NSInPredicateOperatorType`
A predicate to determine if the left hand side is in the right hand side.

For strings, returns `YES` if the left hand side is a substring of the right hand side . For collections, returns `YES` if the left hand side is in the right hand side .

Available in Mac OS X v10.4 and later.

Declared in `NSComparisonPredicate.h`.

`NSCustomSelectorPredicateOperatorType`
Predicate that uses a custom selector that takes a single argument and returns a `BOOL` value.

The selector is invoked on the left hand side with the right hand side.

Available in Mac OS X v10.4 and later.

Declared in `NSComparisonPredicate.h`.

`NSContainsPredicateOperatorType`

A predicate to determine if the left hand side contains the right hand side.

Returns `YES` if `[lhs contains rhs]`; the left hand side must be an `NSExpression` object that evaluates to a collection

Available in Mac OS X v10.5 and later.

Declared in `NSComparisonPredicate.h`.

`NSBetweenPredicateOperatorType`

A predicate to determine if the right hand side lies between bounds specified by the left hand side.

Returns `YES` if `[lhs between rhs]`; the right hand side must be an array in which the first element sets the lower bound and the second element the upper, inclusive. Comparison is performed using compare: or the class-appropriate equivalent.

Available in Mac OS X v10.5 and later.

Declared in `NSComparisonPredicate.h`.

Declared In

`NSComparisonPredicate.h`

NSCompoundPredicate Class Reference

Inherits from	NSPredicate : NSObject
Conforms to	NSCoding (NSPredicate) NSCopying (NSPredicate) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.4 and later.
Declared in	Foundation/NSCompoundPredicate.h
Companion guide	Predicate Programming Guide
Related sample code	CoreRecipes iSpend PredicateEditorSample SpotlightFortunes

Overview

`NSCompoundPredicate` is a subclass of `NSPredicate` used to represent logical "gate" operations (AND/OR/NOT) and comparison operations.

Comparison operations are based on two expressions, as represented by instances of the `NSExpression` class. Expressions are created for constant values, key paths, and so on.

On Mac OS X v10.5 and later, `NSCompoundPredicate` allows you to create an AND or OR compound predicate (but not a NOT compound predicate) using an array with 0,1, or more elements. A compound predicate with 0 elements evaluates to TRUE, and a compound predicate with a single sub-predicate evaluates to the truth of its sole subpredicate.

Tasks

Constructors

`+ andPredicateWithSubpredicates:` (page 308)
Returns a new predicate formed by AND-ing the predicates in a given array.

+ notPredicateWithSubpredicate: (page 308)
Returns a new predicate formed by NOT-ing a given predicate.

+ orPredicateWithSubpredicates: (page 309)
Returns a new predicate formed by OR-ing the predicates in a given array.

- initWithType:subpredicates: (page 310)
Returns the receiver initialized to a given type using predicates from a given array.

Getting Information About a Compound Predicate

- compoundPredicateType (page 309)
Returns the predicate type for the receiver.

- subpredicates (page 310)
Returns the array of the receiver's subpredicates.

Class Methods

andPredicateWithSubpredicates:

Returns a new predicate formed by AND-ing the predicates in a given array.

```
+ (NSPredicate *)andPredicateWithSubpredicates:(NSArray *)subpredicates
```

Parameters

subpredicates
An array of NSPredicate objects.

Return Value
A new predicate formed by AND-ing the predicates specified by *subpredicates*.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
iSpend
PredicateEditorSample
SpotlightFortunes

Declared In
NSCompoundPredicate.h

notPredicateWithSubpredicate:

Returns a new predicate formed by NOT-ing a given predicate.

```
+ (NSPredicate *)notPredicateWithSubpredicate:(NSPredicate *)predicate
```

Parameters

predicate

A predicate.

Return Value

A new predicate formed by NOT-ing the predicate specified by *predicate*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

NSCompoundPredicate.h

orPredicateWithSubpredicates:

Returns a new predicate formed by OR-ing the predicates in a given array.

```
+ (NSPredicate *)orPredicateWithSubpredicates:(NSArray *)subpredicates
```

Parameters

subpredicates

An array of `NSPredicate` objects.

Return Value

A new predicate formed by OR-ing the predicates specified by *subpredicates*.

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

iSpend

Declared In

NSCompoundPredicate.h

Instance Methods

compoundPredicateType

Returns the predicate type for the receiver.

```
- (NSCompoundPredicateType)compoundPredicateType
```

Return Value

The predicate type for the receiver.

Availability

Available in Mac OS X v10.4 and later.

Declared In

NSCompoundPredicate.h

initWithType:subpredicates:

Returns the receiver initialized to a given type using predicates from a given array.

```
- (id)initWithType:(NSCompoundPredicateType)type subpredicates:(NSArray
    *)subpredicates
```

Parameters

type

The type of the new predicate.

subpredicates

An array of `NSPredicate` objects.

Return Value
The receiver initialized with its type set to `type` and subpredicates array to *subpredicates*.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
PredicateEditorSample

Declared In
`NSCompoundPredicate.h`

subpredicates

Returns the array of the receiver's subpredicates.

```
- (NSArray *)subpredicates
```

Return Value
The array of the receiver's subpredicates.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSCompoundPredicate.h`

Constants

Compound Predicate Types

These constants describe the possible types of `NSCompoundPredicate`.

```
typedef enum {
    NSNotPredicateType = 0,
    NSAndPredicateType,
    NSOrPredicateType,
} NSCompoundPredicateType;
```

Constants

`NSNotPredicateType`

A logical NOT predicate.

Available in Mac OS X v10.4 and later.

Declared in `NSCompoundPredicate.h`.

`NSAndPredicateType`

A logical AND predicate.

Available in Mac OS X v10.4 and later.

Declared in `NSCompoundPredicate.h`.

`NSOrPredicateType`

A logical OR predicate.

Available in Mac OS X v10.4 and later.

Declared in `NSCompoundPredicate.h`.

Declared In

`NSCompoundPredicate.h`

NSCondition Class Reference

Inherits from	NSObject
Conforms to	NSLocking NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	Foundation/NSLock.h
Companion guide	Threading Programming Guide

Overview

The `NSCondition` class implements a condition variable whose semantics follow those used for POSIX-style conditions. A condition object acts as both a lock and a checkpoint in a given thread. The lock protects your code while it tests the condition and performs the task triggered by the condition. The checkpoint behavior requires that the condition be true before the thread proceeds with its task. While the condition is not true, the thread blocks. It remains blocked until another thread signals the condition object.

The semantics for using an `NSCondition` object are as follows:

1. Lock the condition object.
2. Test a boolean predicate. (This predicate is a boolean flag or other variable in your code that indicates whether it is safe to perform the task protected by the condition.)
3. If the boolean predicate is false, call the condition object's `wait` or `waitUntilDate:` method to block the thread. Upon returning from these methods, go to step 2 to retest your boolean predicate. (Continue waiting and retesting the predicate until it is true.)
4. If the boolean predicate is true, perform the task.
5. Optionally update any predicates (or signal any conditions) affected by your task.
6. When your task is done, unlock the condition object.

The pseudocode for performing the preceding steps would therefore look something like the following:

```
lock the condition
while (!(boolean_predicate)) {
    wait on condition
```

```
}
do protected work
(optionally, signal or broadcast the condition again or change a predicate value)
unlock the condition
```

Whenever you use a condition object, the first step is to lock the condition. Locking the condition ensures that your predicate and task code are protected from interference by other threads using the same condition. Once you have completed your task, you can set other predicates or signal other conditions based on the needs of your code. You should always set predicates and signal conditions while holding the condition object's lock.

When a thread waits on a condition, the condition object unlocks its lock and blocks the thread. When the condition is signaled, the system wakes up the thread. The condition object then reacquires its lock before returning from the `wait` or `waitUntilDate:` method. Thus, from the point of view of the thread, it is as if it always held the lock.

A boolean predicate is an important part of the semantics of using conditions because of the way signaling works. Signaling a condition does not guarantee that the condition itself is true. There are timing issues involved in signaling that may cause false signals to appear. Using a predicate ensures that these spurious signals do not cause you to perform work before it is safe to do so. The predicate itself is simply a flag or other variable in your code that you test in order to acquire a Boolean result.

For more information on how to use conditions, see Using POSIX Thread Locks in *Threading Programming Guide*.

Tasks

Waiting for the Lock

- `wait` (page 316)
  Blocks the current thread until the condition is signaled.
- `waitUntilDate:` (page 317)
  Blocks the current thread until the condition is signaled or the specified time limit is reached.

Signaling Waiting Threads

- `signal` (page 316)
  Signals the condition, waking up one thread waiting on it.
- `broadcast` (page 315)
  Signals the condition, waking up all threads waiting on it.

Accessor Methods

- `setName:` (page 315)
  Assigns a name to the receiver.

- name (page 315)
  Returns the name associated with the receiver.

Instance Methods

broadcast

Signals the condition, waking up all threads waiting on it.

```
- (void)broadcast
```

Discussion
If no threads are waiting on the condition, this method does nothing.

To avoid race conditions, you should invoke this method only while the receiver is locked.

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

Declared In
NSLock.h

name

Returns the name associated with the receiver.

```
- (NSString *)name
```

Return Value
The name of the receiver.

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

See Also
- setName: (page 315)

Declared In
NSLock.h

setName:

Assigns a name to the receiver.

```
- (void)setName:(NSString *)newName
```

Parameters
newName
  The new name for the receiver. This method makes a copy of the specified string.

Discussion
You can use a name string to identify a condition object within your code. Cocoa also uses this name as part of any error descriptions involving the receiver.

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

See Also
- `name` (page 315)

Declared In
`NSLock.h`

signal

Signals the condition, waking up one thread waiting on it.

```
- (void)signal
```

Discussion
You use this method to wake up one thread that is waiting on the condition. You may call this method multiple times to wake up multiple threads. If no threads are waiting on the condition, this method does nothing.

To avoid race conditions, you should invoke this method only while the receiver is locked.

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

Declared In
`NSLock.h`

wait

Blocks the current thread until the condition is signaled.

```
- (void)wait
```

Discussion
You must lock the receiver prior to calling this method.

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

See Also
- `lock` (page 2091) (`NSLocking`)

Declared In
`NSLock.h`

waitUntilDate:

Blocks the current thread until the condition is signaled or the specified time limit is reached.

```
- (BOOL)waitUntilDate:(NSDate *)limit
```

Parameters

limit

The time at which to wake up the thread if the condition has not been signaled.

Return Value

`YES` if the condition was signaled; otherwise, `NO` if the time limit was reached.

Discussion

You must lock the receiver prior to calling this method.

Availability

Available in Mac OS X v10.5 and later.

See Also

`- lock` (page 2091) (`NSLocking`)

Declared In

`NSLock.h`

NSConditionLock Class Reference

Inherits from	NSObject
Conforms to	NSLocking NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSLock.h
Companion guide	Threading Programming Guide
Related sample code	QTAudioExtractionPanel Vertex Optimization

Overview

The `NSConditionLock` class defines objects whose locks can be associated with specific, user-defined conditions. Using an `NSConditionLock` object, you can ensure that a thread can acquire a lock only if a certain condition is met. Once it has acquired the lock and executed the critical section of code, the thread can relinquish the lock and set the associated condition to something new. The conditions themselves are arbitrary: you define them as needed for your application.

Adopted Protocols

NSLocking
- `lock` (page 2091)
- `unlock` (page 2092)

Tasks

Initializing an NSConditionLock Object

`- initWithCondition:` (page 321)
Initializes a newly allocated `NSConditionLock` object and sets its condition.

Returning the Condition

- `condition` (page 320)
  Returns the condition associated with the receiver.

Acquiring and Releasing a Lock

- `lockBeforeDate:` (page 321)
  Attempts to acquire a lock before a specified moment in time.
- `lockWhenCondition:` (page 321)
  Attempts to acquire a lock.
- `lockWhenCondition:beforeDate:` (page 322)
  Attempts to acquire a lock before a specified moment in time.
- `tryLock` (page 323)
  Attempts to acquire a lock without regard to the receiver's condition.
- `tryLockWhenCondition:` (page 324)
  Attempts to acquire a lock if the receiver's condition is equal to the specified condition.
- `unlockWithCondition:` (page 324)
  Relinquishes the lock and sets the receiver's condition.

Accessor Methods

- `setName:` (page 323)
  Assigns a name to the receiver.
- `name` (page 322)
  Returns the name associated with the receiver.

Instance Methods

condition

Returns the condition associated with the receiver.

```
- (NSInteger)condition
```

Return Value
The condition associated with the receiver. If no condition has been set, returns 0.

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

Declared In
`NSLock.h`

initWithCondition:

Initializes a newly allocated `NSConditionLock` object and sets its condition.

```
- (id)initWithCondition:(NSInteger)condition
```

Parameters

condition

The user-defined condition for the lock. The value of *condition* is user-defined; see the class description for more information.

Return Value
An initialized condition lock object; may be different than the original receiver.

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

Related Sample Code
QTAudioExtractionPanel
Vertex Optimization

Declared In
`NSLock.h`

lockBeforeDate:

Attempts to acquire a lock before a specified moment in time.

```
- (BOOL)lockBeforeDate:(NSDate *)limit
```

Parameters

limit

The date by which the lock must be acquired or the attempt will time out.

Return Value
`YES` if the lock is acquired within the time limit, `NO` otherwise.

Discussion
The condition associated with the receiver isn't taken into account in this operation. This method blocks the thread's execution until the receiver acquires the lock or *limit* is reached.

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

See Also
`- lockWhenCondition:beforeDate:` (page 322)

Declared In
`NSLock.h`

lockWhenCondition:

Attempts to acquire a lock.

```
- (void)lockWhenCondition:(NSInteger)condition
```

Parameters

condition

The condition to match on.

Discussion

The receiver's condition must be equal to *condition* before the locking operation will succeed. This method blocks the thread's execution until the lock can be acquired.

Availability

Available in Mac OS X v10.0 and later.

See Also

- lockWhenCondition:beforeDate: (page 322)
- unlockWithCondition: (page 324)

Declared In

NSLock.h

lockWhenCondition:beforeDate:

Attempts to acquire a lock before a specified moment in time.

```
- (BOOL)lockWhenCondition:(NSInteger)condition beforeDate:(NSDate *)limit
```

Parameters

condition

The condition to match on.

limit

The date by which the lock must be acquired or the attempt will time out.

Return Value

YES if the lock is acquired within the time limit, NO otherwise.

Discussion

The receiver's condition must be equal to *condition* before the locking operation will succeed. This method blocks the thread's execution until the lock can be acquired or *limit* is reached.

Availability

Available in Mac OS X v10.0 and later.

See Also

- lockBeforeDate: (page 321)
- lockWhenCondition: (page 321)

Declared In

NSLock.h

name

Returns the name associated with the receiver.

```
- (NSString *)name
```

Return Value
The name of the receiver.

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

See Also
- `setName:` (page 323)

Declared In
`NSLock.h`

setName:

Assigns a name to the receiver.

```
- (void)setName:(NSString *)newName
```

Parameters

newName
The new name for the receiver. This method makes a copy of the specified string.

Discussion
You can use a name string to identify a condition lock within your code. Cocoa also uses this name as part of any error descriptions involving the receiver.

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

See Also
- `name` (page 322)

Declared In
`NSLock.h`

tryLock

Attempts to acquire a lock without regard to the receiver's condition.

```
- (BOOL)tryLock
```

Return Value
`YES` if the lock could be acquired, `NO` otherwise.

Discussion
This method returns immediately.

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

See Also
- `tryLockWhenCondition:` (page 324)

Declared In
NSLock.h

tryLockWhenCondition:

Attempts to acquire a lock if the receiver's condition is equal to the specified condition.

```
- (BOOL)tryLockWhenCondition:(NSInteger)condition
```

Return Value
YES if the lock could be acquired, NO otherwise.

Discussion
As part of its implementation, this method invokes lockWhenCondition:beforeDate: (page 322). This method returns immediately.

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

See Also
- tryLock (page 323)

Declared In
NSLock.h

unlockWithCondition:

Relinquishes the lock and sets the receiver's condition.

```
- (void)unlockWithCondition:(NSInteger)condition
```

Parameters
condition
The user-defined condition for the lock. The value of *condition* is user-defined; see the class description for more information.

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

See Also
- lockWhenCondition: (page 321)

Declared In
NSLock.h

NSConnection Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSConnection.h
Companion guide	Distributed Objects Programming Topics
Related sample code	SimpleThreads TrivialThreads

Overview

An `NSConnection` object manages the communication between objects in different threads or between a thread and a process running on a local or remote system. Connection objects form the backbone of the distributed objects mechanism and normally operate in the background. You use the methods of `NSConnection` explicitly when vending an object to other applications, when accessing such a vended object through a proxy, and when altering default communication parameters. At other times, you simply interact with a vended object or its proxy.

In Mac OS X v10.5 and later, a single connection object may be shared by multiple threads and used to access a vended object by default. Prior to Mac OS X v10.5, a separate connection object must be maintained by each thread by default; however, an application can enable sharing by invoking the `enableMultipleThreads` method of the object.

Tasks

Getting the Default Instance

`+ defaultConnection` (page 331)
Returns the default `NSConnection` object for the current thread.

Creating Instances

+ `connectionWithReceivePort:sendPort:` (page 329)
  Returns an `NSConnection` object that communicates using given send and receive ports.

- `initWithReceivePort:sendPort:` (page 337)
  Returns an `NSConnection` object initialized with given send and receive ports.

Running the Connection in a New Thread

- `runInNewThread` (page 345)
  Creates and starts a new `NSThread` object and then runs the receiving connection in the new thread.

- `enableMultipleThreads` (page 336)
  Configures the receiver to allow requests from multiple threads to the remote object, without requiring each thread to each maintain its own connection.

- `multipleThreadsEnabled` (page 339)
  Returns a Boolean value that indicates whether the receiver supports requests from multiple threads.

- `addRunLoop:` (page 335)
  Adds the specified run loop to the list of run loops the receiver monitors and from which it responds to requests.

- `removeRunLoop:` (page 342)
  Removes a given `NSRunLoop` object from the list of run loops the receiver monitors and from which it responds to requests.

Vending a Service

+ `serviceConnectionWithName:rootObject:usingNameServer:` (page 334)
  Creates and returns a new connection object representing a vended service on the specified port name server.

+ `serviceConnectionWithName:rootObject:` (page 333)
  Creates and returns a new connection object representing a vended service on the default system port name server.

- `registerName:` (page 340)
  Registers the specified service using with the default system port name server.

- `registerName:withNameServer:` (page 341)
  Registers a service with the specified port name server.

- `setRootObject:` (page 347)
  Sets the object that the receiver makes available to other applications or threads.

- `rootObject` (page 344)
  Returns the object that the receiver (or its parent) makes available to other applications or threads.

Getting a Remote Object

+ `connectionWithRegisteredName:host:` (page 330)
  Returns the `NSConnection` object whose send port links it to the `NSConnection` object registered with the default `NSPortNameServer` under a given name on a given host.

+ `connectionWithRegisteredName:host:usingNameServer:` (page 331)
  Returns the `NSConnection` object whose send port links it to the `NSConnection` object registered under a given name with a given server on a given host.

- `rootProxy` (page 344)
  Returns the proxy for the root object of the receiver's peer in another application or thread.

+ `rootProxyForConnectionWithRegisteredName:host:` (page 332)
  Returns a proxy for the root object of the `NSConnection` object registered with the default `NSPortNameServer` under a given name on a given host.

+ `rootProxyForConnectionWithRegisteredName:host:usingNameServer:` (page 333)
  Returns a proxy for the root object of the `NSConnection` object registered with *`server`* under *`name`* on a given host.

- `remoteObjects` (page 342)
  Returns all the local proxies for remote objects that have been received over the connection but not deallocated yet.

- `localObjects` (page 339)
  Returns the local objects that have been sent over the connection and still have proxies at the other end.

Getting a Conversation

+ `currentConversation` (page 331)
  Returns a token object representing any conversation in progress in the current thread.

Getting All NSConnection Objects

+ `allConnections` (page 329)
  Returns all valid `NSConnection` objects in the process.

Configuring Instances

- `setRequestTimeout:` (page 347)
  Sets the timeout interval for outgoing remote messages.

- `requestTimeout` (page 344)
  Returns the timeout interval for outgoing remote messages.

- `setReplyTimeout:` (page 347)
  Sets the timeout interval for replies to outgoing remote messages

- `replyTimeout` (page 343)
  Returns the timeout interval for replies to outgoing remote messages.

- `setIndependentConversationQueueing:` (page 346)
  Sets a Boolean value that specifies whether the receiver handles remote messages atomically.
- `independentConversationQueueing` (page 336)
  Returns a Boolean value that indicates whether the receiver handles remote messages atomically.
- `addRequestMode:` (page 335)
  Adds *mode* to the set of run-loop input modes that the receiver uses for connection requests.
- `removeRequestMode:` (page 342)
  Removes *mode* from the set of run-loop input modes the receiver uses for connection requests.
- `requestModes` (page 343)
  Returns the set of request modes the receiver's receive port is registered for with its `NSRunLoop` object.
- `invalidate` (page 338)
  Invalidates (but doesn't release) the receiver.
- `isValid` (page 338)
  Returns a Boolean value that indicates whether the receiver is known to be valid.

Getting Ports

- `receivePort` (page 340)
  Returns the `NSPort` object on which the receiver receives incoming network messages.
- `sendPort` (page 345)
  Returns the `NSPort` object that the receiver sends outgoing network messages through.

Getting Statistics

- `statistics` (page 348)
  Returns an `NSDictionary` object containing various statistics for the receiver.

Setting the Delegate

- `setDelegate:` (page 346)
  Sets the receiver's delegate.
- `delegate` (page 336)
  Returns the receiver's delegate.

Authenticating

- `authenticateComponents:withData:` (page 348) *delegate method*
  Returns a Boolean value that indicates whether given authentication data is valid for a given set of components.
- `authenticationDataForComponents:` (page 349) *delegate method*
  Returns an `NSData` object to be used as an authentication stamp for an outgoing message.

Responding to a Connection

- `connection:shouldMakeNewConnection:` (page 350) *delegate method*
  Returns a Boolean value that indicates whether the parent connection should allow a given new connection to be created.
- `connection:handleRequest:` (page 350) *delegate method*
  This method should be implemented by `NSConnection` object delegates that want to intercept distant object requests.
- `createConversationForConnection:` (page 351) *delegate method*
  Returns an arbitrary object identifying a new conversation being created for the connection in the current thread.
- `makeNewConnection:sender:` (page 351) *delegate method*
  Returns a Boolean value that indicates whether the parent should allow a given new connection to be created and configured.

Class Methods

allConnections

Returns all valid `NSConnection` objects in the process.

```
+ (NSArray *)allConnections
```

Return Value
An array containing all valid `NSConnection` objects in the process.

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

See Also
`- isValid` (page 338)

Declared In
`NSConnection.h`

connectionWithReceivePort:sendPort:

Returns an `NSConnection` object that communicates using given send and receive ports.

```
+ (id)connectionWithReceivePort:(NSPort *)receivePort sendPort:(NSPort *)sendPort
```

Parameters
receivePort
A receive port.
sendPort
A send port.

Return Value
An `NSConnection` object that communicates using *receivePort* and *sendPort*.

Discussion
See `initWithReceivePort:sendPort:` (page 337) for more information.

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

See Also
`+ defaultConnection` (page 331)

Related Sample Code
SimpleThreads
TrivialThreads

Declared In
`NSConnection.h`

connectionWithRegisteredName:host:

Returns the `NSConnection` object whose send port links it to the `NSConnection` object registered with the default `NSPortNameServer` under a given name on a given host.

```
+ (id)connectionWithRegisteredName:(NSString *)name host:(NSString *)hostName
```

Parameters

name

The name of an `NSConnection` object.

hostName

The name of the host. The domain name *hostName* is an Internet domain name (for example, "`sales.anycorp.com`"). If *hostName* is `nil` or empty, then only the local host is searched for the named `NSConnection` object.

Return Value
The `NSConnection` object whose send port links it to the `NSConnection` object registered with the default `NSPortNameServer` under *name* on the host named *hostName*. Returns `nil` if no `NSConnection` object can be found for *name* and *hostName*.

The returned `NSConnection` object is a child of the default `NSConnection` object for the current thread (that is, it shares the default `NSConnection` object's receive port).

Discussion
To get the object vended by the `NSConnection` object, use the `rootProxy` (page 344) instance method. The `rootProxyForConnectionWithRegisteredName:host:` (page 332) class method immediately returns this object.

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

See Also
`+ defaultConnection` (page 331)
`+ connectionWithRegisteredName:host:usingNameServer:` (page 331)

Declared In
`NSConnection.h`

connectionWithRegisteredName:host:usingNameServer:

Returns the `NSConnection` object whose send port links it to the `NSConnection` object registered under a given name with a given server on a given host.

```
+ (id)connectionWithRegisteredName:(NSString *)name host:(NSString *)hostName
    usingNameServer:(NSPortNameServer *)server
```

Parameters

name
The connection name.

hostName
The host name.

server
The name server.

Return Value
The `NSConnection` object whose send port links it to the `NSConnection` object registered with *server* under *name* on the host named *hostName*.

Discussion
See `connectionWithRegisteredName:host:` (page 330) for more information.

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

Declared In
`NSConnection.h`

currentConversation

Returns a token object representing any conversation in progress in the current thread.

```
+ (id)currentConversation
```

Return Value
A token object representing any conversation in progress in the current thread, or `nil` if there is no conversation in progress.

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

See Also
`- createConversationForConnection:` (page 351)

Declared In
`NSConnection.h`

defaultConnection

Returns the default `NSConnection` object for the current thread.

```
+ (NSConnection *)defaultConnection
```

Return Value
The default `NSConnection` object for the current thread, creating it if necessary.

Discussion
The default `NSConnection` object uses a single `NSPort` object for both receiving and sending and is useful only for vending an object; use the `setRootObject:` (page 347) and `registerName:` (page 340) methods to do this.

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

Declared In
`NSConnection.h`

rootProxyForConnectionWithRegisteredName:host:

Returns a proxy for the root object of the `NSConnection` object registered with the default `NSPortNameServer` under a given name on a given host.

```
+ (NSDistantObject *)rootProxyForConnectionWithRegisteredName:(NSString *)name
    host:(NSString *)hostName
```

Parameters

name
The name under which the connection is registered.

hostName
The host name. The domain name *hostName* is an Internet domain name (for example, `"sales.anycorp.com"`). If *hostName* is `nil` or empty, then only the local host is searched for the named `NSConnection` object.

Return Value
a proxy for the root object of the `NSConnection` object registered with the default `NSPortNameServer` under *name* on the host named *hostName*, or `nil` if that `NSConnection` object has no root object set. Also returns `nil` if no `NSConnection` object can be found for *name* and *hostName*.

Discussion
The `NSConnection` object of the returned proxy is a child of the default `NSConnection` object for the current thread (that is, it shares the default `NSConnection` object's receive port).

This method invokes `connectionWithRegisteredName:host:` (page 330) and sends the resulting `NSConnection` object a `rootProxy` (page 344) message.

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

See Also
`- setRootObject:` (page 347)
`+ rootProxyForConnectionWithRegisteredName:host:usingNameServer:` (page 333)

Declared In
`NSConnection.h`

rootProxyForConnectionWithRegisteredName:host:usingNameServer:

Returns a proxy for the root object of the `NSConnection` object registered with *`server`* under *`name`* on a given host.

```
+ (NSDistantObject *)rootProxyForConnectionWithRegisteredName:(NSString *)name
    host:(NSString *)hostName usingNameServer:(NSPortNameServer *)server
```

Parameters

`name`

The name of an `NSConnection` object .

`hostName`

A host name.

`server`

The server.

Return Value

A proxy for the root object of the `NSConnection` object registered with *`server`* under *`name`* on the host named *`hostName`*, or `nil` if that `NSConnection` object has no root object set.

Discussion

See `rootProxyForConnectionWithRegisteredName:host:` (page 332) for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSConnection.h`

serviceConnectionWithName:rootObject:

Creates and returns a new connection object representing a vended service on the default system port name server.

```
+ (id)serviceConnectionWithName:(NSString *)name rootObject:(id)root
```

Parameters

`name`

The name of the service you want to publish.

`root`

The object to use as the root object for the published service. This is the object vended by the connection.

Return Value

An `NSConnection` object representing the vended service or `nil` if there was a problem setting up the connection object.

Discussion

This method creates the server-side of a connection object and registers it with the default system port name server. Clients wishing to connect to this service can request a communications port from the same port server and use that port to to communicate.

Availability

Available in Mac OS X v10.5 and later.

See Also
+ serviceConnectionWithName:rootObject:usingNameServer: (page 334)
+ connectionWithRegisteredName:host: (page 330)
- rootObject (page 344)
+ systemDefaultPortNameServer (page 1270) (NSPortNameServer)

Declared In
NSConnection.h

serviceConnectionWithName:rootObject:usingNameServer:

Creates and returns a new connection object representing a vended service on the specified port name server.

```
+ (id)serviceConnectionWithName:(NSString *)name rootObject:(id)root
    usingNameServer:(NSPortNameServer *)server
```

Parameters

name
The name of the service you want to publish.

root
The object to use as the root object for the published service. This is the object vended by the connection.

server
The port name server with which to register your service.

Return Value
An NSConnection object representing the vended service or nil if there was a problem setting up the connection object.

Discussion
This method creates the server-side of a connection object and registers it with the specified port name server. Clients wishing to connect to this service can request a communications port from the same port server and use that port to communicate.

If the specified service name corresponds to a service that is autolaunched by launchd, this method allows the service to check in with the launchd process. If the service is not autolaunched by launchd, this method registers the new connection with the specified name. For more information about launchd and its role in launching services, see *System Startup Programming Topics*

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

See Also
+ connectionWithRegisteredName:host:usingNameServer: (page 331)
- rootObject (page 344)

Declared In
NSConnection.h

Instance Methods

addRequestMode:

Adds *mode* to the set of run-loop input modes that the receiver uses for connection requests.

```
- (void)addRequestMode:(NSString *)mode
```

Parameters

mode

The mode to add to the receiver.

Discussion

The default input mode is `NSDefaultRunLoopMode`. See the `NSRunLoop` class specification for more information on input modes.

Availability

Available in Mac OS X v10.0 and later.

See Also

`addPort:forMode:` (page 1333) (NSRunLoop)

Declared In

`NSConnection.h`

addRunLoop:

Adds the specified run loop to the list of run loops the receiver monitors and from which it responds to requests.

```
- (void)addRunLoop:(NSRunLoop *)runloop
```

Parameters

runloop

The run loop to add to the receiver.

Discussion

This method is invoked automatically when a request comes in from a new run loop if `enableMultipleThreads` (page 336) has been set.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `enableMultipleThreads` (page 336)
- `removeRunLoop:` (page 342)

Declared In

`NSConnection.h`

delegate

Returns the receiver's delegate.

```
- (id)delegate
```

Return Value
The receiver's delegate.

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

See Also
`- setDelegate:` (page 346)

Declared In
`NSConnection.h`

enableMultipleThreads

Configures the receiver to allow requests from multiple threads to the remote object, without requiring each thread to each maintain its own connection.

```
- (void)enableMultipleThreads
```

Discussion
In Mac OS X v10.5 and later, multiple thread support is enabled by default and this method does nothing.

Prior to Mac OS X v10.5, multiple thread support is disabled by default and must be enabled explicitly. When disabled, each thread must create its own `NSConnection` object in order to access a given remote object. When enabled, threads may use the same `NSConnection` object to access the remote object. If this feature is disabled and an attempt is made to connect to the receiver from a thread other than the one that created it, the receiver raises an `NSObjectInaccessibleException`.

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

See Also
`- multipleThreadsEnabled` (page 339)

Declared In
`NSConnection.h`

independentConversationQueueing

Returns a Boolean value that indicates whether the receiver handles remote messages atomically.

```
- (BOOL)independentConversationQueueing
```

Return Value
`YES` if the receiver handles remote messages atomically, otherwise `NO`.

Discussion
See Configuring an NSConnection for more information on independent conversation queueing.

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

See Also
- `setIndependentConversationQueueing:` (page 346)

Declared In
`NSConnection.h`

initWithReceivePort:sendPort:

Returns an `NSConnection` object initialized with given send and receive ports.

```
- (id)initWithReceivePort:(NSPort *)receivePort sendPort:(NSPort *)sendPort
```

Parameters

receivePort
The receive port for the new connection.

sendPort
The send port for the new connection.

Return Value
An `NSConnection` object initialized with *receivePort* and *sendPort*. The returned object might be different than the original receiver.

Discussion
The new `NSConnection` object adds *receivePort* to the current `NSRunLoop` object with `NSDefaultRunLoopMode` as the mode. If the application doesn't use an `NSApplication` object to handle events, it needs to run the `NSRunLoop` object with one of its various `run...` messages.

This method posts an `NSConnectionDidInitializeNotification` (page 353) once the connection is initialized.

The *receivePort* and *sendPort* parameters affect initialization as follows:

- If an `NSConnection` object with the same ports already exists, releases the receiver, retains the existing connection, and returns it.
- If an `NSConnection` object exists that uses the same ports, but switched in role, then the new `NSConnection` object communicates with it. Messages sent to a proxy held by either connection are forwarded through the other `NSConnection` object. This rule applies both within and across address spaces.

  This behavior is useful for setting up distributed object connections between threads within an application. See Communicating With Distributed Objects for more information.
- If *receivePort* and *sendPort* are `nil`, deallocates the receiver and returns `nil`.
- If *receivePort* is `nil`, the `NSConnection` object allocates and uses a new port of the same class as *sendPort*.
- If *sendPort* is `nil` or if both ports are the same, the `NSConnection` object uses *receivePort* for both sending and receiving and is useful only for vending an object. Use the `registerName:` (page 340) and `setRootObject:` (page 347) instance methods to vend an object.

- If an `NSConnection` object exists that uses *`receivePort`* as both of its ports, it's treated as the parent of the new `NSConnection` object, and its root object and all its configuration settings are applied to the new `NSConnection` object. You should neither register a name for nor set the root object of the new `NSConnection` object. See Configuring an NSConnection for more information.
- If *`receivePort`* and *`sendPort`* are different and neither is shared with another `NSConnection` object, the receiver can be used to vend an object as well as to communicate with other `NSConnection` objects. However, it has no other `NSConnection` object to communicate with until one is set up.
- The *`receivePort`* parameter can't be shared by `NSConnection` objects in different threads.

This method is the designated initializer for the `NSConnection` class.

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

See Also
`+ defaultConnection` (page 331)

Related Sample Code
SimpleThreads
TrivialThreads

Declared In
`NSConnection.h`

invalidate

Invalidates (but doesn't release) the receiver.

```
- (void)invalidate
```

Discussion
After withdrawing the ports the receiver has registered with the current run loop, `invalidate` posts an `NSConnectionDidDieNotification` (page 352) and then invalidates all remote objects and exported local proxies.

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

See Also
`- isValid` (page 338)
`removePort:forMode:` (page 1337) (NSRunLoop)
`- requestModes` (page 343)

Declared In
`NSConnection.h`

isValid

Returns a Boolean value that indicates whether the receiver is known to be valid.

```
- (BOOL)isValid
```

Return Value
`YES` if the receiver is known to be valid, otherwise `NO`.

Discussion
An `NSConnection` object becomes invalid when either of its ports becomes invalid, but only notes that it has become invalid when it tries to send or receive a message. When this happens it posts an `NSConnectionDidDieNotification` (page 352) to the default notification center.

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

See Also
`- invalidate` (page 338)
`isValid` (page 1252) (NSPort)

Declared In
`NSConnection.h`

localObjects

Returns the local objects that have been sent over the connection and still have proxies at the other end.

```
- (NSArray *)localObjects
```

Return Value
An array containing the local objects that have been sent over the connection and still have proxies at the other end.

Discussion
When an object's remote proxy is deallocated, a message is sent back to the receiver to notify it that the local object is no longer shared over the connection.

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

See Also
`- remoteObjects` (page 342)

Declared In
`NSConnection.h`

multipleThreadsEnabled

Returns a Boolean value that indicates whether the receiver supports requests from multiple threads.

```
- (BOOL)multipleThreadsEnabled
```

Return Value
`YES` if the receiver supports requests from multiple threads.

Discussion
In Mac OS X v10.5 and later, multiple threads are enabled by default.

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

See Also
- `enableMultipleThreads` (page 336)

Declared In
`NSConnection.h`

receivePort

Returns the `NSPort` object on which the receiver receives incoming network messages.

```
- (NSPort *)receivePort
```

Return Value
The `NSPort` object on which the receiver receives incoming network messages.

Discussion
You can inspect this object for debugging purposes or use it to create another `NSConnection` object, but shouldn't use it to send or receive messages explicitly. Don't set the delegate of the receive port; it already has a delegate established by the `NSConnection` object.

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

See Also
- `sendPort` (page 345)
- `initWithReceivePort:sendPort:` (page 337)

Declared In
`NSConnection.h`

registerName:

Registers the specified service using with the default system port name server.

```
- (BOOL)registerName:(NSString *)name
```

Parameters

name

The name under which to register the receiver.

Return Value
`YES` if the operation was successful, otherwise `NO` (for example, if another `NSConnection` object on the same host is already registered under *name*).

Discussion
This method connects the receive port of the receiving `NSConnection` object with the specified service name. It registers the name using the port name server returned by the `systemDefaultPortNameServer` (page 1270) method of `NSPortNameServer`. If the operation is successful,

other NSConnection objects can contact the receiver using the connectionWithRegisteredName:host: (page 330) and rootProxyForConnectionWithRegisteredName:host: (page 332) class methods.

If the receiver was already registered under a name and this method returns NO, the old name remains in effect. If this method is successful, it also unregisters the old name.

To unregister an NSConnection object, simply invoke registerName: and supply nil as the connection name.

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

See Also
- setRootObject: (page 347)
- registerName:withNameServer: (page 341)
+ systemDefaultPortNameServer (page 1270) (NSPortNameServer)

Declared In
NSConnection.h

registerName:withNameServer:

Registers a service with the specified port name server.

```
- (BOOL)registerName:(NSString *)name withNameServer:(NSPortNameServer *)server
```

Parameters

name
The name under which to register the receiver.

server
The name server.

Return Value
YES if the operation was successful, otherwise NO (for example, if another NSConnection object on the same host is already registered under *name*).

Discussion
This method connects the receive port of the receiving NSConnection object with the specified service name. If the operation is successful, other NSConnection objects can contact the receiver using the connectionWithRegisteredName:host: (page 330) and rootProxyForConnectionWithRegisteredName:host: (page 332) class methods.

If the receiver was already registered under a name and this method returns NO, the old name remains in effect. If this method is successful, it also unregisters the old name.

To unregister an NSConnection object, simply invoke registerName: and supply nil as the connection name.

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

Declared In
NSConnection.h

remoteObjects

Returns all the local proxies for remote objects that have been received over the connection but not deallocated yet.

```
- (NSArray *)remoteObjects
```

Return Value
An array containing all the local proxies for remote objects that have been received over the connection but not deallocated yet.

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

See Also
`- localObjects` (page 339)

Declared In
`NSConnection.h`

removeRequestMode:

Removes *mode* from the set of run-loop input modes the receiver uses for connection requests.

```
- (void)removeRequestMode:(NSString *)mode
```

Parameters
mode
The mode to remove from the set of run-loop input modes the receiver uses for connection requests.

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

See Also
`- requestModes` (page 343)
`removePort:forMode:` (page 1337) (NSRunLoop)

Declared In
`NSConnection.h`

removeRunLoop:

Removes a given `NSRunLoop` object from the list of run loops the receiver monitors and from which it responds to requests.

```
- (void)removeRunLoop:(NSRunLoop *)runloop
```

Parameters
runloop
The run loop to remove from the receiver.

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

See Also
- `addRunLoop:` (page 335)

Declared In
`NSConnection.h`

replyTimeout

Returns the timeout interval for replies to outgoing remote messages.

```
- (NSTimeInterval)replyTimeout
```

Return Value
The timeout interval for replies to outgoing remote messages.

Discussion
If a non-oneway remote message is sent and no reply is received by the timeout, an `NSPortTimeoutException` is raised.

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

See Also
- `requestTimeout` (page 344)
- `setReplyTimeout:` (page 347)

Declared In
`NSConnection.h`

requestModes

Returns the set of request modes the receiver's receive port is registered for with its `NSRunLoop` object.

```
- (NSArray *)requestModes
```

Return Value
An array of `NSString` objects that represents the set of request modes the receiver's receive port is registered for with its `NSRunLoop` object.

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

See Also
- `addRequestMode:` (page 335)
- `addPort:forMode:` (page 1333) (NSRunLoop)
- `removeRequestMode:` (page 342)

Declared In
`NSConnection.h`

requestTimeout

Returns the timeout interval for outgoing remote messages.

```
- (NSTimeInterval)requestTimeout
```

Return Value
The timeout interval for outgoing remote messages.

Discussion
If a remote message can't be sent before the timeout, an `NSPortTimeoutException` is raised.

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

See Also
- `replyTimeout` (page 343)
- `setRequestTimeout:` (page 347)

Declared In
`NSConnection.h`

rootObject

Returns the object that the receiver (or its parent) makes available to other applications or threads.

```
- (id)rootObject
```

Return Value
The object that the receiver (or its parent) makes available to other applications or threads, or `nil` if there is no root object.

Discussion
To get a proxy to this object in another application or thread, invoke the `rootProxyForConnectionWithRegisteredName:host:` (page 332) class method with the appropriate arguments.

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

See Also
- `rootProxy` (page 344)
- `setRootObject:` (page 347)

Declared In
`NSConnection.h`

rootProxy

Returns the proxy for the root object of the receiver's peer in another application or thread.

```
- (NSDistantObject *)rootProxy
```

Return Value
The proxy for the root object of the receiver's peer in another application or thread.

Discussion
The proxy returned can change between invocations if the peer `NSConnection` object's root object is changed.

> **Note:** If the `NSConnection` object uses separate send and receive ports and has no peer, when you invoke `rootProxy` it will block for the duration of the reply timeout interval, waiting for a reply.

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

See Also
`- rootObject` (page 344)

Related Sample Code
SimpleThreads
TrivialThreads

Declared In
`NSConnection.h`

runInNewThread

Creates and starts a new `NSThread` object and then runs the receiving connection in the new thread.

```
- (void)runInNewThread
```

Discussion
If the newly created thread is the first to be detached from the current thread, this method posts an `NSWillBecomeMultiThreadedNotification` with `nil` to the default notification center.

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

Declared In
`NSConnection.h`

sendPort

Returns the `NSPort` object that the receiver sends outgoing network messages through.

```
- (NSPort *)sendPort
```

Return Value
The `NSPort` object that the receiver sends outgoing network messages through.

Discussion
You can inspect this object for debugging purposes or use it to create another `NSConnection` object, but shouldn't use it to send or receive messages explicitly. Don't set the delegate of the send port; it already has a delegate established by the `NSConnection` object.

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

See Also
- `receivePort` (page 340)
- `initWithReceivePort:sendPort:` (page 337)

Declared In
`NSConnection.h`

setDelegate:

Sets the receiver's delegate.

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

Parameters

anObject
The receiver's delegate.

Discussion
A connection's delegate can process incoming messages itself instead of letting `NSConnection` object handle them. The delegate can also authenticate messages and accept, deny, or modify new connections.

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

Declared In
`NSConnection.h`

setIndependentConversationQueueing:

Sets a Boolean value that specifies whether the receiver handles remote messages atomically.

```
- (void)setIndependentConversationQueueing:(BOOL)flag
```

Parameters

flag
`YES` if the receiver handles remote messages atomically, otherwise `NO`.

Discussion
The default is `NO`. An `NSConnection` object normally forwards remote message to the intended recipients as they come in. See Configuring an NSConnection for more information.

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

See Also
- `independentConversationQueueing` (page 336)

Declared In
`NSConnection.h`

setReplyTimeout:

Sets the timeout interval for replies to outgoing remote messages

```
- (void)setReplyTimeout:(NSTimeInterval)seconds
```

Parameters

seconds

The timeout interval for replies to outgoing remote messages.

Discussion

If a non-oneway remote message is sent and no reply is received by the timeout, an `NSPortTimeoutException` is raised. The default timeout is the maximum possible value.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `setRequestTimeout:` (page 347)
- `replyTimeout` (page 343)

Declared In

`NSConnection.h`

setRequestTimeout:

Sets the timeout interval for outgoing remote messages.

```
- (void)setRequestTimeout:(NSTimeInterval)seconds
```

Parameters

seconds

The timeout interval for outgoing remote messages.

Discussion

If a remote message can't be sent before the timeout, an `NSPortTimeoutException` is raised. The default timeout is the maximum possible value.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `setReplyTimeout:` (page 347)
- `requestTimeout` (page 344)

Declared In

`NSConnection.h`

setRootObject:

Sets the object that the receiver makes available to other applications or threads.

```
- (void)setRootObject:(id)anObject
```

Parameters

anObject

The root object for the receiver.

Discussion

This only affects new connection requests and `rootProxy` (page 344) messages to established `NSConnection` objects; applications that have proxies to the old root object can still send messages through it.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- rootObject` (page 344)

Related Sample Code

SimpleThreads

Declared In

`NSConnection.h`

statistics

Returns an `NSDictionary` object containing various statistics for the receiver.

```
- (NSDictionary *)statistics
```

Return Value

An `NSDictionary` object containing various statistics for the receiver, such as the number of vended objects, the number of requests and replies, and so on.

Discussion

The statistics dictionary should be used only for debugging purposes.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSConnection.h`

Delegate Methods

authenticateComponents:withData:

Returns a Boolean value that indicates whether given authentication data is valid for a given set of components.

```
- (BOOL)authenticateComponents:(NSArray *)components withData:(NSData
    *)authenticationData
```

Parameters

components

An array that contains NSData and NSPort objects belonging to an NSPortMessage object. See the NSPortMessage class specification for more information.

authenticationData

Authentication data created by the delegate of the peer NSConnection object with authenticationDataForComponents: (page 349).

Return Value

YES if the *authenticationData* provided is valid for *components*, otherwise NO.

Discussion

Use this message for validation of incoming messages. An NSConnection object raises an NSFailedAuthenticationException on receipt of a remote message the delegate doesn't authenticate.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSConnection.h

authenticationDataForComponents:

Returns an NSData object to be used as an authentication stamp for an outgoing message.

```
- (NSData *)authenticationDataForComponents:(NSArray *)components
```

Parameters

components

An array containing the elements of a network message, in the form of NSPort and NSData objects.

Return Value

An NSData object to be used as an authentication stamp for an outgoing message.

Discussion

The delegate should use only the NSData elements to create the authentication stamp. See the NSPortMessage class specification for more information on the components.

If authenticationDataForComponents: (page 349) returns nil, an NSGenericException will be raised. If the delegate determines that the message shouldn't be authenticated, it should return an empty NSData object. The delegate on the other side of the connection must then be prepared to accept an empty NSData object as the second parameter to authenticateComponents:withData: (page 348) and to handle the situation appropriately.

The *components* parameter will be validated on receipt by the delegate of the peer NSConnection object with authenticateComponents:withData: (page 348).

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSConnection.h

connection:handleRequest:

This method should be implemented by `NSConnection` object delegates that want to intercept distant object requests.

```
- (BOOL)connection:(NSConnection *)conn handleRequest:(NSDistantObjectRequest
    *)doReq
```

Parameters

conn

The connection object for which the receiver is the delegate.

doReq

The distant object request.

Return Value

`YES` if the request was handled by the delegate, `NO` if the request should proceed as if the delegate did not intercept it.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSConnection.h`

connection:shouldMakeNewConnection:

Returns a Boolean value that indicates whether the parent connection should allow a given new connection to be created.

```
- (BOOL)connection:(NSConnection *)parentConnection
    shouldMakeNewConnection:(NSConnection *)newConnnection
```

Parameters

parentConnection

The connection object for which the receiver is the delegate.

newConnnection

The new connection.

Return Value

`YES` if *parentConnection* should allow *newConnnection* to be created and set up, `NO` if *parentConnection* should refuse and immediately release *newConnection*.

Discussion

Use this method to limit the amount of `NSConnection` objects created in your application or to change the parameters of child `NSConnection` objects.

Use `NSConnectionDidInitializeNotification` (page 353) instead of this delegate method if possible.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSConnection.h`

createConversationForConnection:

Returns an arbitrary object identifying a new conversation being created for the connection in the current thread.

```
- (id)createConversationForConnection:(NSConnection *)conn
```

Parameters

conn

The connection object for which the receiver is the delegate.

Return Value

An arbitrary object identifying a new conversation being created for the connection in the current thread.

Discussion

New conversations are created only if `independentConversationQueueing` (page 336) is `YES` for *conn*. If you do not implement this method, `NSConnection` object creates an instance of `NSObject`.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ currentConversation` (page 331)

`conversation` (page 536) (NSDistantObjectRequest)

Declared In

`NSConnection.h`

makeNewConnection:sender:

Returns a Boolean value that indicates whether the parent should allow a given new connection to be created and configured.

```
- (BOOL)makeNewConnection:(NSConnection *)newConnection sender:(NSConnection
    *)parentConnection
```

Parameters

newConnection

The new connection.

parentConnection

The parent connection.

Return Value

`YES` if *parentConnection* should allow *newConnnection* to be created and configured, `NO` if *parentConnection* should refuse and immediately release *newConnection*.

Discussion

Use this method to limit the amount of `NSConnection` objects created in your application or to change the parameters of child `NSConnection` objects.

Use `NSConnectionDidInitializeNotification` (page 353) instead of this delegate method if possible.

Availability

Available in Mac OS X v10.0 and later.

Declared In
NSConnection.h

Constants

NSConnection run loop mode

NSConnection defines the following run loop mode—see NSRunLoop for more details.

```
extern NSString *NSConnectionReplyMode;
```

Constants
NSConnectionReplyMode
The mode to indicate an NSConnection object waiting for replies.

You should rarely need to use this mode.

Declared in NSConnection.h.

Available in Mac OS X v10.0 and later.

Declared In
Foundation/NSConnection.h

Connection Exception Names

The name of an exception raised in case of authentication failure.

```
extern NSString *NSFailedAuthenticationException;
```

Constants
NSFailedAuthenticationException
Raised by NSConnection on receipt of a remote message the delegate doesn't authenticate.

Available in Mac OS X v10.0 and later.

Declared in NSConnection.h.

Declared In
Foundation/NSConnection.h

Notifications

NSConnectionDidDieNotification

Posted when an NSConnection object is deallocated or when it's notified that its NSPort object has become invalid. The notification object is the NSConnection object. This notification does not contain a *userInfo* dictionary.

An NSConnection object attached to a remote NSSocketPort object cannot detect when the remote port becomes invalid, even if the remote port is on the same machine. Therefore, it cannot post this notification when the connection is lost. Instead, you must detect the timeout error when the next message is sent.

The `NSConnection` object posting this notification is no longer useful, so all receivers should unregister themselves for any notifications involving the `NSConnection` object.

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

See Also
`NSPortDidBecomeInvalidNotification` (NSPort notification)

Declared In
`NSConnection.h`

NSConnectionDidInitializeNotification

Posted when an `NSConnection` object is initialized using `initWithReceivePort:sendPort:` (page 337) (the designated initializer for `NSConnection`). The notification object is the `NSConnection` object. This notification does not contain a *`userInfo`* dictionary.

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

See Also
`- initWithReceivePort:sendPort:` (page 337)

Declared In
`NSConnection.h`

NSCountCommand Class Reference

Inherits from	NSScriptCommand : NSObject
Conforms to	NSCoding (NSScriptCommand) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptStandardSuiteCommands.h
Companion guide	Cocoa Scripting Guide

Overview

An instance of `NSCountCommand` counts the number of objects of a specified class in the specified object container (such as the number of words in a paragraph or document) and returns the result.

`NSCountCommand` is part of Cocoa's built-in scripting support. It works automatically to support the `count` command through key-value coding. Most applications don't need to subclass `NSCountCommand` or call its methods.

NSCountedSet Class Reference

Inherits from	NSMutableSet : NSSet : NSObject
Conforms to	NSCoding (NSSet) NSCopying (NSSet) NSMutableCopying (NSSet) NSFastEnumeration (NSSet) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSSet.h
Companion guide	Collections Programming Topics for Cocoa
Related sample code	Dicey

Overview

The `NSCountedSet` class declares the programmatic interface to an object that manages a mutable set of objects. `NSCountedSet` provides support for the mathematical concept of a counted set. A counted set, both in its mathematical sense and in the implementation of `NSCountedSet`, is an unordered collection of elements, just as in a regular set, but the elements of the set aren't necessarily distinct. A counted set is also known as a bag.

Each distinct object inserted into an `NSCountedSet` object has a counter associated with it. `NSCountedSet`keeps track of the number of times objects are inserted and requires that objects be removed the same number of times. Thus, there is only one instance of an object in an `NSSet` object even if the object has been added to the set multiple times. The `count` (page 1451) method defined by the superclass `NSSet` has special significance; it returns the number of distinct objects, not the total number of times objects are represented in the set. The `NSSet` and `NSMutableSet` classes are provided for static and dynamic sets (respectively) whose elements are distinct.

You add objects to or remove objects from a counted set using the `addObject:` (page 358) and `removeObject:` (page 361) methods. You can traverse elements of an `NSCountedSet` object using the enumerator returned by `objectEnumerator` (page 361). The `countForObject:` (page 359) method returns the number of times a given object has been added to this set.

While `NSCountedSet` and `CFBag` are not toll-free bridged, they provide similar functionality. For more information on `CFBag`, consult the *CFBag Reference*.

Tasks

Initializing a Counted Set

- `initWithArray:` (page 359)
  Returns a counted set object initialized with the contents of a given array.
- `initWithSet:` (page 360)
  Returns a counted set object initialized with the contents of a given set.
- `initWithCapacity:` (page 360)
  Returns a counted set object initialized with enough memory to hold a given number of objects.

Adding and Removing Entries

- `addObject:` (page 358)
  Adds a given object to the receiver.
- `removeObject:` (page 361)
  Removes a given object from the receiver.

Examining a Counted Set

- `countForObject:` (page 359)
  Returns the count associated with a given object in the receiver.
- `objectEnumerator` (page 361)
  Returns an enumerator object that lets you access each object in the set once, independent of its count.

Instance Methods

addObject:

Adds a given object to the receiver.

```
- (void)addObject:(id)anObject
```

Parameters

anObject
  The object to add to the receiver.

Discussion

If *anObject* is already a member, `addObject:` increments the count associated with the object. If *anObject* is not already a member, it is sent a `retain` (page 2108) message.

Availability

Available in Mac OS X v10.0 and later.

Declared In
NSSet.h

countForObject:

Returns the count associated with a given object in the receiver.

```
- (NSUInteger)countForObject:(id)anObject
```

Parameters

anObject
The object for which to return the count.

Return Value
The count associated with *anObject* in the receiver, which can be thought of as the number of occurrences of *anObject* present in the receiver.

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

See Also
- count (page 1451) (NSSet)

Related Sample Code
Dicey

Declared In
NSSet.h

initWithArray:

Returns a counted set object initialized with the contents of a given array.

```
- (id)initWithArray:(NSArray *)anArray
```

Parameters

anArray
An array of objects to add to the new set.

Return Value
An initialized counted set object with the contents of *anArray*. The returned object might be different than the original receiver.

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

See Also
initWithArray: (page 1452) (NSSet)
setWithArray: (page 1445) (NSSet)

Declared In
NSSet.h

initWithCapacity:

Returns a counted set object initialized with enough memory to hold a given number of objects.

```
- (id)initWithCapacity:(NSUInteger)numItems
```

Parameters

numItems

The initial capacity of the new counted set.

Return Value

A counted set object initialized with enough memory to hold *numItems* objects

Discussion

The method is the designated initializer for `NSCountedSet`.

Note that the capacity is simply a hint to help initial memory allocation—the initial count of the object is `0`, and the set still grows and shrinks as you add and remove objects. The hint is typically useful if the set will become large.

Availability

Available in Mac OS X v10.0 and later.

See Also

`initWithCapacity:` (page 973) (`NSMutableSet`)

`setWithCapacity:` (page 971) (`NSMutableSet`)

Declared In

`NSSet.h`

initWithSet:

Returns a counted set object initialized with the contents of a given set.

```
- (id)initWithSet:(NSSet *)aSet
```

Parameters

aSet

An set of objects to add to the new set.

Return Value

An initialized counted set object with the contents of *aSet*. The returned object might be different than the original receiver.

Availability

Available in Mac OS X v10.0 and later.

See Also

`initWithSet:` (page 1454) (NSSet)

`setWithSet:` (page 1448) (NSSet)

Declared In

`NSSet.h`

objectEnumerator

Returns an enumerator object that lets you access each object in the set once, independent of its count.

```
- (NSEnumerator *)objectEnumerator
```

Return Value
An enumerator object that lets you access each object in the set once, independent of its count.

Discussion
If you add a given object to the counted set multiple times, an enumeration of the set will produce that object only once.

You shouldn't modify the set during enumeration. If you intend to modify the set, use the `allObjects` (page 1449) method to create a "snapshot," then enumerate the snapshot and modify the original set.

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

See Also
`nextObject` (page 558) (NSEnumerator)

Declared In
`NSSet.h`

removeObject:

Removes a given object from the receiver.

```
- (void)removeObject:(id)anObject
```

Parameters
anObject
The object to remove from the receiver.

Discussion
If *anObject* is present in the set, decrements the count associated with it. If the count is decremented to `0`, *anObject* is removed from the set and sent a `release` (page 2106) message. `removeObject:` does nothing if *anObject* is not present in the receiver.

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

See Also
`- countForObject:` (page 359)

Declared In
`NSSet.h`

NSCreateCommand Class Reference

Inherits from	NSScriptCommand : NSObject
Conforms to	NSCoding (NSScriptCommand) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptStandardSuiteCommands.h
Companion guide	Cocoa Scripting Guide

Overview

An instance of `NSCreateCommand` creates the specified scriptable object (such as a document), optionally supplying the new object with the specified attributes. This command corresponds to AppleScript's `make` command.

`NSCreateCommand` is part of Cocoa's built-in scripting support. Most applications don't need to subclass `NSCreateCommand` or invoke its methods.

When an instance of `NSCreateCommand` is executed, it creates a new object using `[[theClassToBeCreated allocWithZone:NULL] init]` (where `theClassToBeCreated` is the class of the object to be created), unless the command has a `with data` argument. In the latter case, the new object is created by invoking `[[NSScriptCoercionHandler sharedCoercionHandler] coerceValue:theDataAsAnObject toClass:theClassToBeCreated]`. Any properties specified by a `with properties` argument are then set in the new object using `-setScriptingProperties:`.

If an `NSCreateCommand` object with no argument corresponding to the `at` parameter is executed (for example, `tell application "Mail" to make new mailbox with properties {name:"testFolder"}`), and the receiver of the command (not necessarily the application object) has a to-many relationship to objects of the class to be instantiated, and the class description for the receiving class returns `NO` when sent an `isLocationRequiredToCreateForKey:` message, the `NSCreateCommand` object creates a new object and sends the receiver an `insertValue:atIndex:inPropertyWithKey:` (page 2118) message to place the new object in the container. This is part of Cocoa's scripting support for inserting newly-created objects into containers without explicitly specifying a location.

Tasks

Getting Information About a Create Command

- `createClassDescription` (page 364)
  Returns the class description for the class that is to be created.
- `resolvedKeyDictionary` (page 364)
  Returns a dictionary that contains the properties that were specified in the `make` Apple event command that has been converted to this `NSCreateCommand` object.

Instance Methods

createClassDescription

Returns the class description for the class that is to be created.

```
- (NSScriptClassDescription *)createClassDescription
```

Return Value
The class description for the class that is to be created.

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

Declared In
`NSScriptStandardSuiteCommands.h`

resolvedKeyDictionary

Returns a dictionary that contains the properties that were specified in the `make` Apple event command that has been converted to this `NSCreateCommand` object.

```
- (NSDictionary *)resolvedKeyDictionary
```

Return Value
A dictionary that contains the properties that were specified in the `make` Apple event script command that has been converted to this `NSCreateCommand` object.

Discussion
The keys in the returned dictionary are the names of properties (attributes or relationships, in the script suite) that have been specified for the command, and the corresponding values in the dictionary are the values that those properties should take. The required and optional arguments for the `make` command are specified in the core suite definition, `NSCoreSuite.scriptSuite`.

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

Declared In
`NSScriptStandardSuiteCommands.h`

NSData Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSMutableCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSData.h Foundation/NSSerialization.h (Deprecated)
Companion guides	Binary Data Programming Guide for Cocoa Property List Programming Guide
Related sample code	CocoaHTTPServer CocoaSOAP iSpend Sketch-112 StickiesExample

Overview

`NSData` and its mutable subclass `NSMutableData` provide data objects, object-oriented wrappers for byte buffers. Data objects let simple allocated buffers (that is, data with no embedded pointers) take on the behavior of Foundation objects.

`NSData` creates static data objects, and `NSMutableData` creates dynamic data objects. `NSData` and `NSMutableData` are typically used for data storage and are also useful in Distributed Objects applications, where data contained in data objects can be copied or moved between applications.

Using 32-bit Cocoa, the size of the data is subject to a theoretical 2GB limit (in practice, because memory will be used by other objects this limit will be smaller); using 64-bit Cocoa, the size of the data is subject to a theoretical limit of about 8EB (in practice, the limit should not be a factor).

`NSData` is “toll-free bridged” with its Core Foundation counterpart, CFData. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSData *` parameter, you can pass a `CFDataRef`, and in a function where you see a `CFDataRef` parameter, you can pass an `NSData` instance (you cast one type to the other to suppress compiler warnings). This also applies to your concrete subclasses of `NSData`. See Interchangeable Data Types for more information on toll-free bridging.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 2034)
- `initWithCoder:` (page 2034)

NSCopying
- `copyWithZone:` (page 2042)

NSMutableCopying
- `mutableCopyWithZone:` (page 2094)

Tasks

Creating Data Objects

`+ data` (page 370)
Creates and returns an empty data object.

`+ dataWithBytes:length:` (page 370)
Creates and returns a data object containing a given number of bytes copied from a given buffer.

`+ dataWithBytesNoCopy:length:` (page 371)
Creates and returns a data object that holds *length* bytes from the buffer *bytes*.

`+ dataWithBytesNoCopy:length:freeWhenDone:` (page 371)
Creates and returns a data object that holds a given number of bytes from a given buffer.

`+ dataWithContentsOfFile:` (page 372)
Creates and returns a data object by reading every byte from the file specified by a given path.

`+ dataWithContentsOfFile:options:error:` (page 373)
Creates and returns a data object by reading every byte from the file specified by a given path.

`+ dataWithContentsOfMappedFile:` (page 373)
Creates and returns a data object from the mapped file specified by *path*.

`+ dataWithContentsOfURL:` (page 374)
Returns a data object containing the data from the location specified by a given URL.

`+ dataWithContentsOfURL:options:error:` (page 375)
Creates and returns a data object containing the data from the location specified by *aURL*.

`+ dataWithData:` (page 375)
Creates and returns a data object containing the contents of another data object.

`- initWithBytes:length:` (page 378)
Returns a data object initialized by adding to it a given number of bytes of data copied from a given buffer.

`- initWithBytesNoCopy:length:` (page 379)
Returns a data object initialized by adding to it a given number of bytes of data from a given buffer.

`- initWithBytesNoCopy:length:freeWhenDone:` (page 379)
Initializes a newly allocated data object by adding to it *length* bytes of data from the buffer *bytes*.

- `initWithContentsOfFile:` (page 380)
  Returns a data object initialized by reading into it the data from the file specified by a given path.
- `initWithContentsOfFile:options:error:` (page 381)
  Returns a data object initialized by reading into it the data from the file specified by a given path.
- `initWithContentsOfMappedFile:` (page 381)
  Returns a data object initialized by reading into it the mapped file specified by a given path.
- `initWithContentsOfURL:` (page 382)
  Initializes a newly allocated data object initialized with the data from the location specified by *aURL*.
- `initWithContentsOfURL:options:error:` (page 382)
  Returns a data object initialized with the data from the location specified by a given URL.
- `initWithData:` (page 383)
  Returns a data object initialized with the contents of another data object.

Accessing Data

- `bytes` (page 376)
  Returns a pointer to the receiver's contents.
- `description` (page 376)
  Returns an `NSString` object that contains a hexadecimal representation of the receiver's contents.
- `getBytes:` (page 377)
  Copies a data object's contents into a given buffer.
- `getBytes:length:` (page 377)
  Copies a number of bytes from the start of the receiver's data into a given buffer.
- `getBytes:range:` (page 378)
  Copies a range of bytes from the receiver's data into a given buffer.
- `subdataWithRange:` (page 384)
  Returns a data object containing a copy of the receiver's bytes that fall within the limits specified by a given range.

Testing Data

- `isEqualToData:` (page 383)
  Compares the receiving data object to *otherData*.
- `length` (page 383)
  Returns the number of bytes contained in the receiver.

Storing Data

- `writeToFile:atomically:` (page 384)
  Writes the bytes in the receiver to the file specified by a given path.
- `writeToFile:options:error:` (page 385)
  Writes the bytes in the receiver to the file specified by a given path.

- writeToURL:atomically: (page 385)
  Writes the bytes in the receiver to the location specified by *aURL*.
- writeToURL:options:error: (page 386)
  Writes the bytes in the receiver to the location specified by a given URL.

Class Methods

data

Creates and returns an empty data object.

```
+ (id)data
```

Return Value
An empty data object.

Discussion
This method is declared primarily for the use of mutable subclasses of `NSData`.

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

Related Sample Code
EnhancedDataBurn
QTKitMovieShuffler

Declared In
`NSData.h`

dataWithBytes:length:

Creates and returns a data object containing a given number of bytes copied from a given buffer.

```
+ (id)dataWithBytes:(const void *)bytes length:(NSUInteger)length
```

Parameters
bytes
  A buffer containing data for the new object.
length
  The number of bytes to copy from *bytes*. This value must not exceed the length of *bytes*.

Return Value
A data object containing *length* bytes copied from the buffer *bytes*. Returns `nil` if the data object could not be created.

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

See Also
+ dataWithBytesNoCopy:length: (page 371)

+ dataWithBytesNoCopy:length:freeWhenDone: (page 371)

Related Sample Code
CocoaHTTPServer
CocoaSOAP
EnhancedDataBurn
QTCoreVideo301
QTMetadataEditor

Declared In
NSData.h

dataWithBytesNoCopy:length:

Creates and returns a data object that holds *length* bytes from the buffer *bytes*.

```
+ (id)dataWithBytesNoCopy:(void *)bytes length:(NSUInteger)length
```

Parameters

bytes
A buffer containing data for the new object. *bytes* must point to a memory block allocated with malloc.

length
The number of bytes to hold from *bytes*. This value must not exceed the length of *bytes*.

Return Value
A data object that holds *length* bytes from the buffer *bytes*. Returns nil if the data object could not be created.

Discussion
The returned object takes ownership of the *bytes* pointer and frees it on deallocation. Therefore, *bytes* must point to a memory block allocated with malloc.

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

See Also
+ dataWithBytes:length: (page 370)
+ dataWithBytesNoCopy:length:freeWhenDone: (page 371)

Declared In
NSData.h

dataWithBytesNoCopy:length:freeWhenDone:

Creates and returns a data object that holds a given number of bytes from a given buffer.

```
+ (id)dataWithBytesNoCopy:(void *)bytes length:(NSUInteger)length
    freeWhenDone:(BOOL)freeWhenDone
```

Parameters

bytes

A buffer containing data for the new object. If *freeWhenDone* is YES, *bytes* must point to a memory block allocated with malloc.

length

The number of bytes to hold from *bytes*. This value must not exceed the length of *bytes*.

freeWhenDone

If YES, the returned object takes ownership of the *bytes* pointer and frees it on deallocation.

Return Value

A data object that holds *length* bytes from the buffer *bytes*. Returns nil if the data object could not be created.

Availability

Available in Mac OS X v10.2 and later.

See Also

+ dataWithBytes:length: (page 370)

+ dataWithBytesNoCopy:length: (page 371)

Related Sample Code

CocoaSpeechSynthesisExample

Declared In

NSData.h

dataWithContentsOfFile:

Creates and returns a data object by reading every byte from the file specified by a given path.

```
+ (id)dataWithContentsOfFile:(NSString *)path
```

Parameters

path

The absolute path of the file from which to read data.

Return Value

A data object by reading every byte from the file specified by *path*. Returns nil if the data object could not be created.

Discussion

This method is equivalent to dataWithContentsOfFile:options:error: (page 373) with no options. If you need to know what was the reason for failure, use dataWithContentsOfFile:options:error: (page 373).

A sample using this method can be found in Working With Binary Data.

Availability

Available in Mac OS X v10.0 and later.

See Also

+ dataWithContentsOfFile:options:error: (page 373)

+ dataWithContentsOfMappedFile: (page 373)

Related Sample Code
CarbonCocoaCoreImageTab
iSpend
LiveVideoMixer2
Reducer
WhackedTV

Declared In
NSData.h

dataWithContentsOfFile:options:error:

Creates and returns a data object by reading every byte from the file specified by a given path.

```
+ (id)dataWithContentsOfFile:(NSString *)path options:(NSUInteger)mask error:(NSError **)errorPtr
```

Parameters

path
The absolute path of the file from which to read data.

mask
A mask that specifies options for reading the data. Constant components are described in "Options for NSData Reading Methods" (page 387).

errorPtr
If an error occurs, upon return contains an NSError object that describes the problem.

Return Value
A data object by reading every byte from the file specified by *path*. Returns nil if the data object could not be created.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSData.h

dataWithContentsOfMappedFile:

Creates and returns a data object from the mapped file specified by *path*.

```
+ (id)dataWithContentsOfMappedFile:(NSString *)path
```

Parameters

path
The absolute path of the file from which to read data.

Return Value
A data object from the mapped file specified by *path*. Returns nil if the data object could not be created.

Discussion
Because of file mapping restrictions, this method should only be used if the file is guaranteed to exist for the duration of the data object's existence. It is generally safer to use the `dataWithContentsOfFile:` (page 372) method.

This methods assumes mapped files are available from the underlying operating system. A mapped file uses virtual memory techniques to avoid copying pages of the file into memory until they are actually needed.

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

See Also
`+ dataWithContentsOfFile:` (page 372)

Related Sample Code
Quartz EB

Declared In
`NSData.h`

dataWithContentsOfURL:

Returns a data object containing the data from the location specified by a given URL.

```
+ (id)dataWithContentsOfURL:(NSURL *)aURL
```

Parameters

aURL

The URL from which to read data.

Return Value
A data object containing the data from the location specified by *aURL*. Returns `nil` if the data object could not be created.

Discussion
If you need to know what was the reason for failure, use `dataWithContentsOfURL:options:error:` (page 375).

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

See Also
`+ dataWithContentsOfURL:options:error:` (page 375)
`- initWithContentsOfURL:` (page 382)

Related Sample Code
CocoaSpeechSynthesisExample
Core Data HTML Store
CustomAtomicStoreSubclass
QTKitFrameStepper
WebKitCIPlugIn

Declared In
`NSData.h`

dataWithContentsOfURL:options:error:

Creates and returns a data object containing the data from the location specified by *aURL*.

```
+ (id)dataWithContentsOfURL:(NSURL *)aURL options:(NSUInteger)mask error:(NSError
    **)errorPtr
```

Parameters

aURL

The URL from which to read data.

mask

A mask that specifies options for reading the data. Constant components are described in "Options for NSData Reading Methods" (page 387).

errorPtr

If there is an error reading in the data, upon return contains an `NSError` object that describes the problem.

Availability

Available in Mac OS X v10.4 and later.

See Also

- initWithContentsOfURL: (page 382)

Declared In

NSData.h

dataWithData:

Creates and returns a data object containing the contents of another data object.

```
+ (id)dataWithData:(NSData *)aData
```

Parameters

aData

A data object.

Return Value

A data object containing the contents of *aData*. Returns `nil` if the data object could not be created.

Availability

Available in Mac OS X v10.0 and later.

See Also

- initWithData: (page 383)

Related Sample Code

Core Data HTML Store

Declared In

NSData.h

Instance Methods

bytes

Returns a pointer to the receiver's contents.

```
- (const void *)bytes
```

Return Value
A read-only pointer to the receiver's contents.

Discussion
If the `length` (page 383) of the receiver is 0, this method returns `nil`.

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

See Also
- `description` (page 376)
- `getBytes:` (page 377)
- `getBytes:length:` (page 377)
- `getBytes:range:` (page 378)

Related Sample Code
AudioBurn
CocoaHTTPServer
CocoaSOAP
EnhancedDataBurn
QTSSConnectionMonitor

Declared In
`NSData.h`

description

Returns an `NSString` object that contains a hexadecimal representation of the receiver's contents.

```
- (NSString *)description
```

Return Value
An `NSString` object that contains a hexadecimal representation of the receiver's contents in `NSData` property list format.

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

See Also
- `bytes` (page 376)
- `getBytes:` (page 377)
- `getBytes:length:` (page 377)

- getBytes:range: (page 378)

Related Sample Code
Fiendishthngs

Declared In
NSData.h

getBytes:

Copies a data object's contents into a given buffer.

```
- (void)getBytes:(void *)buffer
```

Parameters

buffer

A buffer into which to copy the receiver's data. The buffer must be at least length (page 383) bytes.

Discussion
You can see a sample using this method in Working With Binary Data.

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

See Also
- bytes (page 376)
- description (page 376)
- getBytes:length: (page 377)
- getBytes:range: (page 378)

Related Sample Code
JavaSplashScreen
OpenGLCaptureToMovie
QTCoreVideo301
QTMetadataEditor
Quartz Composer QCTV

Declared In
NSData.h

getBytes:length:

Copies a number of bytes from the start of the receiver's data into a given buffer.

```
- (void)getBytes:(void *)buffer length:(NSUInteger)length
```

Parameters

buffer

A buffer into which to copy data.

length

The number of bytes from the start of the receiver's data to copy to *buffer*.

Discussion
The number of bytes copied is the smaller of the *length* parameter and the `length` of the data encapsulated in the object.

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

See Also
- `bytes` (page 376)
- `description` (page 376)
- `getBytes:` (page 377)
- `getBytes:range:` (page 378)

Declared In
`NSData.h`

getBytes:range:

Copies a range of bytes from the receiver's data into a given buffer.

```
- (void)getBytes:(void *)buffer range:(NSRange)range
```

Parameters

buffer
A buffer into which to copy data.

range
The range of bytes in the receiver's data to copy to *buffer*. The range must lie within the range of bytes of the receiver's data.

Discussion
If *range* isn't within the receiver's range of bytes, an `NSRangeException` is raised.

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

See Also
- `bytes` (page 376)
- `description` (page 376)
- `getBytes:` (page 377)
- `getBytes:length:` (page 377)

Declared In
`NSData.h`

initWithBytes:length:

Returns a data object initialized by adding to it a given number of bytes of data copied from a given buffer.

```
- (id)initWithBytes:(const void *)bytes length:(NSUInteger)length
```

Discussion
A data object initialized by adding to it *length* bytes of data copied from the buffer *bytes*. The returned object might be different than the original receiver.

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

See Also
+ dataWithBytes:length: (page 370)
- initWithBytesNoCopy:length: (page 379)
- initWithBytesNoCopy:length:freeWhenDone: (page 379)

Declared In
NSData.h

initWithBytesNoCopy:length:

Returns a data object initialized by adding to it a given number of bytes of data from a given buffer.

```
- (id)initWithBytesNoCopy:(void *)bytes length:(NSUInteger)length
```

Parameters

bytes

A buffer containing data for the new object. *bytes* must point to a memory block allocated with malloc.

length

The number of bytes to hold from *bytes*. This value must not exceed the length of *bytes*.

Return Value
A data object initialized by adding to it *length* bytes of data from the buffer *bytes*. The returned object might be different than the original receiver.

Discussion
The returned object takes ownership of the *bytes* pointer and frees it on deallocation. Therefore, *bytes* must point to a memory block allocated with malloc.

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

See Also
+ dataWithBytes:length: (page 370)
- initWithBytes:length: (page 378)
- initWithBytesNoCopy:length:freeWhenDone: (page 379)

Declared In
NSData.h

initWithBytesNoCopy:length:freeWhenDone:

Initializes a newly allocated data object by adding to it *length* bytes of data from the buffer *bytes*.

```
- (id)initWithBytesNoCopy:(void *)bytes length:(NSUInteger)length
    freeWhenDone:(BOOL)flag
```

Parameters

bytes

A buffer containing data for the new object. If *flag* is YES, *bytes* must point to a memory block allocated with malloc.

length

The number of bytes to hold from *bytes*. This value must not exceed the length of *bytes*.

flag

If YES, the returned object takes ownership of the *bytes* pointer and frees it on deallocation.

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

See Also
+ dataWithBytesNoCopy:length:freeWhenDone: (page 371)
- initWithBytes:length: (page 378)
- initWithBytesNoCopy:length: (page 379)

Declared In
NSData.h

initWithContentsOfFile:

Returns a data object initialized by reading into it the data from the file specified by a given path.

```
- (id)initWithContentsOfFile:(NSString *)path
```

Parameters

path

The absolute path of the file from which to read data.

Return Value
A data object initialized by reading into it the data from the file specified by *path*. The returned object might be different than the original receiver.

Discussion
This method is equivalent to initWithContentsOfFile:options:error: (page 381) with no options.

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

See Also
+ dataWithContentsOfFile: (page 372)
- initWithContentsOfMappedFile: (page 381)

Declared In
NSData.h

initWithContentsOfFile:options:error:

Returns a data object initialized by reading into it the data from the file specified by a given path.

```
- (id)initWithContentsOfFile:(NSString *)path options:(NSUInteger)mask error:(NSError **)errorPtr
```

Parameters

path

The absolute path of the file from which to read data.

mask

A mask that specifies options for reading the data. Constant components are described in "Options for NSData Reading Methods" (page 387).

errorPtr

If an error occurs, upon return contains an `NSError` object that describes the problem.

Return Value

A data object initialized by reading into it the data from the file specified by *path*. The returned object might be different than the original receiver.

Availability

Available in Mac OS X v10.4 and later.

See Also

`+ dataWithContentsOfFile:options:error:` (page 373)

Declared In

`NSData.h`

initWithContentsOfMappedFile:

Returns a data object initialized by reading into it the mapped file specified by a given path.

```
- (id)initWithContentsOfMappedFile:(NSString *)path
```

Parameters

path

The absolute path of the file from which to read data.

Return Value

A data object initialized by reading into it the mapped file specified by *path*. The returned object might be different than the original receiver.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ dataWithContentsOfMappedFile:` (page 373)
`- initWithContentsOfFile:` (page 380)

Declared In

`NSData.h`

initWithContentsOfURL:

Initializes a newly allocated data object initialized with the data from the location specified by *aURL*.

```
- (id)initWithContentsOfURL:(NSURL *)aURL
```

Parameters

aURL

The URL from which to read data

Return Value

An `NSData` object initialized with the data from the location specified by *aURL*. The returned object might be different than the original receiver.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ dataWithContentsOfURL:` (page 374)

Declared In

`NSData.h`

initWithContentsOfURL:options:error:

Returns a data object initialized with the data from the location specified by a given URL.

```
- (id)initWithContentsOfURL:(NSURL *)aURL options:(NSUInteger)mask error:(NSError
    **)errorPtr
```

Parameters

aURL

The URL from which to read data.

mask

A mask that specifies options for reading the data. Constant components are described in "`Options for NSData Reading Methods`" (page 387).

errorPtr

If there is an error reading in the data, upon return contains an `NSError` object that describes the problem.

Return Value

A data object initialized with the data from the location specified by *aURL*. The returned object might be different than the original receiver.

Availability

Available in Mac OS X v10.4 and later.

See Also

`+ dataWithContentsOfURL:options:error:` (page 375)

Declared In

`NSData.h`

initWithData:

Returns a data object initialized with the contents of another data object.

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

Parameters

data

A data object.

Return Value

A data object initialized with the contents *data*. The returned object might be different than the original receiver.

Availability

Available in Mac OS X v10.0 and later.

See Also

+ dataWithData: (page 375)

Declared In

NSData.h

isEqualToData:

Compares the receiving data object to *otherData*.

```
- (BOOL)isEqualToData:(NSData *)otherData
```

Parameters

otherData

The data object with which to compare the receiver.

Return Value

YES if the contents of *otherData* are equal to the contents of the receiver, otherwise NO.

Discussion

Two data objects are equal if they hold the same number of bytes, and if the bytes at the same position in the objects are the same.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSData.h

length

Returns the number of bytes contained in the receiver.

```
- (NSUInteger)length
```

Return Value

The number of bytes contained in the receiver.

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

Related Sample Code
AudioBurn
CocoaHTTPServer
CocoaSOAP
QTMetadataEditor
WhackedTV

Declared In
`NSData.h`

subdataWithRange:

Returns a data object containing a copy of the receiver's bytes that fall within the limits specified by a given range.

```
- (NSData *)subdataWithRange:(NSRange)range
```

Parameters

range

The range in the receiver from which to copy bytes. The range must not exceed the bounds of the receiver.

Return Value
A data object containing a copy of the receiver's bytes that fall within the limits specified by *range*.

Discussion
If *range* isn't within the receiver's range of bytes, an `NSRangeException` is raised.

A sample using this method can be found in Working With Binary Data.

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

Declared In
`NSData.h`

writeToFile:atomically:

Writes the bytes in the receiver to the file specified by a given path.

```
- (BOOL)writeToFile:(NSString *)path atomically:(BOOL)flag
```

Parameters

path

The location to which to write the receiver's bytes. If *path* contains a tilde (~) character, you must expand it with `stringByExpandingTildeInPath` (page 1602) before invoking this method.

atomically

If `YES`, the data is written to a backup file, and then—assuming no errors occur—the backup file is renamed to the name specified by *path*; otherwise, the data is written directly to *path*.

Return Value
`YES` if the operation succeeds, otherwise `NO`.

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

See Also
`- writeToURL:atomically:` (page 385)

Related Sample Code
Aperture Edit Plugin - Borders & Titles
Quartz Composer WWDC 2005 TextEdit
Reducer
TextEditPlus
WhackedTV

Declared In
`NSData.h`

writeToFile:options:error:

Writes the bytes in the receiver to the file specified by a given path.

```
- (BOOL)writeToFile:(NSString *)path options:(NSUInteger)mask error:(NSError
    **)errorPtr
```

Parameters

path

The location to which to write the receiver's bytes.

mask

A mask that specifies options for writing the data. Constant components are described in "Options for NSData Writing Methods" (page 387).

errorPtr

If there is an error writing out the data, upon return contains an `NSError` object that describes the problem.

Return Value
`YES` if the operation succeeds, otherwise `NO`.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- writeToURL:options:error:` (page 386)

Declared In
`NSData.h`

writeToURL:atomically:

Writes the bytes in the receiver to the location specified by *aURL*.

```
- (BOOL)writeToURL:(NSURL *)aURL atomically:(BOOL)atomically
```

Parameters

aURL

The location to which to write the receiver's bytes. Only `file://` URLs are supported.

atomically

If `YES`, the data is written to a backup location, and then—assuming no errors occur—the backup location is renamed to the name specified by *aURL*; otherwise, the data is written directly to *aURL*. *atomically* is ignored if *aURL* is not of a type the supports atomic writes.

Return Value

`YES` if the operation succeeds, otherwise `NO`.

Discussion

Since at present only `file://` URLs are supported, there is no difference between this method and writeToFile:atomically: (page 384), except for the type of the first argument.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- writeToFile:atomically:` (page 384)

Related Sample Code

Core Data HTML Store

CoreRecipes

CustomAtomicStoreSubclass

Declared In

`NSData.h`

writeToURL:options:error:

Writes the bytes in the receiver to the location specified by a given URL.

```
- (BOOL)writeToURL:(NSURL *)aURL options:(NSUInteger)mask error:(NSError **)errorPtr
```

Parameters

aURL

The location to which to write the receiver's bytes.

mask

A mask that specifies options for writing the data. Constant components are described in "`Options for NSData Writing Methods`" (page 387).

errorPtr

If there is an error writing out the data, upon return contains an `NSError` object that describes the problem.

Return Value

`YES` if the operation succeeds, otherwise `NO`.

Discussion

Since at present only `file://` URLs are supported, there is no difference between this method and `writeToFile:options:error:` (page 385), except for the type of the first argument.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- writeToFile:options:error:` (page 385)

Declared In
`NSData.h`

Constants

Options for NSData Reading Methods

Options for methods used to read `NSData` objects.

```
enum {
    NSMappedRead = 1,
    NSUncachedRead = 2
};
```

Constants

`NSMappedRead`
A hint indicating the file should be mapped into virtual memory, if possible.
Available in Mac OS X v10.4 and later.
Declared in `NSData.h`.

`NSUncachedRead`
A hint indicating the file should not be stored in the file-system caches.
For data being read once and discarded, this option can improve performance.
Available in Mac OS X v10.4 and later.
Declared in `NSData.h`.

Declared In
`NSData.h`

Options for NSData Writing Methods

Options for methods used to write `NSData` objects.

```
enum {
    NSAtomicWrite = 1
};
```

Constants

`NSAtomicWrite`
A hint to use an auxiliary file when saving data and then exchange the files.
Available in Mac OS X v10.4 and later.
Declared in `NSData.h`.

Declared In
`NSData.h`

NSDate Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSDate.h Foundation/NSCalendarDate.h
Companion guides	Date and Time Programming Guide for Cocoa Property List Programming Guide
Related sample code	iSpend NewsReader Quartz Composer WWDC 2005 TextEdit Reminders TextEditPlus

Overview

`NSDate` objects represent a single point in time. `NSDate` is a class cluster; its single public superclass, `NSDate`, declares the programmatic interface for specific and relative time values. The objects you create using `NSDate` are referred to as date objects. They are immutable objects. Because of the nature of class clusters, objects returned by the `NSDate` class are instances not of that abstract class but of one of its private subclasses. Although a date object's class is private, its interface is public, as declared by the abstract superclass `NSDate`. Generally, you instantiate a suitable date object by invoking one of the `date...` class methods.

`NSDate` is an abstract class that provides behavior for creating dates, comparing dates, representing dates, computing intervals, and similar functionality. `NSDate` presents a programmatic interface through which suitable date objects are requested and returned. Date objects returned from `NSDate` are lightweight and immutable since they represent an invariant point in time. This class is designed to provide the foundation for arbitrary calendrical representations.

The sole primitive method of `NSDate`, `timeIntervalSinceReferenceDate` (page 408), provides the basis for all the other methods in the `NSDate` interface. This method returns a time value relative to an absolute reference date—the first instant of 1 January 2001, GMT.

NSDate provides several methods to interpret and to create string representations of dates (for example, dateWithNaturalLanguageString:locale: (page 394) and descriptionWithLocale: (page 402)). In general, on Mac OS X v10.4 and later you should use an instance of NSDateFormatter to parse and generate strings using the methods dateFromString: (page 431) and stringFromDate: (page 456)—see NSDateFormatter on Mac OS X 10.4 for more details.

NSDate models the change from the Julian to the Gregorian calendar in October 1582, and calendrical calculations performed in conjunction with NSCalendar take this transition into account. Note, however, that some locales adopted the Gregorian calendar at other times; for example, Great Britain didn't switch over until September 1752.

NSDate is "toll-free bridged" with its Cocoa Foundation counterpart, *CFDate Reference*. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an NSDate * parameter, you can pass a CFDateRef, and in a function where you see a CFDateRef parameter, you can pass an NSDate instance (you cast one type to the other to suppress compiler warnings). See Interchangeable Data Types for more information on toll-free bridging.

Subclassing Notes

The major reason for subclassing NSDate is to create a class with convenience methods for working with a particular calendrical system. But you could also require a custom NSDate class for other reasons, such as to get a date and time value that provides a finer temporal granularity.

Methods to Override

If you want to subclass NSDate to obtain behavior different than that provided by the private or public subclasses, you must do these things:

- Declare a suitable instance variable to hold the date and time value (relative to an absolute reference date).
- Override the timeIntervalSinceReferenceDate (page 408) instance method to provide the correct date and time value based on your instance variable.
- Override initWithTimeIntervalSinceReferenceDate: (page 406), the designated initializer method.

If you are creating a subclass that represents a calendrical system, you must also define methods that partition past and future periods into the units of this calendar.

Because the NSDate class adopts the NSCopying and NSCoding protocols, your subclass must also implement all of the methods in these protocols.

Special Considerations

Your subclass may use a different reference date than the absolute reference date used by NSDate (the first instance of 1 January 2001, GMT). If it does, it must still use the absolute reference date in its implementations of the methods timeIntervalSinceReferenceDate (page 408) and initWithTimeIntervalSinceReferenceDate: (page 406). That is, the reference date referred to in the titles of these methods is the absolute reference date. If you do not use the absolute reference date in these methods, comparisons between NSDate objects of your subclass and NSDate objects of a private subclass will not work.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 2034)
- `initWithCoder:` (page 2034)

NSCopying
- `copyWithZone:` (page 2042)

Tasks

Creating and Initializing Date Objects

`+ date` (page 393)
Creates and returns a new date set to the current date and time.

`+ dateWithNaturalLanguageString:` (page 394)
Creates and returns an `NSDate` object set to the date and time specified by a given string.

`+ dateWithNaturalLanguageString:locale:` (page 394)
Creates and returns an `NSDate` object set to the date and time specified by a given string.

`+ dateWithString:` (page 395)
Creates and returns an `NSDate` object with a date and time value specified by a given string in the international string representation format (`YYYY-MM-DD HH:MM:SS ±HHMM`).

`+ dateWithTimeIntervalSinceNow:` (page 396)
Creates and returns an `NSDate` object set to a given number of seconds from the current date and time.

`+ dateWithTimeIntervalSinceReferenceDate:` (page 397)
Creates and returns an `NSDate` object set to a given number of seconds from the first instant of 1 January 2001, GMT.

`+ dateWithTimeIntervalSince1970:` (page 396)
Creates and returns an `NSDate` object set to the given number of seconds from the first instant of 1 January 1970, GMT.

`- init` (page 404)
Returns an `NSDate` object initialized to the current date and time.

`- initWithString:` (page 404)
Returns an `NSDate` object initialized with a date and time value specified by a given string in the international string representation format.

`- initWithTimeIntervalSinceNow:` (page 405)
Returns an `NSDate` object initialized relative to the current date and time by a given number of seconds.

`- initWithTimeInterval:sinceDate:` (page 405)
Returns an `NSDate` object initialized relative to another given date by a given number of seconds.

- initWithTimeIntervalSinceReferenceDate: (page 406)
  Returns an NSDate object initialized relative the first instant of 1 January 2001, GMT by a given number of seconds.

Getting Temporal Boundaries

+ distantFuture (page 397)
  Creates and returns an NSDate object representing a date in the distant future.
+ distantPast (page 398)
  Creates and returns an NSDate object representing a date in the distant past.

Comparing Dates

- isEqualToDate: (page 406)
  Returns a Boolean value that indicates whether a given object is an NSDate object and exactly equal the receiver.
- earlierDate: (page 403)
  Returns the earlier of the receiver and another given date.
- laterDate: (page 407)
  Returns the later of the receiver and another given date.
- compare: (page 400)
  Returns an NSComparisonResult value that indicates the temporal ordering of the receiver and another given date.

Getting Time Intervals

- timeIntervalSinceDate: (page 408)
  Returns the interval between the receiver and another given date.
- timeIntervalSinceNow (page 408)
  Returns the interval between the receiver and the current date and time.
+ timeIntervalSinceReferenceDate (page 398)
  Returns the interval between the first instant of 1 January 2001, GMT and the current date and time.
- timeIntervalSinceReferenceDate (page 408)
  Returns the interval between the receiver and the first instant of 1 January 2001, GMT.
- timeIntervalSince1970 (page 407)
  Returns the interval between the receiver and the first instant of 1 January 1970, GMT.

Adding a Time Interval

- addTimeInterval: (page 399)
  Returns a new NSDate object that is set to a given number of seconds relative to the receiver.

Representing Dates as Strings

- `description` (page 401)
  Returns a string representation of the receiver.
- `descriptionWithCalendarFormat:timeZone:locale:` (page 401)
  Returns a string representation of the receiver, formatted as specified by given conversion specifiers.
- `descriptionWithLocale:` (page 402)
  Returns a string representation of the receiver using the given locale.

Converting to an NSCalendarDate Object

- `dateWithCalendarFormat:timeZone:` (page 400)
  Converts the receiver to an `NSCalendarDate` object with a given format string and time zone.

Class Methods

date

Creates and returns a new date set to the current date and time.

```
+ (id)date
```

Return Value
A new date object set to the current date and time.

Discussion
This method uses the default initializer method for the class, `init` (page 404).

The following code sample shows how to use `date` to get the current date:

```
NSDate *today = [NSDate date];
```

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

Related Sample Code
Clock Control
DatePicker
iSpend
Reminders
StickiesExample

Declared In
`NSDate.h`

dateWithNaturalLanguageString:

Creates and returns an `NSDate` object set to the date and time specified by a given string.

```
+ (id)dateWithNaturalLanguageString:(NSString *)string
```

Parameters

string

A string that contains a colloquial specification of a date, such as "last Tuesday at dinner," "3pm December 31, 2001," "12/31/01," or "31/12/01."

Return Value

A new `NSDate` object set to the current date and time specified by *string*.

Discussion

This method supports only a limited set of colloquial phrases, primarily in English. It may give unexpected results, and its use is strongly discouraged.

In parsing the string, this method uses the date and time preferences stored in the user's defaults database. (See `dateWithNaturalLanguageString:locale:` (page 394) for a list of the specific items used.)

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Core Data HTML Store

Reminders

Declared In

`NSCalendarDate.h`

dateWithNaturalLanguageString:locale:

Creates and returns an `NSDate` object set to the date and time specified by a given string.

```
+ (id)dateWithNaturalLanguageString:(NSString *)string locale:(id)localeDictionary
```

Parameters

string

A string that contains a colloquial specification of a date, such as "last Tuesday at dinner," "3pm December 31, 2001," "12/31/01," or "31/12/01."

localeDictionary

An `NSDictionary` object containing locale data. To use the user's preferences, you can use `[[NSUserDefaults standardUserDefaults] dictionaryRepresentation]`.

If you pass `nil` or an instance of `NSLocale`, `NSDate` uses the system default locale—this is not the same as the current user's locale.

Return Value

A new `NSDate` object set to the date and time specified by *string* as interpreted according to *localeDictionary*.

Discussion

This method supports only a limited set of colloquial phrases, primarily in English. It may give unexpected results, and its use is strongly discouraged.

The keys and values that represent the locale data from *localeDictionary* are used when parsing the string. In addition to the locale keys listed in the class description, these keys are used when parsing natural language strings:

```
NSDateTimeOrdering
NSEarlierTimeDesignations
NSHourNameDesignations
NSLaterTimeDesignations
NSNextDayDesignations
NSNextNextDayDesignations
NSPriorDayDesignations
NSThisDayDesignations
NSYearMonthWeekDesignations
```

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

See Also
+ dateWithNaturalLanguageString: (page 394)

Declared In
NSCalendarDate.h

dateWithString:

Creates and returns an NSDate object with a date and time value specified by a given string in the international string representation format (YYYY-MM-DD HH:MM:SS ±HHMM).

```
+ (id)dateWithString:(NSString *)aString
```

Parameters

aString

A string that specifies a date and time value in the international string representation format—YYYY-MM-DD HH:MM:SS ±HHMM, where ±HHMM is a time zone offset in hours and minutes from GMT (for example, "2001-03-24 10:45:32 +0600").

You must specify all fields of the format string, including the time zone offset, which must have a plus or minus sign prefix.

Return Value
An NSDate object with a date and time value specified by *aString*.

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

See Also
- initWithString: (page 404)

Declared In
NSCalendarDate.h

dateWithTimeIntervalSince1970:

Creates and returns an `NSDate` object set to the given number of seconds from the first instant of 1 January 1970, GMT.

```
+ (id)dateWithTimeIntervalSince1970:(NSTimeInterval)seconds
```

Parameters

seconds

The number of seconds from the reference date, 1 January 1970, GMT, for the new date. Use a negative argument to specify a date before this date.

Return Value

An `NSDate` object set to *seconds* seconds from the reference date.

Discussion

This method is useful for creating `NSDate` objects from `time_t` values returned by BSD system functions.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- timeIntervalSince1970` (page 407)

Related Sample Code

SharedMemory

Declared In

`NSDate.h`

dateWithTimeIntervalSinceNow:

Creates and returns an `NSDate` object set to a given number of seconds from the current date and time.

```
+ (id)dateWithTimeIntervalSinceNow:(NSTimeInterval)seconds
```

Parameters

seconds

The number of seconds from the current date and time for the new date. Use a negative value to specify a date before the current date.

Return Value

An `NSDate` object set to *seconds* seconds from the current date and time.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- initWithTimeIntervalSinceNow:` (page 405)

Related Sample Code

IdentitySample
SimpleThreads
StickiesExample
TrivialThreads

WhackedTV

Declared In
NSDate.h

dateWithTimeIntervalSinceReferenceDate:

Creates and returns an NSDate object set to a given number of seconds from the first instant of 1 January 2001, GMT.

```
+ (id)dateWithTimeIntervalSinceReferenceDate:(NSTimeInterval)seconds
```

Parameters

seconds

The number of seconds from the absolute reference date (the first instant of 1 January 2001, GMT) for the new date. Use a negative argument to specify a date and time before the reference date.

Return Value
An NSDate object set to *seconds* seconds from the absolute reference date.

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

See Also
- initWithTimeIntervalSinceReferenceDate: (page 406)

Related Sample Code
GridCalendar
NewsReader

Declared In
NSDate.h

distantFuture

Creates and returns an NSDate object representing a date in the distant future.

```
+ (id)distantFuture
```

Return Value
An NSDate object representing a date in the distant future (in terms of centuries).

Discussion
You can pass this value when an NSDate object is required to have the date argument essentially ignored. For example, the NSWindow method nextEventMatchingMask:untilDate:inMode:dequeue: returns nil if an event specified in the event mask does not happen before the specified date. You can use the object returned by distantFuture as the date argument to wait indefinitely for the event to occur.

```
myEvent = [myWindow nextEventMatchingMask:myEventMask
    untilDate:[NSDate distantFuture]
    inMode:NSDefaultRunLoopMode
    dequeue:YES];
```

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

See Also
`+ distantPast` (page 398)

Related Sample Code
CIAnnotation
Core Data HTML Store
DatePicker
LiveVideoMixer2
SeeMyFriends

Declared In
`NSDate.h`

distantPast

Creates and returns an `NSDate` object representing a date in the distant past.

```
+ (id)distantPast
```

Return Value
An `NSDate` object representing a date in the distant past (in terms of centuries).

Discussion
You can use this object as a control date, a guaranteed temporal boundary.

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

See Also
`+ distantFuture` (page 397)

Related Sample Code
CIVideoDemoGL
DatePicker
GLChildWindowDemo
ThreadsExportMovie
Vertex Optimization

Declared In
`NSDate.h`

timeIntervalSinceReferenceDate

Returns the interval between the first instant of 1 January 2001, GMT and the current date and time.

```
+ (NSTimeInterval)timeIntervalSinceReferenceDate
```

Return Value
The interval between the system's absolute reference date (the first instant of 1 January 2001, GMT) and the current date and time.

Discussion
This method is the primitive method for `NSDate`. If you subclass `NSDate`, you must override this method with your own implementation for it.

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

See Also
- `timeIntervalSinceReferenceDate` (page 408)
- `timeIntervalSinceDate:` (page 408)
- `timeIntervalSince1970` (page 407)
- `timeIntervalSinceNow` (page 408)

Declared In
`NSDate.h`

Instance Methods

addTimeInterval:

Returns a new `NSDate` object that is set to a given number of seconds relative to the receiver.

```
- (id)addTimeInterval:(NSTimeInterval)seconds
```

Parameters

seconds

The number of seconds to add to the receiver. Use a negative value for seconds to have the returned object specify a date before the receiver.

Return Value
A new `NSDate` object that is set to *seconds* seconds relative to the receiver. The date returned might have a representation different from the receiver's.

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

See Also
- `initWithTimeInterval:sinceDate:` (page 405)
- `timeIntervalSinceDate:` (page 408)

Declared In
`NSDate.h`

compare:

Returns an `NSComparisonResult` value that indicates the temporal ordering of the receiver and another given date.

```
- (NSComparisonResult)compare:(NSDate *)anotherDate
```

Parameters

anotherDate

The date with which to compare the receiver.

This value must not be `nil`. If the value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

Return Value

If:

- The receiver and *anotherDate* are exactly equal to each other, `NSOrderedSame`
- The receiver is later in time than *anotherDate*, `NSOrderedDescending`
- The receiver is earlier in time than *anotherDate*, `NSOrderedAscending`.

Discussion

This method detects sub-second differences between dates. If you want to compare dates with a less fine granularity, use `timeIntervalSinceDate:` (page 408) to compare the two dates.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `earlierDate:` (page 403)
- `isEqual:` (page 2101) (`NSObject` protocol)
- `laterDate:` (page 407)

Related Sample Code

Reminders

Declared In

`NSDate.h`

dateWithCalendarFormat:timeZone:

Converts the receiver to an `NSCalendarDate` object with a given format string and time zone.

```
- (NSCalendarDate *)dateWithCalendarFormat:(NSString *)formatString
    timeZone:(NSTimeZone *)timeZone
```

Parameters

formatString

The format for the returned string (see Converting Dates to Strings for a discussion of how to create the format string). Pass `nil` to use the default format string, "`%Y-%m-%d %H:%M:%S %z`" (this conforms to the international format `YYYY-MM-DD HH:MM:SS ±HHMM`.)

timeZone
The time zone for the new calendar date. Pass `nil` to use the default time zone—specific to the current locale.

Return Value
A new `NSCalendarDate` object bound to *formatString* and the time zone *timeZone*.

Special Considerations

Important: `NSCalendarDate` is slated for deprecation, and its use is strongly discouraged.

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

See Also
- `- description` (page 401)
- `- descriptionWithCalendarFormat:timeZone:locale:` (page 401)
- `- descriptionWithLocale:` (page 402)
`dateWithString:calendarFormat:` (page 222) (`NSCalendarDate`)

Declared In
`NSCalendarDate.h`

description

Returns a string representation of the receiver.

```
- (NSString *)description
```

Return Value
A string representation of the receiver in the international format `YYYY-MM-DD HH:MM:SS ±HHMM`, where `±HHMM` represents the time zone offset in hours and minutes from GMT (for example, "`2001-03-24 10:45:32 +0600`").

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

See Also
- `- descriptionWithLocale:` (page 402)

Declared In
`NSDate.h`

descriptionWithCalendarFormat:timeZone:locale:

Returns a string representation of the receiver, formatted as specified by given conversion specifiers.

```
- (NSString *)descriptionWithCalendarFormat:(NSString *)formatString
    timeZone:(NSTimeZone *)aTimeZone locale:(id)localeDictionary
```

Parameters

formatString

The format for the returned string (see Converting Dates to Strings for a discussion of how to create the format string). Pass `nil` to use the default format string, "`%Y-%m-%d %H:%M:%S %z`" (this conforms to the international format `YYYY-MM-DD HH:MM:SS ±HHMM`.)

aTimeZone

The time zone in which to represent the receiver. Pass `nil` to use the default time zone—specific to the current locale.

localeDictionary

An `NSDictionary` object containing locale data. To use the user's preferences, you can use `[[NSUserDefaults standardUserDefaults] dictionaryRepresentation]`.

If you pass `nil` or an instance of `NSLocale`, `NSDate` uses the system default locale—this is not the same as the current user's locale.

Return Value

A string representation of the receiver, formatted as specified by the given conversion specifiers.

Discussion

There are several problems with the implementation of this method that cannot be fixed for compatibility reasons. To format a date correctly, you should consider using a date formatter object instead (see `NSDateFormatter` and *Data Formatting Programming Guide for Cocoa*).

You could use this method to print the current time as follows:

```
sprintf(aString, "The current time is %s\n", [[[NSDate  date]
    descriptionWithCalendarFormat:@"%H:%M:%S %Z" timeZone:nil
    locale:[[NSUserDefaults standardUserDefaults] dictionaryRepresentation]]
        UTF8String]);
```

Availability

Available in Mac OS X v10.0 and later.

See Also

`- description` (page 401)

`descriptionWithCalendarFormat:locale:` (page 229) (`NSCalendarDate`)

`- descriptionWithLocale:` (page 402)

Related Sample Code

SharedMemory

Declared In

`NSCalendarDate.h`

descriptionWithLocale:

Returns a string representation of the receiver using the given locale.

```
- (NSString *)descriptionWithLocale:(id)locale
```

Parameters

locale

An NSLocale object.

If you pass nil, NSDate formats the date in the same way as the description (page 401) method.

On Mac OS X v10.4 and earlier, this parameter was an NSDictionary object. If you pass in an NSDictionary object on Mac OS X v10.5, NSDate uses the default user locale—the same as if you passed in [NSLocale currentLocale].

Return Value

A string representation of the receiver, using the given locale, or if the locale argument is nil, in the international format YYYY-MM-DD HH:MM:SS ±HHMM, where ±HHMM represents the time zone offset in hours and minutes from GMT (for example, "2001-03-24 10:45:32 +0600")

Special Considerations

On Mac OS X v10.4 and earlier, *localeDictionary* is an NSDictionary object containing locale data. To use the user's preferences, you can use [[NSUserDefaults standardUserDefaults] dictionaryRepresentation].

Availability

Available in Mac OS X v10.0 and later.

See Also

- description (page 401)

Declared In

NSCalendarDate.h

earlierDate:

Returns the earlier of the receiver and another given date.

```
- (NSDate *)earlierDate:(NSDate *)anotherDate
```

Parameters

anotherDate

The date with which to compare the receiver.

Return Value

The earlier of the receiver and *anotherDate*, determined using timeIntervalSinceDate: (page 408). If the receiver and *anotherDate* represent the same date, returns the receiver.

Availability

Available in Mac OS X v10.0 and later.

See Also

- compare: (page 400)
- isEqual: (page 2101) (NSObject protocol)
- laterDate: (page 407)

Declared In

NSDate.h

init

Returns an `NSDate` object initialized to the current date and time.

```
- (id)init
```

Return Value
An `NSDate` object initialized to the current date and time.

Discussion
This method uses the designated initializer, `initWithTimeIntervalSinceReferenceDate:` (page 406).

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

See Also
`+ date` (page 393)
`- initWithTimeIntervalSinceReferenceDate:` (page 406)

Declared In
`NSDate.h`

initWithString:

Returns an `NSDate` object initialized with a date and time value specified by a given string in the international string representation format.

```
- (id)initWithString:(NSString *)description
```

Parameters
description

A string that specifies a date and time value in the international string representation format—`YYYY-MM-DD HH:MM:SS ±HHMM`, where `±HHMM` is a time zone offset in hours and minutes from GMT (for example, "`2001-03-24 10:45:32 +0600`").

You must specify all fields of the format string, including the time zone offset, which must have a plus or minus sign prefix.

Return Value
An `NSDate` object initialized with a date and time value specified by *aString*.

Discussion
This method uses the designated initializer, `initWithTimeIntervalSinceReferenceDate:` (page 406).

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

See Also
`+ dateWithString:` (page 395)
`- description` (page 401)

Declared In
`NSCalendarDate.h`

initWithTimeInterval:sinceDate:

Returns an `NSDate` object initialized relative to another given date by a given number of seconds.

```
- (id)initWithTimeInterval:(NSTimeInterval)seconds sinceDate:(NSDate *)refDate
```

Parameters

seconds

The number of seconds to add to *refDate*. A negative value means the receiver will be earlier than *refDate*.

refDate

The reference date.

Return Value
An `NSDate` object initialized relative to *refDate* by *seconds* seconds.

Discussion
This method uses the designated initializer, `initWithTimeIntervalSinceReferenceDate:` (page 406).

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

Declared In
`NSDate.h`

initWithTimeIntervalSinceNow:

Returns an `NSDate` object initialized relative to the current date and time by a given number of seconds.

```
- (id)initWithTimeIntervalSinceNow:(NSTimeInterval)seconds
```

Parameters

seconds

The number of seconds from relative to the current date and time to which the receiver should be initialized. A negative value means the returned object will represent a date in the past.

Return Value
An `NSDate` object initialized relative to the current date and time by *seconds* seconds.

Discussion
This method uses the designated initializer, `initWithTimeIntervalSinceReferenceDate:` (page 406).

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

See Also
`+ dateWithTimeIntervalSinceNow:` (page 396)

Related Sample Code
PDFKitLinker2
SimpleScriptingProperties
Vertex Optimization

Declared In
`NSDate.h`

initWithTimeIntervalSinceReferenceDate:

Returns an `NSDate` object initialized relative the first instant of 1 January 2001, GMT by a given number of seconds.

```
- (id)initWithTimeIntervalSinceReferenceDate:(NSTimeInterval)seconds
```

Parameters

seconds

The number of seconds to add to the reference date (the first instant of 1 January 2001, GMT). A negative value means the receiver will be earlier than the reference date.

Return Value

An `NSDate` object initialized relative to the absolute reference date by *seconds* seconds.

Discussion

This method is the designated initializer for the `NSDate` class and is declared primarily for the use of subclasses of `NSDate`. When you subclass `NSDate` to create a concrete date class, you must override this method.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ dateWithTimeIntervalSinceReferenceDate:` (page 397)

Declared In

`NSDate.h`

isEqualToDate:

Returns a Boolean value that indicates whether a given object is an `NSDate` object and exactly equal the receiver.

```
- (BOOL)isEqualToDate:(NSDate *)anotherDate
```

Parameters

anotherDate

The date to compare with the receiver.

Return Value

`YES` if the *anotherDate* is an `NSDate` object and is exactly equal to the receiver, otherwise `NO`.

Discussion

This method detects sub-second differences between dates. If you want to compare dates with a less fine granularity, use `timeIntervalSinceDate:` (page 408) to compare the two dates.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `compare:` (page 400)
- `earlierDate:` (page 403)
- `isEqual:` (page 2101) (`NSObject` protocol)
- `laterDate:` (page 407)

Declared In
NSDate.h

laterDate:

Returns the later of the receiver and another given date.

```
- (NSDate *)laterDate:(NSDate *)anotherDate
```

Parameters

anotherDate
The date with which to compare the receiver.

Return Value
The later of the receiver and *anotherDate*, determined using `timeIntervalSinceDate:` (page 408). If the receiver and *anotherDate* represent the same date, returns the receiver.

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

See Also
- `compare:` (page 400)
- `earlierDate:` (page 403)
- `isEqual:` (page 2101) (`NSObject` protocol)

Declared In
NSDate.h

timeIntervalSince1970

Returns the interval between the receiver and the first instant of 1 January 1970, GMT.

```
- (NSTimeInterval)timeIntervalSince1970
```

Return Value
The interval between the receiver and the reference date, 1 January 1970, GMT. If the receiver is earlier than the reference date, the value is negative.

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

See Also
- `timeIntervalSinceDate:` (page 408)
- `timeIntervalSinceNow` (page 408)
- `timeIntervalSinceReferenceDate` (page 408)
+ `timeIntervalSinceReferenceDate` (page 398)

Declared In
NSDate.h

timeIntervalSinceDate:

Returns the interval between the receiver and another given date.

```
- (NSTimeInterval)timeIntervalSinceDate:(NSDate *)anotherDate
```

Parameters

anotherDate

The date with which to compare the receiver.

Return Value

The interval between the receiver and *anotherDate*. If the receiver is earlier than *anotherDate*, the return value is negative.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `timeIntervalSince1970` (page 407)
- `timeIntervalSinceNow` (page 408)
- `timeIntervalSinceReferenceDate` (page 408)

Related Sample Code

URL CacheInfo

Declared In

`NSDate.h`

timeIntervalSinceNow

Returns the interval between the receiver and the current date and time.

```
- (NSTimeInterval)timeIntervalSinceNow
```

Return Value

The interval between the receiver and the current date and time. If the receiver is earlier than the current date and time, the return value is negative.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `timeIntervalSinceDate:` (page 408)
- `timeIntervalSince1970` (page 407)
- `timeIntervalSinceReferenceDate` (page 408)

Declared In

`NSDate.h`

timeIntervalSinceReferenceDate

Returns the interval between the receiver and the first instant of 1 January 2001, GMT.

```
- (NSTimeInterval)timeIntervalSinceReferenceDate
```

Return Value
The interval between the receiver and the system's absolute reference date (the first instant of 1 January 2001, GMT). If the receiver is earlier than the reference date, the value is negative.

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

See Also
- `- timeIntervalSinceDate:` (page 408)
- `- timeIntervalSinceNow` (page 408)
- `+ timeIntervalSinceReferenceDate` (page 398)

Related Sample Code
CITransitionSelectorSample2
NewsReader
OpenGLCaptureToMovie
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSDate.h`

Constants

NSTimeIntervalSince1970

`NSDate` provides a constant that specifies the number of seconds from 1 January 1970 to the reference date, 1 January 2001.

```
#define NSTimeIntervalSince1970  978307200.0
```

Constants
`NSTimeIntervalSince1970`
The number of seconds from 1 January 1970 to the reference date, 1 January 2001.
Available in Mac OS X v10.0 and later.
Declared in `NSDate.h`.

Discussion
1 January 1970 is the epoch (or starting point) for Unix time.

Declared In
`NSDate.h`

NSDateComponents Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.4 and later.
Declared in	Foundation/NSCalendar.h
Companion guide	Date and Time Programming Guide for Cocoa
Related sample code	Birthdays Reminders

Overview

`NSDateComponents` encapsulates the components of a date in an extendable, object-oriented manner. It is used to specify a date by providing the temporal components that make up a date and time: hour, minutes, seconds, day, month, year, and so on. It can also be used to specify a duration of time, for example, 5 hours and 16 minutes. An `NSDateComponents` object is not required to define all the component fields. When a new instance of `NSDateComponents` is created the date components are set to `NSUndefinedDateComponent`.

Important: An `NSDateComponents` object is meaningless in itself; you need to know what calendar it is interpreted against, and you need to know whether the values are absolute values of the units, or quantities of the units.

An instance of `NSDateComponents` is not responsible for answering questions about a date beyond the information with which it was initialized. For example, if you initialize one with May 6, 2004, its weekday is `NSUndefinedDateComponent`, not Thursday. To get the correct day of the week, you must create a suitable instance of `NSCalendar`, create an `NSDate` object using `dateFromComponents:` and then use `components:fromDate:` to retrieve the weekday—as illustrated in the following example.

```
NSDateComponents *comps = [[NSDateComponents alloc] init];
[comps setDay:6];
[comps setMonth:5];
[comps setYear:2004];
NSCalendar *gregorian = [[NSCalendar alloc]
    initWithCalendarIdentifier:NSGregorianCalendar];
NSDate *date = [gregorian dateFromComponents:comps];
```

```
[comps release];
NSDateComponents *weekdayComponents =
    [gregorian components:NSWeekdayCalendarUnit fromDate:date];
int weekday = [weekdayComponents weekday];
```

For more details, see Calendars in *Date and Time Programming Guide for Cocoa*.

Tasks

Getting Information About an NSDateComponents Object

- `era` (page 413)
  Returns the number of era units for the receiver.
- `year` (page 422)
  Returns the number of year units for the receiver.
- `month` (page 415)
  Returns the number of month units for the receiver.
- `day` (page 413)
  Returns the number of day units for the receiver.
- `hour` (page 414)
  Returns the number of hour units for the receiver.
- `minute` (page 414)
  Returns the number of minute units for the receiver.
- `second` (page 415)
  Returns the number of second units for the receiver.
- `week` (page 420)
  Returns the number of week units for the receiver.
- `weekday` (page 421)
  Returns the number of weekday units for the receiver.
- `weekdayOrdinal` (page 421)
  Returns the ordinal number of weekday units for the receiver.

Setting Information for an NSDateComponents Object

- `setEra:` (page 416)
  Sets the number of era units for the receiver.
- `setYear:` (page 420)
  Sets the number of year units for the receiver.
- `setMonth:` (page 417)
  Sets the number of month units for the receiver.
- `setDay:` (page 415)
  Sets the number of day units for the receiver.

- `setHour:` (page 416)
  Sets the number of hour units for the receiver.
- `setMinute:` (page 417)
  Sets the number of minute units for the receiver.
- `setSecond:` (page 418)
  Sets the number of second units for the receiver.
- `setWeek:` (page 418)
  Sets the number of week units for the receiver.
- `setWeekday:` (page 419)
  Sets the number of weekday units for the receiver.
- `setWeekdayOrdinal:` (page 419)
  Sets the ordinal number of weekday units for the receiver.

Instance Methods

day

Returns the number of day units for the receiver.

```
- (NSInteger)day
```

Return Value
The number of day units for the receiver.

Discussion
This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setDay:` (page 415)

Related Sample Code
Birthdays

Declared In
`NSCalendar.h`

era

Returns the number of era units for the receiver.

```
- (NSInteger)era
```

Return Value
The number of era units for the receiver.

Discussion
This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setEra:` (page 416)

Declared In
`NSCalendar.h`

hour

Returns the number of hour units for the receiver.

```
- (NSInteger)hour
```

Return Value
The number of hour units for the receiver.

Discussion
This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setHour:` (page 416)

Declared In
`NSCalendar.h`

minute

Returns the number of minute units for the receiver.

```
- (NSInteger)minute
```

Return Value
The number of minute units for the receiver.

Discussion
This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setMinute:` (page 417)

Declared In
NSCalendar.h

month

Returns the number of month units for the receiver.

```
- (NSInteger)month
```

Return Value
The number of month units for the receiver.

Discussion
This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setMonth: (page 417)

Declared In
NSCalendar.h

second

Returns the number of second units for the receiver.

```
- (NSInteger)second
```

Return Value
The number of second units for the receiver.

Discussion
This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setSecond: (page 418)

Declared In
NSCalendar.h

setDay:

Sets the number of day units for the receiver.

```
- (void)setDay:(NSInteger)v
```

Parameters

v

The number of day units for the receiver.

Discussion
This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `day` (page 413)

Related Sample Code
Reminders

Declared In
`NSCalendar.h`

setEra:

Sets the number of era units for the receiver.

```
- (void)setEra:(NSInteger)v
```

Parameters

v

The number of era units for the receiver.

Discussion
This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `era` (page 413)

Declared In
`NSCalendar.h`

setHour:

Sets the number of hour units for the receiver.

```
- (void)setHour:(NSInteger)v
```

Parameters

v

The number of hour units for the receiver.

Discussion
This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `hour` (page 414)

Related Sample Code
Reminders

Declared In
`NSCalendar.h`

setMinute:

Sets the number of minute units for the receiver.

```
- (void)setMinute:(NSInteger)v
```

Parameters

v

The number of minute units for the receiver.

Discussion
This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `minute` (page 414)

Related Sample Code
Reminders

Declared In
`NSCalendar.h`

setMonth:

Sets the number of month units for the receiver.

```
- (void)setMonth:(NSInteger)v
```

Parameters

v

The number of month units for the receiver.

Discussion
This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `month` (page 415)

Related Sample Code
Reminders

Declared In
`NSCalendar.h`

setSecond:

Sets the number of second units for the receiver.

```
- (void)setSecond:(NSInteger)v
```

Parameters

v

The number of second units for the receiver.

Discussion
This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `second` (page 415)

Related Sample Code
Reminders

Declared In
`NSCalendar.h`

setWeek:

Sets the number of week units for the receiver.

```
- (void)setWeek:(NSInteger)v
```

Parameters

v

The number of week units for the receiver.

Discussion
This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `week` (page 420)

Declared In
`NSCalendar.h`

setWeekday:

Sets the number of weekday units for the receiver.

```
- (void)setWeekday:(NSInteger)v
```

Parameters

v
The number of weekday units for the receiver.

Discussion
Weekday units are the numbers 1 through *n*, where *n* is the number of days in the week. For example, in the Gregorian calendar, *n* is 7 and Sunday is represented by 1.

This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `weekday` (page 421)

Declared In
`NSCalendar.h`

setWeekdayOrdinal:

Sets the ordinal number of weekday units for the receiver.

```
- (void)setWeekdayOrdinal:(NSInteger)v
```

Parameters

v
The ordinal number of weekday units for the receiver.

Discussion
Weekday ordinal units represent the position of the weekday within the next larger calendar unit, such as the month. For example, *2* is the weekday ordinal unit for the *second* Friday of the month.

This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `weekdayOrdinal` (page 421)

Declared In
`NSCalendar.h`

setYear:

Sets the number of year units for the receiver.

```
- (void)setYear:(NSInteger)v
```

Parameters

v
The number of year units for the receiver.

Discussion
This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `year` (page 422)

Related Sample Code
Reminders

Declared In
`NSCalendar.h`

week

Returns the number of week units for the receiver.

```
- (NSInteger)week
```

Return Value
The number of week units for the receiver.

Discussion
This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setWeek:` (page 418)

Declared In
`NSCalendar.h`

weekday

Returns the number of weekday units for the receiver.

```
- (NSInteger)weekday
```

Return Value
The number of weekday units for the receiver.

Discussion
Weekday units are the numbers 1 through *n*, where *n* is the number of days in the week. For example, in the Gregorian calendar, *n* is 7 and Sunday is represented by 1.

This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setWeekday:` (page 419)

Related Sample Code
Birthdays

Declared In
`NSCalendar.h`

weekdayOrdinal

Returns the ordinal number of weekday units for the receiver.

```
- (NSInteger)weekdayOrdinal
```

Return Value
The ordinal number of weekday units for the receiver.

Discussion
Weekday ordinal units represent the position of the weekday within the next larger calendar unit, such as the month. For example, *2* is the weekday ordinal unit for the *second* Friday of the month.

This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setWeekdayOrdinal: (page 419)

Declared In
NSCalendar.h

year

Returns the number of year units for the receiver.

```
- (NSInteger)year
```

Return Value
The number of year units for the receiver.

Discussion
This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setYear: (page 420)

Declared In
NSCalendar.h

Constants

NSDateComponents undefined component identifier

This constant specifies that an NSDateComponents component is undefined.

```
enum {
    NSUndefinedDateComponent = 0x7fffffff
};
```

Constants
NSUndefinedDateComponent
Specifies that the component is undefined.

Available in Mac OS X v10.4 and later.

Declared in NSCalendar.h.

Declared In
NSCalendar.h

NSDateFormatter Class Reference

Inherits from	NSFormatter : NSObject
Conforms to	NSCoding (NSFormatter) NSCopying (NSFormatter) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSDateFormatter.h
Companion guide	Data Formatting Programming Guide for Cocoa
Related sample code	Core Data HTML Store DatePicker iSpend Mountains Reminders

Overview

Instances of `NSDateFormatter` create string representations of `NSDate` (and `NSCalendarDate`) objects, and convert textual representations of dates and times into `NSDate` objects. You can express the representation of dates and times flexibly: "Thu 22 Dec 1994" is just as acceptable as "12/22/94."

With Mac OS X v10.4 and later, `NSDateFormatter` has two modes of operation (or behaviors). By default, instances of `NSDateFormatter` have the same behavior as they did on Mac OS X versions 10.0 to 10.3. You can, however, configure instances (or set a default for all instances) to adopt a new behavior implemented for Mac OS X version 10.4. See *Data Formatting Programming Guide for Cocoa* for a full description of the old and new behaviors.

iPhone OS Note: iPhone OS supports only the modern 10.4+ behavior. 10.0-style methods and format strings are not available on iPhone OS.

If you initialize a formatter using `initWithDateFormat:allowNaturalLanguage:` (page 435), you are (for backwards compatibility reasons) creating an "old-style" date formatter. To use the new behavior, you initialize the formatter with `init` (page 434). If you have not set the default class behavior (see `setDefaultFormatterBehavior:` (page 429)), you send the instance a `setFormatterBehavior:` (page 441) message with the argument `NSDateFormatterBehavior10_4`. You can then set the date format as appropriate, typically using a format style as illustrated in the following code fragment.

```
// assume default behavior set for class using
// [NSDateFormatter setDefaultFormatterBehavior:NSDateFormatterBehavior10_4];

NSDateFormatter *dateFormatter = [[[NSDateFormatter alloc] init]  autorelease];
[dateFormatter setDateStyle:NSDateFormatterMediumStyle];
[dateFormatter setTimeStyle:NSDateFormatterNoStyle];

NSDate *date = [NSDate dateWithTimeIntervalSinceReferenceDate:118800];
NSString *formattedDateString = [dateFormatter stringFromDate:date];
NSLog(@"formattedDateString for locale %@: %@",
        [[dateFormatter locale] localeIdentifier], formattedDateString);

// Output: formattedDateString for locale en_US: Jan 2, 2001
```

Note that the format for a given style is dependent on a user's preferences, including the locale setting.

Note also that by default the new-style formatter returns `NSDate` objects instead of `NSCalendarDate` objects. You can change this behavior using `setGeneratesCalendarDates:` (page 441).

Tasks

Initializing a Date Formatter

- `init` (page 434)
  Initializes and returns an `NSDateFormatter` instance.
- `initWithDateFormat:allowNaturalLanguage:` (page 435)
  Initializes and returns an `NSDateFormatter` instance that uses the Mac OS X v10.0 formatting behavior and the given date format string in its conversions.

Managing Behavior

- `allowsNaturalLanguage` (page 429)
  Returns a Boolean value that indicates whether the receiver attempts to process dates entered as a vernacular string.
- `formatterBehavior` (page 432)
  Returns the formatter behavior for the receiver.
- `setFormatterBehavior:` (page 441)
  Sets the formatter behavior for the receiver.
+ `defaultFormatterBehavior` (page 428)
  Returns the default formatting behavior for instances of the class.
+ `setDefaultFormatterBehavior:` (page 429)
  Sets the default formatting behavior for instances of the class.
- `generatesCalendarDates` (page 433)
  Returns a Boolean value that indicates whether the receiver generates calendar dates.
- `setGeneratesCalendarDates:` (page 441)
  Sets whether the receiver generates calendar dates.

- `isLenient` (page 436)
  Returns a Boolean value that indicates whether the receiver uses heuristics when parsing a string.
- `setLenient:` (page 442)
  Sets whether the receiver uses heuristics when parsing a string.

Converting Objects

- `dateFromString:` (page 431)
  Returns a date representation of a given string interpreted using the receiver's current settings.
- `stringFromDate:` (page 456)
  Returns a string representation of a given date formatted using the receiver's current settings.
- `getObjectValue:forString:range:error:` (page 433)
  Returns by reference a date representation of a given string and the range of the string used, and returns a Boolean value that indicates whether the string could be parsed.

Managing Formats and Styles

- `dateFormat` (page 430)
  Returns the date format string used by the receiver.
- `setDateFormat:` (page 439)
  Sets the date format for the receiver.
- `dateStyle` (page 431)
  Returns the date style of the receiver.
- `setDateStyle:` (page 439)
  Sets the date style of the receiver.
- `timeStyle` (page 456)
  Returns the time style of the receiver.
- `setTimeStyle:` (page 448)
  Sets the time style of the receiver.

Managing Attributes

- `calendar` (page 430)
  Returns the calendar for the receiver.
- `setCalendar:` (page 439)
  Sets the calendar for the receiver.
- `defaultDate` (page 432)
  Returns the default date for the receiver.
- `setDefaultDate:` (page 440)
  Sets the default date for the receiver.
- `locale` (page 436)
  Returns the locale for the receiver.

- `setLocale:` (page 442)
  Sets the locale for the receiver.
- `timeZone` (page 457)
  Returns the time zone for the receiver.
- `setTimeZone:` (page 449)
  Sets the time zone for the receiver.
- `twoDigitStartDate` (page 457)
  Returns the earliest date that can be denoted by a two-digit year specifier.
- `setTwoDigitStartDate:` (page 449)
  Sets the two-digit start date for the receiver.
- `gregorianStartDate` (page 434)
  Returns the start date of the Gregorian calendar for the receiver.
- `setGregorianStartDate:` (page 441)
  Sets the start date of the Gregorian calendar for the receiver.

Managing AM and PM Symbols

- `AMSymbol` (page 430)
  Returns the AM symbol for the receiver.
- `setAMSymbol:` (page 438)
  Sets the AM symbol for the receiver.
- `PMSymbol` (page 437)
  Returns the PM symbol for the receiver.
- `setPMSymbol:` (page 443)
  Sets the PM symbol for the receiver.

Managing Weekday Symbols

- `weekdaySymbols` (page 459)
  Returns the array of weekday symbols for the receiver.
- `setWeekdaySymbols:` (page 451)
  Sets the weekday symbols for the receiver.
- `shortWeekdaySymbols` (page 454)
  Returns the array of short weekday symbols for the receiver.
- `setShortWeekdaySymbols:` (page 447)
  Sets the short weekday symbols for the receiver.
- `veryShortWeekdaySymbols` (page 459)
  Returns the array of very short weekday symbols for the receiver.
- `setVeryShortWeekdaySymbols:` (page 451)
  Sets the vert short weekday symbols for the receiver
- `standaloneWeekdaySymbols` (page 455)
  Returns the array of standalone weekday symbols for the receiver.

- `setStandaloneWeekdaySymbols:` (page 448)
  Sets the standalone weekday symbols for the receiver.
- `shortStandaloneWeekdaySymbols` (page 454)
  Returns the array of short standalone weekday symbols for the receiver.
- `setShortStandaloneWeekdaySymbols:` (page 446)
  Sets the short standalone weekday symbols for the receiver.
- `veryShortStandaloneWeekdaySymbols` (page 458)
  Returns the array of very short standalone weekday symbols for the receiver.
- `setVeryShortStandaloneWeekdaySymbols:` (page 450)
  Sets the very short standalone weekday symbols for the receiver.

Managing Month Symbols

- `monthSymbols` (page 437)
  Returns the month symbols for the receiver.
- `setMonthSymbols:` (page 443)
  Sets the month symbols for the receiver.
- `shortMonthSymbols` (page 452)
  Returns the array of short month symbols for the receiver.
- `setShortMonthSymbols:` (page 444)
  Sets the short month symbols for the receiver.
- `veryShortMonthSymbols` (page 458)
  Returns the very short month symbols for the receiver.
- `setVeryShortMonthSymbols:` (page 450)
  Sets the very short month symbols for the receiver.
- `standaloneMonthSymbols` (page 455)
  Returns the standalone month symbols for the receiver.
- `setStandaloneMonthSymbols:` (page 447)
  Sets the standalone month symbols for the receiver.
- `shortStandaloneMonthSymbols` (page 453)
  Returns the short standalone month symbols for the receiver.
- `setShortStandaloneMonthSymbols:` (page 445)
  Sets the short standalone month symbols for the receiver.
- `veryShortStandaloneMonthSymbols` (page 458)
  Returns the very short month symbols for the receiver.
- `setVeryShortStandaloneMonthSymbols:` (page 450)
  Sets the very short standalone month symbols for the receiver.

Managing Quarter Symbols

- `quarterSymbols` (page 438)
  Returns the quarter symbols for the receiver.

- `setQuarterSymbols:` (page 444)
  Sets the quarter symbols for the receiver.
- `shortQuarterSymbols` (page 452)
  Returns the short quarter symbols for the receiver.
- `setShortQuarterSymbols:` (page 445)
  Sets the short quarter symbols for the receiver.
- `standaloneQuarterSymbols` (page 455)
  Returns the standalone quarter symbols for the receiver.
- `setStandaloneQuarterSymbols:` (page 447)
  Sets the standalone quarter symbols for the receiver.
- `shortStandaloneQuarterSymbols` (page 453)
  Returns the short standalone quarter symbols for the receiver.
- `setShortStandaloneQuarterSymbols:` (page 446)
  Sets the short standalone quarter symbols for the receiver.

Managing Era Symbols

- `eraSymbols` (page 432)
  Returns the era symbols for the receiver.
- `setEraSymbols:` (page 440)
  Sets the era symbols for the receiver.
- `longEraSymbols` (page 437)
  Returns the long era symbols for the receiver
- `setLongEraSymbols:` (page 443)
  Sets the long era symbols for the receiver.

Class Methods

defaultFormatterBehavior

Returns the default formatting behavior for instances of the class.

```
+ (NSDateFormatterBehavior)defaultFormatterBehavior
```

Return Value
The default formatting behavior for instances of the class. For possible values, see NSDateFormatterBehavior (page 461).

Discussion
The default is `NSDateFormatterBehavior10_0`.

Availability
Available in Mac OS X v10.4 and later.

See Also
`+ setDefaultFormatterBehavior:` (page 429).

- formatterBehavior (page 432)
- setFormatterBehavior: (page 441)

Declared In
NSDateFormatter.h

setDefaultFormatterBehavior:

Sets the default formatting behavior for instances of the class.

```
+ (void)setDefaultFormatterBehavior:(NSDateFormatterBehavior)behavior
```

Parameters

behavior

The default formatting behavior for instances of the class. For possible values, see NSDateFormatterBehavior (page 461).

Availability
Available in Mac OS X v10.4 and later.

See Also
+ defaultFormatterBehavior (page 428)
- formatterBehavior (page 432)
- setFormatterBehavior: (page 441)

Related Sample Code
DatePicker

Declared In
NSDateFormatter.h

Instance Methods

allowsNaturalLanguage

Returns a Boolean value that indicates whether the receiver attempts to process dates entered as a vernacular string.

```
- (BOOL)allowsNaturalLanguage
```

Return Value
YES if the receiver attempts to process dates entered as a vernacular string ("today," "next week," "dinner time," and so on), otherwise NO.

Discussion
Natural-language processing supports only a limited set of colloquial phrases, primarily in English. It may give unexpected results, and its use is strongly discouraged.

Special Considerations

This method is for use with formatters using NSDateFormatterBehavior10_0 behavior.

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

Declared In
NSDateFormatter.h

AMSymbol

Returns the AM symbol for the receiver.

```
- (NSString *)AMSymbol
```

Return Value
The AM symbol for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setAMSymbol: (page 438)
- PMSymbol (page 437)
- setPMSymbol: (page 443)

Declared In
NSDateFormatter.h

calendar

Returns the calendar for the receiver.

```
- (NSCalendar *)calendar
```

Return Value
The calendar for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setCalendar: (page 439)

Declared In
NSDateFormatter.h

dateFormat

Returns the date format string used by the receiver.

```
- (NSString *)dateFormat
```

Return Value
The date format string used by the receiver.

Discussion
See Date Format String Syntax (MacÂ OSÂ X Versions 10.0 to 10.3) for a list of the conversion specifiers permitted in date format strings.

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

See Also
- `setDateFormat:` (page 439)

Declared In
`NSDateFormatter.h`

dateFromString:

Returns a date representation of a given string interpreted using the receiver's current settings.

```
- (NSDate *)dateFromString:(NSString *)string
```

Parameters
string
The string to parse.

Return Value
A date representation of *string* interpreted using the receiver's current settings.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `getObjectValue:forString:range:error:` (page 433)
- `stringFromDate:` (page 456)

Related Sample Code
Reminders

Declared In
`NSDateFormatter.h`

dateStyle

Returns the date style of the receiver.

```
- (NSDateFormatterStyle)dateStyle
```

Return Value
The date style of the receiver. For possible values, see NSDateFormatterStyle (page 460).

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setDateStyle:` (page 439)

Declared In
NSDateFormatter.h

defaultDate

Returns the default date for the receiver.

```
- (NSDate *)defaultDate
```

Return Value
The default date for the receiver.

Discussion
The default default date is `nil`.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setDefaultDate: (page 440)

Declared In
NSDateFormatter.h

eraSymbols

Returns the era symbols for the receiver.

```
- (NSArray *)eraSymbols
```

Return Value
An array containing `NSString` objects representing the era symbols for the receiver (for example, {"B.C.E.", "C.E."}).

Availability
Available in Mac OS X v10.4 and later.

See Also
- setEraSymbols: (page 440)
- longEraSymbols (page 437)

Declared In
NSDateFormatter.h

formatterBehavior

Returns the formatter behavior for the receiver.

```
- (NSDateFormatterBehavior)formatterBehavior
```

Return Value
The formatter behavior for the receiver. For possible values, see `NSDateFormatterBehavior` (page 461).

Availability
Available in Mac OS X v10.4 and later.

See Also
`+ defaultFormatterBehavior` (page 428).
`+ setDefaultFormatterBehavior:` (page 429)
`- setFormatterBehavior:` (page 441)

Declared In
`NSDateFormatter.h`

generatesCalendarDates

Returns a Boolean value that indicates whether the receiver generates calendar dates.

```
- (BOOL)generatesCalendarDates
```

Return Value
`YES` if the receiver generates calendar dates, otherwise `NO`.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setGeneratesCalendarDates:` (page 441)

Declared In
`NSDateFormatter.h`

getObjectValue:forString:range:error:

Returns by reference a date representation of a given string and the range of the string used, and returns a Boolean value that indicates whether the string could be parsed.

```
- (BOOL)getObjectValue:(id *)obj forString:(NSString *)string range:(inout NSRange
    *)rangep error:(NSError **)error
```

Parameters

obj
If the receiver is able to parse *string*, upon return contains a date representation of *string*.

string
The string to parse.

rangep
If the receiver is able to parse *string*, upon return contains the range of *string* used to create the date.

error
If the receiver is unable to create a date by parsing *string*, upon return contains an NSError object that describes the problem.

Return Value
`YES` if the receiver can create a date by parsing *string*, otherwise `NO`.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `dateFromString:` (page 431)
- `stringForObjectValue:` (page 680)

Related Sample Code
iSpend

Declared In
`NSDateFormatter.h`

gregorianStartDate

Returns the start date of the Gregorian calendar for the receiver.

```
- (NSDate *)gregorianStartDate
```

Return Value
The start date of the Gregorian calendar for the receiver.

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

See Also
- `setGregorianStartDate:` (page 441)

Declared In
`NSDateFormatter.h`

init

Initializes and returns an `NSDateFormatter` instance.

```
- (id)init
```

Return Value
An `NSDateFormatter` instance initialized with locale, time zone, calendar, and behavior set to the appropriate default values.

Discussion
There are many new attributes you can get and set on a 10.4-style date formatter, including the locale, time zone, calendar, format string, the two-digit-year cross-over date, the default date which provides unspecified components, and there is also access to the various textual strings, like the month names. You are encouraged, however, not to change individual settings. Instead you should accept the default settings established on initialization and specify the format using `setDateStyle:` (page 439), `setTimeStyle:` (page 448), and appropriate style constants (see `NSDateFormatterStyle` (page 460)—these are styles that the user can configure in the International preferences panel in System Preferences).

Special Considerations

If you want the Mac OS X 10.4 behavior but have not set the class's default behavior to `NSDateFormatterBehavior10_4`, you also need to send the new instance a `setFormatterBehavior:` (page 441) message with the argument `NSDateFormatterBehavior10_4`.

Availability

Available in Mac OS X v10.4 and later.

See Also

- `initWithDateFormat:allowNaturalLanguage:` (page 435)
- `setDateStyle:` (page 439)
- `setTimeStyle:` (page 448)

Declared In

`NSDateFormatter.h`

initWithDateFormat:allowNaturalLanguage:

Initializes and returns an `NSDateFormatter` instance that uses the Mac OS X v10.0 formatting behavior and the given date format string in its conversions.

```
- (id)initWithDateFormat:(NSString *)format allowNaturalLanguage:(BOOL)flag
```

Parameters

format

The format for the receiver. See Date Format String Syntax (MacÂ OSÂ X Versions 10.0 to 10.3) for a list of conversion specifiers permitted in date format strings.

flag

A flag that specifies whether the receiver should process dates entered as expressions in the vernacular (for example, "tomorrow")—`YES` means that it should.

Return Value

An initialized `NSDateFormatter` instance that uses *format* in its conversions and that *uses the Mac OS X v10.0 formatting behavior*.

Discussion

`NSDateFormatter` attempts natural-language processing only after it fails to interpret an entered string according to *format*. Natural-language processing supports only a limited set of colloquial phrases, primarily in English. It may give unexpected results, and its use is strongly discouraged.

The following example creates a date formatter with the format string (for example) "Mar 15 1994" and then associates the formatter with the cells of a form (`contactsForm`):

```
NSDateFormatter *dateFormat = [[NSDateFormatter alloc]
    initWithDateFormat:@"%b %d %Y" allowNaturalLanguage:NO];
[[contactsForm cells] makeObjectsPerformSelector:@selector(setFormatter:)
     withObject:dateFormat];
```

Important: You cannot use this method to initialize a formatter with the Mac OS X v10.4 formatting behavior, you must use `init` (page 434).

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

See Also
`- init` (page 434)

Declared In
`NSDateFormatter.h`

isLenient

Returns a Boolean value that indicates whether the receiver uses heuristics when parsing a string.

```
- (BOOL)isLenient
```

Return Value
`YES` if the receiver has been set to use heuristics when parsing a string to guess at the date which is intended, otherwise `NO`.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setLenient:` (page 442)

Declared In
`NSDateFormatter.h`

locale

Returns the locale for the receiver.

```
- (NSLocale *)locale
```

Return Value
The locale for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setLocale:` (page 442)

Declared In
`NSDateFormatter.h`

longEraSymbols

Returns the long era symbols for the receiver

```
- (NSArray *)longEraSymbols
```

Return Value
An array containing `NSString` objects representing the era symbols for the receiver (for example, {“Before Common Era”, “Common Era”}).

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

See Also
- `setLongEraSymbols:` (page 443)
- `eraSymbols` (page 432)

Declared In
`NSDateFormatter.h`

monthSymbols

Returns the month symbols for the receiver.

```
- (NSArray *)monthSymbols
```

Return Value
An array of `NSString` objects that specify the month symbols for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setMonthSymbols:` (page 443)
- `shortMonthSymbols` (page 452)
- `veryShortMonthSymbols` (page 458)
- `standaloneMonthSymbols` (page 455)
- `shortStandaloneMonthSymbols` (page 453)
- `veryShortStandaloneMonthSymbols` (page 458)

Declared In
`NSDateFormatter.h`

PMSymbol

Returns the PM symbol for the receiver.

```
- (NSString *)PMSymbol
```

Return Value
The PM symbol for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setPMSymbol:` (page 443)
- `AMSymbol` (page 430)
- `setAMSymbol:` (page 438)

Declared In
`NSDateFormatter.h`

quarterSymbols

Returns the quarter symbols for the receiver.

```
- (NSArray *)quarterSymbols
```

Return Value
An array containing `NSString` objects representing the quarter symbols for the receiver.

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

See Also
- `setQuarterSymbols:` (page 444)
- `shortQuarterSymbols` (page 452)
- `standaloneQuarterSymbols` (page 455)
- `shortStandaloneQuarterSymbols` (page 453)

Declared In
`NSDateFormatter.h`

setAMSymbol:

Sets the AM symbol for the receiver.

```
- (void)setAMSymbol:(NSString *)string
```

Parameters
string
The AM symbol for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `AMSymbol` (page 430)
- `PMSymbol` (page 437)
- `setPMSymbol:` (page 443)

Declared In
`NSDateFormatter.h`

setCalendar:

Sets the calendar for the receiver.

```
- (void)setCalendar:(NSCalendar *)calendar
```

Parameters

calendar

The calendar for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- calendar (page 430)

Declared In
NSDateFormatter.h

setDateFormat:

Sets the date format for the receiver.

```
- (void)setDateFormat:(NSString *)string
```

Parameters

string

The date format for the receiver. See *Data Formatting Programming Guide for Cocoa* for a list of the conversion specifiers permitted in date format strings.

Availability
Available in Mac OS X v10.4 and later.

See Also
- dateFormat (page 430).

Declared In
NSDateFormatter.h

setDateStyle:

Sets the date style of the receiver.

```
- (void)setDateStyle:(NSDateFormatterStyle)style
```

Parameters

style

The date style of the receiver. For possible values, see NSDateFormatterStyle (page 460).

Availability
Available in Mac OS X v10.4 and later.

See Also
- dateStyle (page 431).

Related Sample Code
DatePicker
iSpend
Mountains
NSOperationSample
Reminders

Declared In
NSDateFormatter.h

setDefaultDate:

Sets the default date for the receiver.

```
- (void)setDefaultDate:(NSDate *)date
```

Parameters

date

The default date for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- defaultDate (page 432)

Declared In
NSDateFormatter.h

setEraSymbols:

Sets the era symbols for the receiver.

```
- (void)setEraSymbols:(NSArray *)array
```

Parameters

array

An array containing NSString objects representing the era symbols for the receiver (for example, {"B.C.E.", "C.E."}).

Availability
Available in Mac OS X v10.4 and later.

See Also
- eraSymbols (page 432)
- longEraSymbols (page 437)

Declared In
NSDateFormatter.h

setFormatterBehavior:

Sets the formatter behavior for the receiver.

```
- (void)setFormatterBehavior:(NSDateFormatterBehavior)behavior
```

Parameters

behavior

The formatter behavior for the receiver. For possible values, see `NSDateFormatterBehavior` (page 461).

Availability
Available in Mac OS X v10.4 and later.

See Also
`+ defaultFormatterBehavior` (page 428).
`+ setDefaultFormatterBehavior:` (page 429)
`- formatterBehavior` (page 432)

Declared In
`NSDateFormatter.h`

setGeneratesCalendarDates:

Sets whether the receiver generates calendar dates.

```
- (void)setGeneratesCalendarDates:(BOOL)b
```

Parameters

b

A Boolean value that specifies whether the receiver generates calendar dates.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- generatesCalendarDates` (page 433).

Declared In
`NSDateFormatter.h`

setGregorianStartDate:

Sets the start date of the Gregorian calendar for the receiver.

```
- (void)setGregorianStartDate:(NSDate *)array
```

Parameters

array

The start date of the Gregorian calendar for the receiver.

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

See Also
- gregorianStartDate (page 434)

Declared In
NSDateFormatter.h

setLenient:

Sets whether the receiver uses heuristics when parsing a string.

```
- (void)setLenient:(BOOL)b
```

Parameters
b
YES to use heuristics when parsing a string to guess at the date which is intended, otherwise NO.

Discussion
If a formatter is set to be lenient, when parsing a string it uses heuristics to guess at the date which is intended. As with any guessing, it may get the result date wrong (that is, a date other than that which was intended).

Availability
Available in Mac OS X v10.4 and later.

See Also
- isLenient (page 436)

Declared In
NSDateFormatter.h

setLocale:

Sets the locale for the receiver.

```
- (void)setLocale:(NSLocale *)locale
```

Parameters
locale
The locale for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- locale (page 436)

Related Sample Code
Mountains

Declared In
NSDateFormatter.h

setLongEraSymbols:

Sets the long era symbols for the receiver.

```
- (void)setLongEraSymbols:(NSArray *)array
```

Parameters

array

An array containing `NSString` objects representing the era symbols for the receiver (for example, {“Before Common Era”, “Common Era”}).

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

See Also
- `longEraSymbols` (page 437)
- `eraSymbols` (page 432)

Declared In
`NSDateFormatter.h`

setMonthSymbols:

Sets the month symbols for the receiver.

```
- (void)setMonthSymbols:(NSArray *)array
```

Parameters

array

An array of `NSString` objects that specify the month symbols for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `monthSymbols` (page 437)
- `setShortMonthSymbols:` (page 444)
- `setVeryShortMonthSymbols:` (page 450)
- `setStandaloneMonthSymbols:` (page 447)
- `setShortStandaloneMonthSymbols:` (page 445)
- `setVeryShortStandaloneMonthSymbols:` (page 450)

Declared In
`NSDateFormatter.h`

setPMSymbol:

Sets the PM symbol for the receiver.

```
- (void)setPMSymbol:(NSString *)string
```

Parameters

string

The PM symbol for the receiver.

Availability

Available in Mac OS X v10.4 and later.

See Also

- `PMSymbol` (page 437)
- `AMSymbol` (page 430)
- `setAMSymbol:` (page 438)

Declared In

`NSDateFormatter.h`

setQuarterSymbols:

Sets the quarter symbols for the receiver.

```
- (void)setQuarterSymbols:(NSArray *)array
```

Parameters

array

An array of `NSString` objects that specify the quarter symbols for the receiver.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `quarterSymbols` (page 438)
- `setShortQuarterSymbols:` (page 445)
- `setStandaloneQuarterSymbols:` (page 447)
- `setShortStandaloneQuarterSymbols:` (page 446)

Declared In

`NSDateFormatter.h`

setShortMonthSymbols:

Sets the short month symbols for the receiver.

```
- (void)setShortMonthSymbols:(NSArray *)array
```

Parameters

array

An array of `NSString` objects that specify the short month symbols for the receiver.

Availability

Available in Mac OS X v10.4 and later.

See Also

- `shortMonthSymbols` (page 452)

- setMonthSymbols: (page 443)
- setVeryShortMonthSymbols: (page 450)
- setStandaloneMonthSymbols: (page 447)
- setShortStandaloneMonthSymbols: (page 445)
- setVeryShortStandaloneMonthSymbols: (page 450)

Declared In
NSDateFormatter.h

setShortQuarterSymbols:

Sets the short quarter symbols for the receiver.

```
- (void)setShortQuarterSymbols:(NSArray *)array
```

Parameters

array

An array of NSString objects that specify the short quarter symbols for the receiver.

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

See Also
- shortQuarterSymbols (page 452)
- setQuarterSymbols: (page 444)
- setStandaloneQuarterSymbols: (page 447)
- setShortStandaloneQuarterSymbols: (page 446)

Declared In
NSDateFormatter.h

setShortStandaloneMonthSymbols:

Sets the short standalone month symbols for the receiver.

```
- (void)setShortStandaloneMonthSymbols:(NSArray *)array
```

Parameters

array

An array of NSString objects that specify the short standalone month symbols for the receiver.

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

See Also
- shortStandaloneMonthSymbols (page 453)
- setMonthSymbols: (page 443)
- setShortMonthSymbols: (page 444)
- setVeryShortMonthSymbols: (page 450)
- setStandaloneMonthSymbols: (page 447)
- setVeryShortStandaloneMonthSymbols: (page 450)

Declared In
NSDateFormatter.h

setShortStandaloneQuarterSymbols:

Sets the short standalone quarter symbols for the receiver.

```
- (void)setShortStandaloneQuarterSymbols:(NSArray *)array
```

Parameters

array
An array of NSString objects that specify the short standalone quarter symbols for the receiver.

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

See Also
- shortStandaloneQuarterSymbols (page 453)
- setQuarterSymbols: (page 444)
- setShortQuarterSymbols: (page 445)
- setStandaloneQuarterSymbols: (page 447)

Declared In
NSDateFormatter.h

setShortStandaloneWeekdaySymbols:

Sets the short standalone weekday symbols for the receiver.

```
- (void)setShortStandaloneWeekdaySymbols:(NSArray *)array
```

Parameters

array
An array of NSString objects that specify the short standalone weekday symbols for the receiver.

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

See Also
- shortStandaloneWeekdaySymbols (page 454)
- setWeekdaySymbols: (page 451)
- setShortWeekdaySymbols: (page 447)
- setVeryShortWeekdaySymbols: (page 451)
- setStandaloneWeekdaySymbols: (page 448)
- setVeryShortStandaloneWeekdaySymbols: (page 450)

Declared In
NSDateFormatter.h

setShortWeekdaySymbols:

Sets the short weekday symbols for the receiver.

```
- (void)setShortWeekdaySymbols:(NSArray *)array
```

Parameters

array

An array of `NSString` objects that specify the short weekday symbols for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `shortWeekdaySymbols` (page 454)
- `setWeekdaySymbols:` (page 451)
- `setVeryShortWeekdaySymbols:` (page 451)
- `setStandaloneWeekdaySymbols:` (page 448)
- `setShortStandaloneWeekdaySymbols:` (page 446)
- `setVeryShortStandaloneWeekdaySymbols:` (page 450)

Declared In
`NSDateFormatter.h`

setStandaloneMonthSymbols:

Sets the standalone month symbols for the receiver.

```
- (void)setStandaloneMonthSymbols:(NSArray *)array
```

Parameters

array

An array of `NSString` objects that specify the standalone month symbols for the receiver.

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

See Also
- `standaloneMonthSymbols` (page 455)
- `setMonthSymbols:` (page 443)
- `setShortMonthSymbols:` (page 444)
- `setVeryShortMonthSymbols:` (page 450)
- `setShortStandaloneMonthSymbols:` (page 445)
- `setVeryShortStandaloneMonthSymbols:` (page 450)

Declared In
`NSDateFormatter.h`

setStandaloneQuarterSymbols:

Sets the standalone quarter symbols for the receiver.

```
- (void)setStandaloneQuarterSymbols:(NSArray *)array
```

Parameters

array

An array of `NSString` objects that specify the standalone quarter symbols for the receiver.

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

See Also
- `setStandaloneQuarterSymbols:` (page 447)
- `setQuarterSymbols:` (page 444)
- `setShortQuarterSymbols:` (page 445)
- `setShortStandaloneQuarterSymbols:` (page 446)

Declared In
`NSDateFormatter.h`

setStandaloneWeekdaySymbols:

Sets the standalone weekday symbols for the receiver.

```
- (void)setStandaloneWeekdaySymbols:(NSArray *)array
```

Parameters

array

An array of `NSString` objects that specify the standalone weekday symbols for the receiver.

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

See Also
- `standaloneWeekdaySymbols` (page 455)
- `setWeekdaySymbols:` (page 451)
- `setShortWeekdaySymbols:` (page 447)
- `setVeryShortWeekdaySymbols:` (page 451)
- `setShortStandaloneWeekdaySymbols:` (page 446)
- `setVeryShortStandaloneWeekdaySymbols:` (page 450)

Declared In
`NSDateFormatter.h`

setTimeStyle:

Sets the time style of the receiver.

```
- (void)setTimeStyle:(NSDateFormatterStyle)style
```

Parameters

style

The time style for the receiver. For possible values, see NSDateFormatterStyle (page 460).

Availability
Available in Mac OS X v10.4 and later.

See Also
- `timeStyle` (page 456)

Related Sample Code
DatePicker
Mountains
NSOperationSample
Reminders

Declared In
`NSDateFormatter.h`

setTimeZone:

Sets the time zone for the receiver.

```
- (void)setTimeZone:(NSTimeZone *)tz
```

Parameters

tz
The time zone for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `timeZone` (page 457)

Declared In
`NSDateFormatter.h`

setTwoDigitStartDate:

Sets the two-digit start date for the receiver.

```
- (void)setTwoDigitStartDate:(NSDate *)date
```

Parameters

date
The earliest date that can be denoted by a two-digit year specifier.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `twoDigitStartDate` (page 457)

Declared In
`NSDateFormatter.h`

setVeryShortMonthSymbols:

Sets the very short month symbols for the receiver.

```
- (void)setVeryShortMonthSymbols:(NSArray *)array
```

Parameters

array

An array of `NSString` objects that specify the very short month symbols for the receiver.

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

See Also
- `veryShortMonthSymbols` (page 458)
- `setMonthSymbols:` (page 443)
- `setShortMonthSymbols:` (page 444)
- `setStandaloneMonthSymbols:` (page 447)
- `setShortStandaloneMonthSymbols:` (page 445)
- `setVeryShortStandaloneMonthSymbols:` (page 450)

Declared In
`NSDateFormatter.h`

setVeryShortStandaloneMonthSymbols:

Sets the very short standalone month symbols for the receiver.

```
- (void)setVeryShortStandaloneMonthSymbols:(NSArray *)array
```

Parameters

array

An array of `NSString` objects that specify the very short standalone month symbols for the receiver.

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

See Also
- `veryShortStandaloneMonthSymbols` (page 458)
- `setMonthSymbols:` (page 443)
- `setShortMonthSymbols:` (page 444)
- `setVeryShortMonthSymbols:` (page 450)
- `setStandaloneMonthSymbols:` (page 447)
- `setShortStandaloneMonthSymbols:` (page 445)

Declared In
`NSDateFormatter.h`

setVeryShortStandaloneWeekdaySymbols:

Sets the very short standalone weekday symbols for the receiver.

```
- (void)setVeryShortStandaloneWeekdaySymbols:(NSArray *)array
```

Parameters

array

An array of `NSString` objects that specify the very short standalone weekday symbols for the receiver.

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

See Also
- `veryShortStandaloneWeekdaySymbols` (page 458)
- `setWeekdaySymbols:` (page 451)
- `setShortWeekdaySymbols:` (page 447)
- `setVeryShortWeekdaySymbols:` (page 451)
- `setStandaloneWeekdaySymbols:` (page 448)
- `setShortStandaloneWeekdaySymbols:` (page 446)

Declared In
`NSDateFormatter.h`

setVeryShortWeekdaySymbols:

Sets the vert short weekday symbols for the receiver

```
- (void)setVeryShortWeekdaySymbols:(NSArray *)array
```

Parameters

array

An array of `NSString` objects that specify the very short weekday symbols for the receiver.

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

See Also
- `veryShortWeekdaySymbols` (page 459)
- `setWeekdaySymbols:` (page 451)
- `setShortWeekdaySymbols:` (page 447)
- `setStandaloneWeekdaySymbols:` (page 448)
- `setShortStandaloneWeekdaySymbols:` (page 446)
- `setVeryShortStandaloneWeekdaySymbols:` (page 450)

Declared In
`NSDateFormatter.h`

setWeekdaySymbols:

Sets the weekday symbols for the receiver.

```
- (void)setWeekdaySymbols:(NSArray *)array
```

Parameters

array

An array of `NSString` objects that specify the weekday symbols for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `weekdaySymbols` (page 459)
- `setShortWeekdaySymbols:` (page 447)
- `setVeryShortWeekdaySymbols:` (page 451)
- `setStandaloneWeekdaySymbols:` (page 448)
- `setShortStandaloneWeekdaySymbols:` (page 446)
- `setVeryShortStandaloneWeekdaySymbols:` (page 450)

Declared In
`NSDateFormatter.h`

shortMonthSymbols

Returns the array of short month symbols for the receiver.

```
- (NSArray *)shortMonthSymbols
```

Return Value
An array containing `NSString` objects representing the short month symbols for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setShortMonthSymbols:` (page 444)
- `monthSymbols` (page 437)
- `veryShortMonthSymbols` (page 458)
- `standaloneMonthSymbols` (page 455)
- `shortStandaloneMonthSymbols` (page 453)
- `veryShortStandaloneMonthSymbols` (page 458)

Declared In
`NSDateFormatter.h`

shortQuarterSymbols

Returns the short quarter symbols for the receiver.

```
- (NSArray *)shortQuarterSymbols
```

Return Value
An array containing `NSString` objects representing the short quarter symbols for the receiver.

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

See Also
- `setShortQuarterSymbols:` (page 445)
- `quarterSymbols` (page 438)
- `standaloneQuarterSymbols` (page 455)
- `shortStandaloneQuarterSymbols` (page 453)

Declared In
`NSDateFormatter.h`

shortStandaloneMonthSymbols

Returns the short standalone month symbols for the receiver.

```
- (NSArray *)shortStandaloneMonthSymbols
```

Return Value
An array of `NSString` objects that specify the short standalone month symbols for the receiver.

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

See Also
- `setShortStandaloneMonthSymbols:` (page 445)
- `monthSymbols` (page 437)
- `shortMonthSymbols` (page 452)
- `veryShortMonthSymbols` (page 458)
- `standaloneMonthSymbols` (page 455)
- `veryShortStandaloneMonthSymbols` (page 458)

Declared In
`NSDateFormatter.h`

shortStandaloneQuarterSymbols

Returns the short standalone quarter symbols for the receiver.

```
- (NSArray *)shortStandaloneQuarterSymbols
```

Return Value
An array containing `NSString` objects representing the short standalone quarter symbols for the receiver.

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

See Also
- `setShortStandaloneQuarterSymbols:` (page 446)
- `quarterSymbols` (page 438)
- `shortQuarterSymbols` (page 452)

- standaloneQuarterSymbols (page 455)

Declared In
NSDateFormatter.h

shortStandaloneWeekdaySymbols

Returns the array of short standalone weekday symbols for the receiver.

```
- (NSArray *)shortStandaloneWeekdaySymbols
```

Return Value
An array of NSString objects that specify the short standalone weekday symbols for the receiver.

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

See Also
- setShortStandaloneWeekdaySymbols: (page 446)
- weekdaySymbols (page 459)
- shortWeekdaySymbols (page 454)
- veryShortWeekdaySymbols (page 459)
- standaloneWeekdaySymbols (page 455)
- veryShortStandaloneWeekdaySymbols (page 458)

Declared In
NSDateFormatter.h

shortWeekdaySymbols

Returns the array of short weekday symbols for the receiver.

```
- (NSArray *)shortWeekdaySymbols
```

Return Value
An array of NSString objects that specify the short weekday symbols for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setShortWeekdaySymbols: (page 447)
- weekdaySymbols (page 459)
- veryShortWeekdaySymbols (page 459)
- standaloneWeekdaySymbols (page 455)
- shortStandaloneWeekdaySymbols (page 454)
- veryShortStandaloneWeekdaySymbols (page 458)

Declared In
NSDateFormatter.h

standaloneMonthSymbols

Returns the standalone month symbols for the receiver.

```
- (NSArray *)standaloneMonthSymbols
```

Return Value
An array of `NSString` objects that specify the standalone month symbols for the receiver.

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

See Also
- `monthSymbols` (page 437)
- `setStandaloneMonthSymbols:` (page 447)
- `shortMonthSymbols` (page 452)
- `veryShortMonthSymbols` (page 458)
- `shortStandaloneMonthSymbols` (page 453)
- `veryShortStandaloneMonthSymbols` (page 458)

Declared In
`NSDateFormatter.h`

standaloneQuarterSymbols

Returns the standalone quarter symbols for the receiver.

```
- (NSArray *)standaloneQuarterSymbols
```

Return Value
An array containing `NSString` objects representing the standalone quarter symbols for the receiver.

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

See Also
- `setStandaloneQuarterSymbols:` (page 447)
- `quarterSymbols` (page 438)
- `shortQuarterSymbols` (page 452)
- `shortStandaloneQuarterSymbols` (page 453)

Declared In
`NSDateFormatter.h`

standaloneWeekdaySymbols

Returns the array of standalone weekday symbols for the receiver.

```
- (NSArray *)standaloneWeekdaySymbols
```

Return Value
An array of `NSString` objects that specify the standalone weekday symbols for the receiver.

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

See Also
- `setStandaloneWeekdaySymbols:` (page 448)
- `weekdaySymbols` (page 459)
- `shortWeekdaySymbols` (page 454)
- `veryShortWeekdaySymbols` (page 459)
- `shortStandaloneWeekdaySymbols` (page 454)
- `veryShortStandaloneWeekdaySymbols` (page 458)

Declared In
`NSDateFormatter.h`

stringFromDate:

Returns a string representation of a given date formatted using the receiver's current settings.

```
- (NSString *)stringFromDate:(NSDate *)date
```

Parameters

date
The date to format.

Return Value
A string representation of *date* formatted using the receiver's current settings.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `dateFromString:` (page 431)

Related Sample Code
DatePicker
iSpend
Mountains
NSOperationSample
Reminders

Declared In
`NSDateFormatter.h`

timeStyle

Returns the time style of the receiver.

```
- (NSDateFormatterStyle)timeStyle
```

Return Value
The time style of the receiver. For possible values, see NSDateFormatterStyle (page 460).

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setTimeStyle:` (page 448)

Declared In
`NSDateFormatter.h`

timeZone

Returns the time zone for the receiver.

```
- (NSTimeZone *)timeZone
```

Return Value
The time zone for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setTimeZone:` (page 449)

Declared In
`NSDateFormatter.h`

twoDigitStartDate

Returns the earliest date that can be denoted by a two-digit year specifier.

```
- (NSDate *)twoDigitStartDate
```

Return Value
The earliest date that can be denoted by a two-digit year specifier.

Discussion
If the two-digit start date is set to January 6, 1976, then “January 1, 76” is interpreted as New Year's Day in 2076, whereas “February 14, 76” is interpreted as Valentine's Day in 1976.

The default date is December 31, 1949.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setTwoDigitStartDate:` (page 449)

Declared In
`NSDateFormatter.h`

verySymbols

Returns the very short month symbols for the receiver.

```
- (NSArray *)verySymbols
```

Return Value
An array of `NSString` objects that specify the very short month symbols for the receiver.

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

See Also
- `setVerySymbols:` (page 450)
- `monthSymbols` (page 437)
- `shortMonthSymbols` (page 452)
- `standaloneMonthSymbols` (page 455)
- `shortStandaloneMonthSymbols` (page 453)
- `verySymbols` (page 458)

Declared In
`NSDateFormatter.h`

verySymbols

Returns the very short month symbols for the receiver.

```
- (NSArray *)verySymbols
```

Return Value
An array of `NSString` objects that specify the very short standalone month symbols for the receiver.

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

See Also
- `setVerySymbols:` (page 450)
- `monthSymbols` (page 437)
- `shortMonthSymbols` (page 452)
- `verySymbols` (page 458)
- `standaloneMonthSymbols` (page 455)
- `shortStandaloneMonthSymbols` (page 453)

Declared In
`NSDateFormatter.h`

verySymbols

Returns the array of very short standalone weekday symbols for the receiver.

```
- (NSArray *)verySymbols
```

Return Value
An array of `NSString` objects that specify the very short standalone weekday symbols for the receiver.

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

See Also
- `setShortStandaloneWeekdaySymbols:` (page 446)
- `weekdaySymbols` (page 459)
- `shortWeekdaySymbols` (page 454)
- `veryShortWeekdaySymbols` (page 459)
- `standaloneWeekdaySymbols` (page 455)
- `shortStandaloneWeekdaySymbols` (page 454)

Declared In
`NSDateFormatter.h`

veryShortWeekdaySymbols

Returns the array of very short weekday symbols for the receiver.

```
- (NSArray *)veryShortWeekdaySymbols
```

Return Value
An array of `NSString` objects that specify the very short weekday symbols for the receiver.

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

See Also
- `setVeryShortWeekdaySymbols:` (page 451)
- `weekdaySymbols` (page 459)
- `shortWeekdaySymbols` (page 454)
- `standaloneWeekdaySymbols` (page 455)
- `shortStandaloneWeekdaySymbols` (page 454)
- `veryShortStandaloneWeekdaySymbols` (page 458)

Declared In
`NSDateFormatter.h`

weekdaySymbols

Returns the array of weekday symbols for the receiver.

```
- (NSArray *)weekdaySymbols
```

Return Value
An array of `NSString` objects that specify the weekday symbols for the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setWeekdaySymbols:` (page 451)
- `shortWeekdaySymbols` (page 454)
- `veryShortWeekdaySymbols` (page 459)
- `standaloneWeekdaySymbols` (page 455)
- `shortStandaloneWeekdaySymbols` (page 454)
- `veryShortStandaloneWeekdaySymbols` (page 458)

Declared In
`NSDateFormatter.h`

Constants

NSDateFormatterStyle

The following constants specify predefined date and time format styles.

```
typedef enum {
    NSDateFormatterNoStyle     = kCFDateFormatterNoStyle,
    NSDateFormatterShortStyle  = kCFDateFormatterShortStyle,
    NSDateFormatterMediumStyle = kCFDateFormatterMediumStyle,
    NSDateFormatterLongStyle   = kCFDateFormatterLongStyle,
    NSDateFormatterFullStyle   = kCFDateFormatterFullStyle
} NSDateFormatterStyle;
```

Constants

`NSDateFormatterNoStyle`

Specifies no style.

Equal to `kCFDateFormatterNoStyle`.

Available in Mac OS X v10.4 and later.

Declared in `NSDateFormatter.h`.

`NSDateFormatterShortStyle`

Specifies a short style, typically numeric only, such as “11/23/37” or “3:30pm”.

Equal to `kCFDateFormatterShortStyle`.

Available in Mac OS X v10.4 and later.

Declared in `NSDateFormatter.h`.

`NSDateFormatterMediumStyle`

Specifies a medium style, typically with abbreviated text, such as “Nov 23, 1937”.

Equal to `kCFDateFormatterMediumStyle`.

Available in Mac OS X v10.4 and later.

Declared in `NSDateFormatter.h`.

`NSDateFormatterLongStyle`
Specifies a long style, typically with full text, such as “November 23, 1937” or “3:30:32pm”.

Equal to `kCFDateFormatterLongStyle`.

Available in Mac OS X v10.4 and later.

Declared in `NSDateFormatter.h`.

`NSDateFormatterFullStyle`
Specifies a full style with complete details, such as “Tuesday, April 12, 1952 AD” or “3:30:42pm PST”.

Equal to `kCFDateFormatterFullStyle`.

Available in Mac OS X v10.4 and later.

Declared in `NSDateFormatter.h`.

Discussion
The format for these date and time styles is not exact because they depend on the locale, user preference settings, and the operating system version. Do not use these constants if you want an exact format.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSDateFormatter.h`

NSDateFormatterBehavior

Constants that specify the behavior `NSDateFormatter` should exhibit.

```
typedef enum {
    NSDateFormatterBehaviorDefault = 0,
    NSDateFormatterBehavior10_0    = 1000,
    NSDateFormatterBehavior10_4    = 1040,
} NSDateFormatterBehavior;
```

Constants
`NSDateFormatterBehaviorDefault`
Specifies default formatting behavior.

Available in Mac OS X v10.4 and later.

Declared in `NSDateFormatter.h`.

`NSDateFormatterBehavior10_0`
Specifies formatting behavior equivalent to that in Mac OS X 10.0.

Available in Mac OS X v10.4 and later.

Declared in `NSDateFormatter.h`.

`NSDateFormatterBehavior10_4`
Specifies formatting behavior equivalent for Mac OS X 10.4.

Available in Mac OS X v10.4 and later.

Declared in `NSDateFormatter.h`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSDateFormatter.h`

NSDecimalNumber Class Reference

Inherits from	NSNumber : NSValue : NSObject
Conforms to	NSCoding (NSValue) NSCopying (NSValue) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSDecimalNumber.h
Companion guide	Number and Value Programming Topics for Cocoa
Related sample code	BindingsJoystick Calculator Core Data HTML Store

Overview

`NSDecimalNumber`, an immutable subclass of `NSNumber`, provides an object-oriented wrapper for doing base-10 arithmetic. An instance can represent any number that can be expressed as `mantissa x 10^exponent` where mantissa is a decimal integer up to 38 digits long, and exponent is an integer from –128 through 127.

Tasks

Creating a Decimal Number

`+ decimalNumberWithDecimal:` (page 466)
Creates and returns an `NSDecimalNumber` object equivalent to a given `NSDecimal` structure.

`+ decimalNumberWithMantissa:exponent:isNegative:` (page 466)
Creates and returns an `NSDecimalNumber` object equivalent to the number specified by the arguments.

`+ decimalNumberWithString:` (page 467)
Creates and returns an `NSDecimalNumber` object whose value is equivalent to that in a given numeric string.

+ `decimalNumberWithString:locale:` (page 468)
  Creates and returns an `NSDecimalNumber` object whose value is equivalent to that in a given numeric string, interpreted using a given locale.

+ `one` (page 470)
  Returns an `NSDecimalNumber` object equivalent to the number 1.0.

+ `zero` (page 471)
  Returns an `NSDecimalNumber` object equivalent to the number 0.0.

+ `notANumber` (page 470)
  Returns an `NSDecimalNumber` object that specifies no number.

Initializing a Decimal Number

- `initWithDecimal:` (page 478)
  Returns an `NSDecimalNumber` object initialized to represent a given decimal.

- `initWithMantissa:exponent:isNegative:` (page 478)
  Returns an `NSDecimalNumber` object initialized using the given mantissa, exponent, and sign.

- `initWithString:` (page 479)
  Returns an `NSDecimalNumber` object initialized so that its value is equivalent to that in a given numeric string.

- `initWithString:locale:` (page 480)
  Returns an `NSDecimalNumber` object initialized so that its value is equivalent to that in a given numeric string, interpreted using a given locale.

Performing Arithmetic

- `decimalNumberByAdding:` (page 472)
  Returns a new `NSDecimalNumber` object whose value is the sum of the receiver and another given `NSDecimalNumber` object.

- `decimalNumberBySubtracting:` (page 476)
  Returns a new `NSDecimalNumber` object whose value is that of another given `NSDecimalNumber` object subtracted from the value of the receiver.

- `decimalNumberByMultiplyingBy:` (page 473)
  Returns a new `NSDecimalNumber` object whose value is the value of the receiver multiplied by that of another given `NSDecimalNumber` object.

- `decimalNumberByDividingBy:` (page 473)
  Returns a new `NSDecimalNumber` object whose value is the value of the receiver divided by that of another given `NSDecimalNumber` object.

- `decimalNumberByRaisingToPower:` (page 475)
  Returns a new `NSDecimalNumber` object whose value is the value of the receiver raised to a given power.

- `decimalNumberByMultiplyingByPowerOf10:` (page 474)
  Multiplies the receiver by 10^*power* and returns the product, a newly created `NSDecimalNumber` object.

- `decimalNumberByAdding:withBehavior:` (page 472)
  Adds *decimalNumber* to the receiver and returns the sum, a newly created `NSDecimalNumber` object.
- `decimalNumberBySubtracting:withBehavior:` (page 477)
  Subtracts *decimalNumber* from the receiver and returns the difference, a newly created `NSDecimalNumber` object.
- `decimalNumberByMultiplyingBy:withBehavior:` (page 474)
  Multiplies the receiver by *decimalNumber* and returns the product, a newly created `NSDecimalNumber` object.
- `decimalNumberByDividingBy:withBehavior:` (page 473)
  Divides the receiver by *decimalNumber* and returns the quotient, a newly created `NSDecimalNumber` object.
- `decimalNumberByRaisingToPower:withBehavior:` (page 475)
  Raises the receiver to *power* and returns the result, a newly created `NSDecimalNumber` object.
- `decimalNumberByMultiplyingByPowerOf10:withBehavior:` (page 475)
  Multiplies the receiver by 10^*power* and returns the product, a newly created `NSDecimalNumber` object.

Rounding Off

- `decimalNumberByRoundingAccordingToBehavior:` (page 476)
  Rounds the receiver off in the way specified by *behavior* and returns the result, a newly created `NSDecimalNumber` object.

Accessing the Value

- `decimalValue` (page 477)
  Returns the receiver's value, expressed as an `NSDecimal` structure.
- `doubleValue` (page 478)
  Returns the approximate value of the receiver as a `double`.
- `descriptionWithLocale:` (page 477)
  Returns a string, specified according to a given locale, that represents the contents of the receiver.
- `objCType` (page 480)
  Returns a C string containing the Objective-C type of the data contained in the receiver, which for an `NSDecimalNumber` object is always "d" (for double).

Managing Behavior

+ `defaultBehavior` (page 468)
  Returns the way arithmetic methods, like `decimalNumberByAdding:` (page 472), round off and handle error conditions.
+ `setDefaultBehavior:` (page 470)
  Specifies the way that arithmetic methods, like `decimalNumberByAdding:` (page 472), round off and handle error conditions.

Comparing Decimal Numbers

`- compare:` (page 471)
Returns an `NSComparisonResult` value that indicates the numerical ordering of the receiver and another given `NSDecimalNumber` object.

Getting Maximum and Minimum Possible Values

`+ maximumDecimalNumber` (page 469)
Returns the largest possible value of an `NSDecimalNumber` object.

`+ minimumDecimalNumber` (page 469)
Returns the smallest possible value of an `NSDecimalNumber` object.

Class Methods

decimalNumberWithDecimal:

Creates and returns an `NSDecimalNumber` object equivalent to a given `NSDecimal` structure.

```
+ (NSDecimalNumber *)decimalNumberWithDecimal:(NSDecimal)decimal
```

Parameters

decimal
An `NSDecimal` structure that specifies the value for the new decimal number object.

Return Value
An `NSDecimalNumber` object equivalent to *decimal*.

Discussion
You can initialize *decimal* programmatically or generate it using the `NSScanner` method, `scanDecimal:` (page 1349)

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

Declared In
`NSDecimalNumber.h`

decimalNumberWithMantissa:exponent:isNegative:

Creates and returns an `NSDecimalNumber` object equivalent to the number specified by the arguments.

```
+ (NSDecimalNumber *)decimalNumberWithMantissa:(unsigned long long)mantissa
    exponent:(short)exponent isNegative:(BOOL)isNegative
```

Parameters

mantissa
The mantissa for the new decimal number object.

exponent
The exponent for the new decimal number object.

isNegative
A Boolean value that specifies whether the sign of the number is negative.

Discussion
The arguments express a number in a kind of scientific notation that requires the mantissa to be an integer. So, for example, if the number to be represented is `-12.345`, it is expressed as `12345x10^-3`—*mantissa* is `12345`; *exponent* is `-3`; and *isNegative* is `YES`, as illustrated by the following example.

```
NSDecimalNumber *number = [NSDecimalNumber decimalNumberWithMantissa:12345
                                                            exponent:-3
                                                            isNegative:YES];
```

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

Declared In
`NSDecimalNumber.h`

decimalNumberWithString:

Creates and returns an `NSDecimalNumber` object whose value is equivalent to that in a given numeric string.

```
+ (NSDecimalNumber *)decimalNumberWithString:(NSString *)numericString
```

Parameters
numericString
A numeric string.

Besides digits, *numericString* can include an initial "+" or "–"; a single "E" or "e", to indicate the exponent of a number in scientific notation; and a single `NSDecimalSeparator` to divide the fractional from the integral part of the number.

Return Value
An `NSDecimalNumber` object whose value is equivalent to *numericString*.

Discussion
Whether the `NSDecimalSeparator` is a period (as is used, for example, in the United States) or a comma (as is used, for example, in France) depends on the default locale.

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

See Also
`+ decimalNumberWithString:locale:` (page 468)

Related Sample Code
Calculator
Core Data HTML Store

Declared In
`NSDecimalNumber.h`

decimalNumberWithString:locale:

Creates and returns an `NSDecimalNumber` object whose value is equivalent to that in a given numeric string, interpreted using a given locale.

```
+ (NSDecimalNumber *)decimalNumberWithString:(NSString *)numericString
    locale:(NSDictionary *)locale
```

Parameters

numericString

A numeric string.

Besides digits, *numericString* can include an initial "+" or "–"; a single "E" or "e", to indicate the exponent of a number in scientific notation; and a single `NSDecimalSeparator` to divide the fractional from the integral part of the number.

locale

A dictionary that defines the locale (specifically the `NSDecimalSeparator`) to use to interpret the number in *numericString*.

Return Value

An `NSDecimalNumber` object whose value is equivalent to *numericString*.

Discussion

The *locale* parameter determines whether the `NSDecimalSeparator` is a period (as is used, for example, in the United States) or a comma (as is used, for example, in France).

The following strings show examples of acceptable values for *numericString*:

"2500.6" (or "2500,6", depending on locale)
"–2500.6" (or "–2500.6")
"–2.5006e3" (or "–2,5006e3")
"–2.5006E3" (or "–2,5006E3")

The following strings are unacceptable:

"2,500.6"
"2500 3/5"
"2.5006x10e3"
"two thousand five hundred and six tenths"

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ decimalNumberWithString:` (page 467)

Declared In

`NSDecimalNumber.h`

defaultBehavior

Returns the way arithmetic methods, like `decimalNumberByAdding:` (page 472), round off and handle error conditions.

```
+ (id < NSDecimalNumberBehaviors >)defaultBehavior
```

Discussion
By default, the arithmetic methods use the `NSRoundPlain` behavior; that is, the methods round to the closest possible return value. The methods assume your need for precision does not exceed 38 significant digits and raise exceptions when they try to divide by 0 or produce a number too big or too small to be represented.

If this default behavior doesn't suit your application, you should use methods that let you specify the behavior, like `decimalNumberByAdding:withBehavior:` (page 472). If you find yourself using a particular behavior consistently, you can specify a different default behavior with `setDefaultBehavior:` (page 470).

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

Declared In
`NSDecimalNumber.h`

maximumDecimalNumber

Returns the largest possible value of an `NSDecimalNumber` object.

```
+ (NSDecimalNumber *)maximumDecimalNumber
```

Return Value
The largest possible value of an `NSDecimalNumber` object.

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

See Also
`+ minimumDecimalNumber` (page 469)

Declared In
`NSDecimalNumber.h`

minimumDecimalNumber

Returns the smallest possible value of an `NSDecimalNumber` object.

```
+ (NSDecimalNumber *)minimumDecimalNumber
```

Return Value
The smallest possible value of an `NSDecimalNumber` object.

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

See Also
`+ maximumDecimalNumber` (page 469)

Declared In
`NSDecimalNumber.h`

notANumber

Returns an `NSDecimalNumber` object that specifies no number.

```
+ (NSDecimalNumber *)notANumber
```

Return Value
An `NSDecimalNumber` object that specifies no number.

Discussion
Any arithmetic method receiving `notANumber` as an argument returns `notANumber`.

This value can be a useful way of handling non-numeric data in an input file. This method can also be a useful response to calculation errors. For more information on calculation errors, see the `exceptionDuringOperation:error:leftOperand:rightOperand:` (page 2044) method description in the `NSDecimalNumberBehaviors` protocol specification.

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

Related Sample Code
Calculator

Declared In
`NSDecimalNumber.h`

one

Returns an `NSDecimalNumber` object equivalent to the number 1.0.

```
+ (NSDecimalNumber *)one
```

Return Value
An `NSDecimalNumber` object equivalent to the number 1.0.

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

See Also
`+ zero` (page 471)

Declared In
`NSDecimalNumber.h`

setDefaultBehavior:

Specifies the way that arithmetic methods, like `decimalNumberByAdding:` (page 472), round off and handle error conditions.

```
+ (void)setDefaultBehavior:(id < NSDecimalNumberBehaviors >)behavior
```

Discussion
behavior must conform to the `NSDecimalNumberBehaviors` protocol.

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

Declared In
`NSDecimalNumber.h`

zero

Returns an `NSDecimalNumber` object equivalent to the number 0.0.

```
+ (NSDecimalNumber *)zero
```

Return Value
An `NSDecimalNumber` object equivalent to the number 0.0.

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

See Also
`+ one` (page 470)

Related Sample Code
BindingsJoystick
Calculator

Declared In
`NSDecimalNumber.h`

Instance Methods

compare:

Returns an `NSComparisonResult` value that indicates the numerical ordering of the receiver and another given `NSDecimalNumber` object.

```
- (NSComparisonResult)compare:(NSNumber *)decimalNumber
```

Parameters

`decimalNumber`

The number with which to compare the receiver.

This value must not be `nil`. If this value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

Return Value
`NSOrderedAscending` if the value of *`decimalNumber`* is greater than the receiver; `NSOrderedSame` if they're equal; and `NSOrderedDescending` if the value of *`decimalNumber`* is less than the receiver.

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

Declared In
NSDecimalNumber.h

decimalNumberByAdding:

Returns a new NSDecimalNumber object whose value is the sum of the receiver and another given NSDecimalNumber object.

```
- (NSDecimalNumber *)decimalNumberByAdding:(NSDecimalNumber *)decimalNumber
```

Parameters

decimalNumber
The number to add to the receiver.

Return Value
A new NSDecimalNumber object whose value is the sum of the receiver and *decimalNumber*.

Discussion
This method uses the default behavior when handling calculation errors and rounding.

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

See Also
- decimalNumberByAdding:withBehavior: (page 472)
+ defaultBehavior (page 468)

Related Sample Code
Calculator

Declared In
NSDecimalNumber.h

decimalNumberByAdding:withBehavior:

Adds *decimalNumber* to the receiver and returns the sum, a newly created NSDecimalNumber object.

```
- (NSDecimalNumber *)decimalNumberByAdding:(NSDecimalNumber *)decimalNumber
    withBehavior:(id < NSDecimalNumberBehaviors >)behavior
```

Discussion
behavior specifies the handling of calculation errors and rounding.

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

Declared In
NSDecimalNumber.h

decimalNumberByDividingBy:

Returns a new `NSDecimalNumber` object whose value is the value of the receiver divided by that of another given `NSDecimalNumber` object.

```
- (NSDecimalNumber *)decimalNumberByDividingBy:(NSDecimalNumber *)decimalNumber
```

Parameters

decimalNumber

The number by which to divide the receiver.

Return Value

A new `NSDecimalNumber` object whose value is the value of the receiver divided by *decimalNumber*.

Discussion

This method uses the default behavior when handling calculation errors and rounding.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- decimalNumberByDividingBy:withBehavior:` (page 473)

`+ defaultBehavior` (page 468)

Related Sample Code

Calculator

Declared In

`NSDecimalNumber.h`

decimalNumberByDividingBy:withBehavior:

Divides the receiver by *decimalNumber* and returns the quotient, a newly created `NSDecimalNumber` object.

```
- (NSDecimalNumber *)decimalNumberByDividingBy:(NSDecimalNumber *)decimalNumber
    withBehavior:(id < NSDecimalNumberBehaviors >)behavior
```

Discussion

behavior specifies the handling of calculation errors and rounding.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSDecimalNumber.h`

decimalNumberByMultiplyingBy:

Returns a new `NSDecimalNumber` object whose value is the value of the receiver multiplied by that of another given `NSDecimalNumber` object.

```
- (NSDecimalNumber *)decimalNumberByMultiplyingBy:(NSDecimalNumber *)decimalNumber
```

Parameters

decimalNumber

The number by which to multiply the receiver.

Return Value

A new `NSDecimalNumber` object whose value is *decimalNumber* multiplied by the receiver.

Discussion

This method uses the default behavior when handling calculation errors and when rounding.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- decimalNumberByMultiplyingBy:withBehavior:` (page 474)

`+ defaultBehavior` (page 468)

Related Sample Code

Calculator

Declared In

`NSDecimalNumber.h`

decimalNumberByMultiplyingBy:withBehavior:

Multiplies the receiver by *decimalNumber* and returns the product, a newly created `NSDecimalNumber` object.

```
- (NSDecimalNumber *)decimalNumberByMultiplyingBy:(NSDecimalNumber *)decimalNumber
    withBehavior:(id < NSDecimalNumberBehaviors >)behavior
```

Discussion

behavior specifies the handling of calculation errors and rounding.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSDecimalNumber.h`

decimalNumberByMultiplyingByPowerOf10:

Multiplies the receiver by 10^*power* and returns the product, a newly created `NSDecimalNumber` object.

```
- (NSDecimalNumber *)decimalNumberByMultiplyingByPowerOf10:(short)power
```

Discussion

This method uses the default behavior when handling calculation errors and when rounding.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- decimalNumberByMultiplyingByPowerOf10:withBehavior:` (page 475)

+ defaultBehavior (page 468)

Declared In
NSDecimalNumber.h

decimalNumberByMultiplyingByPowerOf10:withBehavior:

Multiplies the receiver by 10^*power* and returns the product, a newly created NSDecimalNumber object.

```
- (NSDecimalNumber *)decimalNumberByMultiplyingByPowerOf10:(short)power
    withBehavior:(id < NSDecimalNumberBehaviors >)behavior
```

Discussion
behavior specifies the handling of calculation errors and rounding.

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

Declared In
NSDecimalNumber.h

decimalNumberByRaisingToPower:

Returns a new NSDecimalNumber object whose value is the value of the receiver raised to a given power.

```
- (NSDecimalNumber *)decimalNumberByRaisingToPower:(NSUInteger)power
```

Parameters

power
The power to which to raise the receiver.

Return Value
A new NSDecimalNumber object whose value is the value of the receiver raised to the power *power*.

Discussion
This method uses the default behavior when handling calculation errors and when rounding.

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

See Also
- decimalNumberByRaisingToPower:withBehavior: (page 475)
+ defaultBehavior (page 468)

Declared In
NSDecimalNumber.h

decimalNumberByRaisingToPower:withBehavior:

Raises the receiver to *power* and returns the result, a newly created NSDecimalNumber object.

```
- (NSDecimalNumber *)decimalNumberByRaisingToPower:(NSUInteger)power withBehavior:(id
    < NSDecimalNumberBehaviors >)behavior
```

Discussion
behavior specifies the handling of calculation errors and rounding.

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

Declared In
NSDecimalNumber.h

decimalNumberByRoundingAccordingToBehavior:

Rounds the receiver off in the way specified by *behavior* and returns the result, a newly created NSDecimalNumber object.

```
- (NSDecimalNumber *)decimalNumberByRoundingAccordingToBehavior:(id <
    NSDecimalNumberBehaviors >)behavior
```

Discussion
For a description of the different ways of rounding, see the roundingMode (page 1109) method in the NSDecimalNumberBehaviors protocol specification.

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

Declared In
NSDecimalNumber.h

decimalNumberBySubtracting:

Returns a new NSDecimalNumber object whose value is that of another given NSDecimalNumber object subtracted from the value of the receiver.

```
- (NSDecimalNumber *)decimalNumberBySubtracting:(NSDecimalNumber *)decimalNumber
```

Parameters
decimalNumber
The number to subtract from the receiver.

Return Value
A new NSDecimalNumber object whose value is *decimalNumber* subtracted from the receiver.

Discussion
This method uses the default behavior when handling calculation errors and when rounding.

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

See Also
- decimalNumberBySubtracting:withBehavior: (page 477)
+ defaultBehavior (page 468)

Related Sample Code
Calculator

Declared In
NSDecimalNumber.h

decimalNumberBySubtracting:withBehavior:

Subtracts *decimalNumber* from the receiver and returns the difference, a newly created NSDecimalNumber object.

```
- (NSDecimalNumber *)decimalNumberBySubtracting:(NSDecimalNumber *)decimalNumber
    withBehavior:(id < NSDecimalNumberBehaviors >)behavior
```

Discussion
behavior specifies the handling of calculation errors and rounding.

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

Declared In
NSDecimalNumber.h

decimalValue

Returns the receiver's value, expressed as an NSDecimal structure.

```
- (NSDecimal)decimalValue
```

Return Value
The receiver's value, expressed as an NSDecimal structure.

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

Declared In
NSDecimalNumber.h

descriptionWithLocale:

Returns a string, specified according to a given locale, that represents the contents of the receiver.

```
- (NSString *)descriptionWithLocale:(NSDictionary *)locale
```

Parameters
locale
A dictionary that defines the locale (specifically the NSDecimalSeparator) to use to generate the returned string.

Return Value
A string that represents the contents of the receiver, according to *locale*.

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

Declared In
`NSDecimalNumber.h`

doubleValue

Returns the approximate value of the receiver as a `double`.

```
- (double)doubleValue
```

Return Value
The approximate value of the receiver as a `double`.

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

Declared In
`NSDecimalNumber.h`

initWithDecimal:

Returns an `NSDecimalNumber` object initialized to represent a given decimal.

```
- (id)initWithDecimal:(NSDecimal)decimal
```

Parameters
decimal
The value of the new object.

Return Value
An `NSDecimalNumber` object initialized to represent *decimal*.

Discussion
This method is the designated initializer for `NSDecimalNumber`.

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

Declared In
`NSDecimalNumber.h`

initWithMantissa:exponent:isNegative:

Returns an `NSDecimalNumber` object initialized using the given mantissa, exponent, and sign.

```
- (id)initWithMantissa:(unsigned long long)mantissa exponent:(short)exponent
    isNegative:(BOOL)flag
```

Parameters

mantissa

The mantissa for the new decimal number object.

exponent

The exponent for the new decimal number object.

flag

A Boolean value that specifies whether the sign of the number is negative.

Return Value

An `NSDecimalNumber` object initialized using the given mantissa, exponent, and sign.

Discussion

The arguments express a number in a type of scientific notation that requires the mantissa to be an integer. So, for example, if the number to be represented is 1.23, it is expressed as 123x10^–2—*mantissa* is 123; *exponent* is –2; and *isNegative*, which refers to the sign of the mantissa, is `NO`.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ decimalNumberWithMantissa:exponent:isNegative:` (page 466)

Declared In

`NSDecimalNumber.h`

initWithString:

Returns an `NSDecimalNumber` object initialized so that its value is equivalent to that in a given numeric string.

```
- (id)initWithString:(NSString *)numericString
```

Parameters

numericString

A numeric string.

Besides digits, *numericString* can include an initial "+" or "–"; a single "E" or "e", to indicate the exponent of a number in scientific notation; and a single `NSDecimalSeparator` to divide the fractional from the integral part of the number. For a listing of acceptable and unacceptable strings, see the class method `decimalNumberWithString:locale:` (page 468).

Return Value

An `NSDecimalNumber` object initialized so that its value is equivalent to that in *numericString*.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSDecimalNumber.h`

initWithString:locale:

Returns an `NSDecimalNumber` object initialized so that its value is equivalent to that in a given numeric string, interpreted using a given locale.

```
- (id)initWithString:(NSString *)numericString locale:(NSDictionary *)locale
```

Parameters

numericString

A numeric string.

Besides digits, *numericString* can include an initial "+" or "–"; a single "E" or "e", to indicate the exponent of a number in scientific notation; and a single `NSDecimalSeparator` to divide the fractional from the integral part of the number.

locale

A dictionary that defines the locale (specifically the `NSDecimalSeparator`) to use to interpret the number in *numericString*.

Return Value

An `NSDecimalNumber` object initialized so that its value is equivalent to that in *numericString*, interpreted using *locale*.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ decimalNumberWithString:locale:` (page 468)

Declared In

`NSDecimalNumber.h`

objCType

Returns a C string containing the Objective-C type of the data contained in the receiver, which for an `NSDecimalNumber` object is always "d" (for double).

```
- (const char *)objCType
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSDecimalNumber.h`

Constants

NSDecimalNumber Exception Names

Names of the various exceptions raised by `NSDecimalNumber` to indicate computational errors.

```
extern NSString *NSDecimalNumberExactnessException;
extern NSString *NSDecimalNumberOverflowException;
extern NSString *NSDecimalNumberUnderflowException;
extern NSString *NSDecimalNumberDivideByZeroException;
```

Constants

`NSDecimalNumberExactnessException`

The name of the exception raised if there is an exactness error.

Available in Mac OS X v10.0 and later.

Declared in `NSDecimalNumber.h`.

`NSDecimalNumberOverflowException`

The name of the exception raised on overflow.

Available in Mac OS X v10.0 and later.

Declared in `NSDecimalNumber.h`.

`NSDecimalNumberUnderflowException`

The name of the exception raised on underflow.

Available in Mac OS X v10.0 and later.

Declared in `NSDecimalNumber.h`.

`NSDecimalNumberDivideByZeroException`

The name of the exception raised on divide by zero.

Available in Mac OS X v10.0 and later.

Declared in `NSDecimalNumber.h`.

Declared In

`NSDecimalNumber.h`

NSDecimalNumberHandler Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSDecimalNumberBehaviors NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSDecimalNumber.h
Companion guide	Number and Value Programming Topics for Cocoa

Overview

`NSDecimalNumberHandler` is a class that adopts the `NSDecimalNumberBehaviors` protocol. This class allows you to set the way an `NSDecimalNumber` object rounds off and handles errors, without having to create a custom class.

You can use an instance of this class as an argument to any of the `NSDecimalNumber` methods that end with `...Behavior:`. If you don't think you need special behavior, you probably don't need this class—it is likely that `NSDecimalNumber`'s default behavior will suit your needs.

For more information, see the NSDecimalNumberBehaviors protocol specification.

Adopted Protocols

NSDecimalNumberBehaviors
- `- roundingMode` (page 2044)
- `- scale` (page 2045)
- `- exceptionDuringOperation:error:leftOperand:rightOperand:` (page 2044)

NSCoding
- `- encodeWithCoder:` (page 2034)
- `- initWithCoder:` (page 2034)

Tasks

Creating a Decimal Number Handler

`+ defaultDecimalNumberHandler` (page 485)
Returns the default instance of `NSDecimalNumberHandler`.

`+ decimalNumberHandlerWithRoundingMode:scale:raiseOnExactness:raiseOnOverflow:raiseOnUnderflow:raiseOnDivideByZero:` (page 484)
Returns an `NSDecimalNumberHandler` object with customized behavior.

Initializing a Decimal Number Handler

`- initWithRoundingMode:scale:raiseOnExactness:raiseOnOverflow:raiseOnUnderflow:raiseOnDivideByZero:` (page 485)
Returns an `NSDecimalNumberHandler` object initialized so it behaves as specified by the method's arguments.

Class Methods

decimalNumberHandlerWithRoundingMode:scale:raiseOnExactness:raiseOnOverflow:raiseOnUnderflow:raiseOnDivideByZero:

Returns an `NSDecimalNumberHandler` object with customized behavior.

```
+ (id)decimalNumberHandlerWithRoundingMode:(NSRoundingMode)roundingMode
    scale:(short)scale raiseOnExactness:(BOOL)raiseOnExactness
    raiseOnOverflow:(BOOL)raiseOnOverflow raiseOnUnderflow:(BOOL)raiseOnUnderflow
     raiseOnDivideByZero:(BOOL)raiseOnDivideByZero
```

Parameters

roundingMode
The rounding mode to use. There are four possible values: `NSRoundUp`, `NSRoundDown`, `NSRoundPlain`, and `NSRoundBankers`.

scale
The number of digits a rounded value should have after its decimal point.

raiseOnExactness
If `YES`, in the event of an exactness error the handler will raise an exception, otherwise it will ignore the error and return control to the calling method.

raiseOnOverflow
If `YES`, in the event of an overflow error the handler will raise an exception, otherwise it will ignore the error and return control to the calling method

raiseOnUnderflow
If `YES`, in the event of an underflow error the handler will raise an exception, otherwise it will ignore the error and return control to the calling method

raiseOnDivideByZero
If YES, in the event of a divide by zero error the handler will raise an exception, otherwise it will ignore the error and return control to the calling method

Return Value
An NSDecimalNumberHandler object with customized behavior.

Discussion
See the NSDecimalNumberBehaviors protocol specification for a complete explanation of the possible behaviors.

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

Declared In
NSDecimalNumber.h

defaultDecimalNumberHandler

Returns the default instance of NSDecimalNumberHandler.

```
+ (id)defaultDecimalNumberHandler
```

Return Value
The default instance of NSDecimalNumberHandler.

Discussion
This default decimal number handler rounds to the closest possible return value. It assumes your need for precision does not exceed 38 significant digits, and it raises an exception when its NSDecimalNumber object tries to divide by 0 or when its NSDecimalNumber object produces a number too big or too small to be represented.

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

Declared In
NSDecimalNumber.h

Instance Methods

initWithRoundingMode:scale:raiseOnExactness:raiseOnOverflow:raiseOnUnderflow:raiseOnDivideByZero:

Returns an NSDecimalNumberHandler object initialized so it behaves as specified by the method's arguments.

```
- (id)initWithRoundingMode:(NSRoundingMode)roundingMode scale:(short)scale
    raiseOnExactness:(BOOL)raiseOnExactness raiseOnOverflow:(BOOL)raiseOnOverflow
     raiseOnUnderflow:(BOOL)raiseOnUnderflow
    raiseOnDivideByZero:(BOOL)raiseOnDivideByZero
```

Parameters

roundingMode

The rounding mode to use. There are four possible values: `NSRoundUp`, `NSRoundDown`, `NSRoundPlain`, and `NSRoundBankers`.

scale

The number of digits a rounded value should have after its decimal point.

raiseOnExactness

If `YES`, in the event of an exactness error the handler will raise an exception, otherwise it will ignore the error and return control to the calling method.

raiseOnOverflow

If `YES`, in the event of an overflow error the handler will raise an exception, otherwise it will ignore the error and return control to the calling method

raiseOnUnderflow

If `YES`, in the event of an underflow error the handler will raise an exception, otherwise it will ignore the error and return control to the calling method

raiseOnDivideByZero

If `YES`, in the event of a divide by zero error the handler will raise an exception, otherwise it will ignore the error and return control to the calling method

Return Value

An initialized `NSDecimalNumberHandler` object initialized with customized behavior. The returned object might be different than the original receiver.

Discussion

See the NSDecimalNumberBehaviors protocol specification for a complete explanation of the possible behaviors.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSDecimalNumber.h`

NSDeleteCommand Class Reference

Inherits from	NSScriptCommand : NSObject
Conforms to	NSCoding (NSScriptCommand) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptStandardSuiteCommands.h
Companion guides	Cocoa Scripting Guide Key-Value Coding Programming Guide

Overview

An instance of `NSDeleteCommand` deletes the specified scriptable object or objects (such as words, paragraphs, and so on).

Suppose, for example, a user executes a script that sends the command `delete the third rectangle in the first document` to the Sketch sample application (located in `/Developer/Examples/AppKit`). Cocoa creates an `NSDeleteCommand` object to perform the operation. When the command is executed, it uses the key-value coding mechanism (by invoking `removeValueAtIndex:fromPropertyWithKey:`) to remove the specified object or objects from their container. See the description for `removeValueAtIndex:fromPropertyWithKey:` (page 2119) for related information.

`NSDeleteCommand` is part of Cocoa's built-in scripting support. Most applications don't need to subclass `NSDeleteCommand` or call its methods.

Tasks

Working with Specifiers

- `keySpecifier` (page 488)
  Returns a specifier for the object or objects to be deleted.
- `setReceiversSpecifier:` (page 488)
  Sets the receiver's object specifier.

Instance Methods

keySpecifier

Returns a specifier for the object or objects to be deleted.

```
- (NSScriptObjectSpecifier *)keySpecifier
```

Return Value
A specifier for the object or objects to be deleted.

Discussion
Note that this may be different than the specifier or specifiers set by `setReceiversSpecifier:` (page 488).

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

Declared In
`NSScriptStandardSuiteCommands.h`

setReceiversSpecifier:

Sets the receiver's object specifier.

```
- (void)setReceiversSpecifier:(NSScriptObjectSpecifier *)receiversRef
```

Parameters
receiversRef
The receiver's object specifier.

Discussion
This method overrides `setReceiversSpecifier:` (page 1390) in `NSScriptCommand`. It performs the same function as the overridden method, with a critical difference: it causes the container specifier part of the passed-in object specifier to become the receiver specifier of the command, and the key part of the passed-in object specifier to become the key specifier. If, for example, *receiversRef* is a specifier for `the third rectangle of the first document`, the receiver specifier is `the first document` while the key specifier is `the third rectangle`.

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

Declared In
`NSScriptStandardSuiteCommands.h`

NSDeserializer Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSSerialization.h
Availability	Deprecated in Mac OS X v10.2.
Companion guide	Archives and Serializations Programming Guide for Cocoa

Overview

Note: NSDeserializer is obsolete and has been deprecated. Instead use `NSPropertyListSerialization`.

The `NSDeserializer` class declares methods that convert a representation of a property list (as contained in an `NSData` object) into a structure of property list objects in memory. The `NSDeserializer` class object itself provides these methods—you don't create instances of `NSDeserializer`. Options to these methods allow you to specify that container objects (arrays or dictionaries) in the resulting graph be mutable or immutable; that deserialization begin at the start of the data or from some position within it; or that deserialization occur lazily, so a property list is deserialized only if it is actually going to be accessed.

Tasks

Deserializing a Property List

`+ deserializePropertyListFromData:atCursor:mutableContainers:` (page 490) Deprecated in Mac OS X v10.2

Returns a property list object from a given location in a given serialized representation of a property list.

`+ deserializePropertyListFromData:mutableContainers:` (page 490) Deprecated in Mac OS X v10.2

Returns a property list object from given serialized data, optionally making the list elements mutable.

`+ deserializePropertyListLazilyFromData:atCursor:length:mutableContainers:` (page 491) Deprecated in Mac OS X v10.2

Returns a property list from a given location in a given serialized representation of a property list.

Class Methods

deserializePropertyListFromData:atCursor:mutableContainers:

Returns a property list object from a given location in a given serialized representation of a property list. (Deprecated in Mac OS X v10.2.)

```
+ (id)deserializePropertyListFromData:(NSData *)data atCursor:(unsigned *)cursor
    mutableContainers:(BOOL)mutable
```

Parameters

data

A serialized representation of a property list.

cursor

mutable

If YES and the property list object is a dictionary or an array, the recomposed object is made mutable

Return Value

A property list object corresponding to the representation in *data* at the location *cursor*. Returns nil if the property list object is not valid for property lists.

Availability

Deprecated in Mac OS X v10.2.

Declared In

NSSerialization.h

deserializePropertyListFromData:mutableContainers:

Returns a property list object from given serialized data, optionally making the list elements mutable. (Deprecated in Mac OS X v10.2.)

```
+ (id)deserializePropertyListFromData:(NSData *)serialization
    mutableContainers:(BOOL)mutable
```

Parameters

serialization

A serialized representation of a property list.

mutable

If YES and the property list object is a dictionary or an array, the recomposed object is made mutable.

Return Value

A property list object corresponding to the representation in *serialization*, or nil if *serialization* does not represent a property list.

Availability

Deprecated in Mac OS X v10.2.

Declared In

NSSerialization.h

deserializePropertyListLazilyFromData:atCursor:length:mutableContainers:

Returns a property list from a given location in a given serialized representation of a property list. (Deprecated in Mac OS X v10.2.)

```
+ (id)deserializePropertyListLazilyFromData:(NSData *)data atCursor:(unsigned
    *)cursor length:(unsigned)length mutableContainers:(BOOL)mutable
```

Parameters

data

A serialized representation of a property list.

cursor

The cursor location.

length

The number of bytes to read.

mutable

If `YES` and the object is a dictionary or an array, the recomposed object is made mutable.

Return Value

A property list from *data* at location *cursor*, or `nil` if *data* does not represent a property list.

Discussion

The deserialization proceeds lazily—that is, if the data at *cursor* has a length greater than *length*, a proxy is substituted for the actual property list as long as the constituent objects of that property list are not accessed.

Availability

Deprecated in Mac OS X v10.2.

Declared In

`NSSerialization.h`

NSDictionary Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSMutableCopying NSFastEnumeration NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSDictionary.h Foundation/NSFileManager.h Foundation/NSKeyValueCoding.h
Companion guides	Collections Programming Topics for Cocoa Property List Programming Guide
Related sample code	MyPhoto QTCoreVideo301 Quartz Composer WWDC 2005 TextEdit StickiesExample TextEditPlus

Overview

The `NSDictionary` class declares the programmatic interface to objects that manage immutable associations of keys and values. Use this class or its subclass `NSMutableDictionary` when you need a convenient and efficient way to retrieve data associated with an arbitrary key. (For convenience, we use the term **dictionary** to refer to any instance of one of these classes without specifying its exact class membership.)

A key-value pair within a dictionary is called an entry. Each entry consists of one object that represents the key and a second object that is that key's value. Within a dictionary, the keys are unique. That is, no two keys in a single dictionary are equal (as determined by `isEqual:` (page 2101)). In general, a key can be any object (provided that it conforms to the `NSCopying` protocol—see below), but note that when using key-value coding the key must be a string (see Key-Value Coding Fundamentals). Neither a key nor a value can be `nil`; if you need to represent a null value in a dictionary, you should use `NSNull`.

An instance of `NSDictionary` is an immutable dictionary: you establish its entries when it's created and cannot modify them afterward. An instance of `NSMutableDictionary` is a mutable dictionary: you can add or delete entries at any time, and the object automatically allocates memory as needed. The dictionary classes adopt the `NSCopying` and `NSMutableCopying` protocols, making it convenient to convert a dictionary of one type to the other.

`NSDictionary` and `NSMutableDictionary` are part of a class cluster, so the objects you create with this interface are not actual instances of the these two classes. Rather, the instances belong to one of their private subclasses. Although a dictionary's class is private, its interface is public, as declared by these abstract superclasses, `NSDictionary` and `NSMutableDictionary`.

Internally, a dictionary uses a hash table to organize its storage and to provide rapid access to a value given the corresponding key. However, the methods defined in this cluster insulate you from the complexities of working with hash tables, hashing functions, or the hashed value of keys. The methods described below take keys directly, not their hashed form.

Methods that add entries to dictionaries—whether as part of initialization (for all dictionaries) or during modification (for mutable dictionaries)—copy each key argument (keys must conform to the `NSCopying` protocol) and add the copies to the dictionary. Each corresponding value object receives a `retain` (page 2108) message to ensure that it won't be deallocated before the dictionary is through with it.

Enumeration

You can enumerate the contents of a dictionary by key or by value using the `NSEnumerator` object returned by keyEnumerator (page 519) and `objectEnumerator` (page 520) respectively. On Mac OS X v10.5 and later, `NSDictionary` supports the `NSFastEnumeration` protocol. You can use the `for...in` construct to enumerate the keys of a dictionary, as illustrated in the following example.

```
NSArray *keys = [NSArray arrayWithObjects:@"key1", @"key2", @"key3", nil];
NSArray *objects = [NSArray arrayWithObjects:@"value1", @"value2", @"value3",
nil];
NSDictionary *dictionary = [NSDictionary dictionaryWithObjects:objects
forKeys:keys];

for (id key in dictionary) {
    NSLog(@"key: %@, value: %@", key, [dictionary objectForKey:key]);
}
```

Primitive Methods

Three primitive methods of `NSDictionary`—`count` (page 505), `objectForKey:` (page 521), and `keyEnumerator` (page 519)—provide the basis for all of the other methods in its interface. The count (page 505) method returns the number of entries in the dictionary. objectForKey: (page 521) returns the value associated with a given key. keyEnumerator (page 519) returns an object that lets you iterate through each of the keys in the dictionary. The other methods declared here operate by invoking one or more of these primitives. The non-primitive methods provide convenient ways of accessing multiple entries at once.

Descriptions and Persistence

You can use the `description...` and `writeToFile:atomically:` (page 523) methods to write a *property list representation* of a dictionary to a string or to a file, respectively. These are not intended to be used for general persistent storage of your custom data objects—see instead *Archives and Serializations Programming Guide for Cocoa*.

Toll-Free Bridging

`NSDictionary` is "toll-free bridged" with its Core Foundation counterpart, *CFDictionary Reference*. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSDictionary *` parameter, you can pass in a `CFDictionaryRef`, and where you see a `CFDictionaryRef` parameter, you can pass in an `NSDictionary` instance (you cast one type to the other to suppress compiler warnings). This bridging also applies to concrete subclasses of `NSDictionary`. See Interchangeable Data Types for more information on toll-free bridging.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 2034)
- `initWithCoder:` (page 2034)

NSCopying
- `copyWithZone:` (page 2042)

NSMutableCopying
- `mutableCopyWithZone:` (page 2094)

NSFastEnumeration
- `countByEnumeratingWithState:objects:count:` (page 2053)

Tasks

Creating a Dictionary

`+ dictionary` (page 498)
Creates and returns an empty dictionary.

`+ dictionaryWithContentsOfFile:` (page 499)
Creates and returns a dictionary using the keys and values found in a file specified by a given path.

`+ dictionaryWithContentsOfURL:` (page 500)
Creates and returns a dictionary using the keys and values found in a resource specified by a given URL.

+ `dictionaryWithDictionary:` (page 500)
  Creates and returns a dictionary containing the keys and values from another given dictionary.

+ `dictionaryWithObject:forKey:` (page 500)
  Creates and returns a dictionary containing a given key and value.

+ `dictionaryWithObjects:forKeys:` (page 501)
  Creates and returns a dictionary containing entries constructed from the contents of an array of keys and an array of values.

+ `dictionaryWithObjects:forKeys:count:` (page 502)
  Creates and returns a dictionary containing *count* objects from the *objects* array.

+ `dictionaryWithObjectsAndKeys:` (page 503)
  Creates and returns a dictionary containing entries constructed from the specified set of values and keys.

Initializing an NSDictionary Instance

- `initWithContentsOfFile:` (page 515)
  Initializes a newly allocated dictionary using the keys and values found in a file at a given path.

- `initWithContentsOfURL:` (page 515)
  Initializes a newly allocated dictionary using the keys and values found at a given URL.

- `initWithDictionary:` (page 516)
  Initializes a newly allocated dictionary by placing in it the keys and values contained in another given dictionary.

- `initWithDictionary:copyItems:` (page 516)
  Initializes a newly allocated dictionary using the objects contained in another given dictionary.

- `initWithObjects:forKeys:` (page 517)
  Initializes a newly allocated dictionary with entries constructed from the contents of the *objects* and *keys* arrays.

- `initWithObjects:forKeys:count:` (page 517)
  Initializes a newly allocated dictionary with *count* entries.

- `initWithObjectsAndKeys:` (page 518)
  Initializes a newly allocated dictionary with entries constructed from the specified set of values and keys.

Counting Entries

- `count` (page 505)
  Returns the number of entries in the receiver.

Comparing Dictionaries

- `isEqualToDictionary:` (page 519)
  Returns a Boolean value that indicates whether the contents of the receiver are equal to the contents of another given dictionary.

Accessing Keys and Values

- `allKeys` (page 504)
  Returns a new array containing the receiver's keys.
- `allKeysForObject:` (page 504)
  Returns a new array containing the keys corresponding to all occurrences of a given object in the receiver.
- `allValues` (page 505)
  Returns a new array containing the receiver's values.
- `getObjects:andKeys:` (page 514)
  Returns by reference C arrays of the keys and values in the receiver.
- `keyEnumerator` (page 519)
  Returns an enumerator object that lets you access each key in the receiver.
- `keysSortedByValueUsingSelector:` (page 520)
  Returns an array of the receiver's keys, in the order they would be in if the receiver were sorted by its values.
- `objectEnumerator` (page 520)
  Returns an enumerator object that lets you access each value in the receiver.
- `objectForKey:` (page 521)
  Returns the value associated with a given key.
- `objectsForKeys:notFoundMarker:` (page 522)
  Returns the set of objects from the receiver that corresponds to the specified *keys* as an NSArray.
- `valueForKey:` (page 522)
  Returns the value associated with a given key.

Storing Dictionaries

- `writeToFile:atomically:` (page 523)
  Writes a property list representation of the contents of the receiver to a given path.
- `writeToURL:atomically:` (page 524)
  Writes a property list representation of the contents of the receiver to a given URL.

Accessing File Attributes

- `fileCreationDate` (page 508)
  Returns the value for the `NSFileCreationDate` key.
- `fileExtensionHidden` (page 508)
  Returns the value for the `NSFileExtensionHidden` key.
- `fileGroupOwnerAccountID` (page 508)
  Returns the value for the `NSFileGroupOwnerAccountID` key.
- `fileGroupOwnerAccountName` (page 509)
  Returns the value for the `NSFileGroupOwnerAccountName` key.
- `fileHFSCreatorCode` (page 509)
  Returns the value for the `NSFileHFSCreatorCode` key.

- `fileHFSTypeCode` (page 510)
  Returns the value for the `NSFileHFSTypeCode` key.
- `fileIsAppendOnly` (page 510)
  Returns the value for the `NSFileAppendOnly` key.
- `fileIsImmutable` (page 510)
  Returns the value for the `NSFileImmutable` key.
- `fileModificationDate` (page 511)
  Returns the value for the key `NSFileModificationDate`.
- `fileOwnerAccountID` (page 511)
  Returns the value for the `NSFileOwnerAccountID` key.
- `fileOwnerAccountName` (page 512)
  Returns the value for the key `NSFileOwnerAccountName`.
- `filePosixPermissions` (page 512)
  Returns the value for the key `NSFilePosixPermissions`.
- `fileSize` (page 512)
  Returns the value for the key `NSFileSize`.
- `fileSystemFileNumber` (page 513)
  Returns the value for the key `NSFileSystemFileNumber`.
- `fileSystemNumber` (page 513)
  Returns the value for the key `NSFileSystemNumber`.
- `fileType` (page 514)
  Returns the value for the key `NSFileType`.

Creating a Description

- `description` (page 505)
  Returns a string that represents the contents of the receiver, formatted as a property list.
- `descriptionInStringsFileFormat` (page 506)
  Returns a string that represents the contents of the receiver, formatted in `.strings` file format.
- `descriptionWithLocale:` (page 506)
  Returns a string object that represents the contents of the receiver, formatted as a property list.
- `descriptionWithLocale:indent:` (page 507)
  Returns a string object that represents the contents of the receiver, formatted as a property list.

Class Methods

dictionary

Creates and returns an empty dictionary.

```
+ (id)dictionary
```

Return Value
A new empty dictionary.

Discussion
This method is declared primarily for use with mutable subclasses of `NSDictionary`.

If you don't want a temporary object, you can also create an empty dictionary using `alloc...` and `init`.

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

Related Sample Code
QTKitMovieShuffler
QTSSInspector
StickiesExample

Declared In
`NSDictionary.h`

dictionaryWithContentsOfFile:

Creates and returns a dictionary using the keys and values found in a file specified by a given path.

```
+ (id)dictionaryWithContentsOfFile:(NSString *)path
```

Parameters

path

A full or relative pathname. The file identified by *path* must contain a string representation of a property list whose root object is a dictionary. The dictionary must contain only property list objects (instances of `NSData`, `NSDate`, `NSNumber`, `NSString`, `NSArray`, or `NSDictionary`). For more details, see *Property List Programming Guide*.

Return Value
A new dictionary that contains the dictionary at *path*, or `nil` if there is a file error or if the contents of the file are an invalid representation of a dictionary.

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

See Also
`- initWithContentsOfFile:` (page 515)

Related Sample Code
CapabilitiesSample
Cocoa - SGDataProc
LSMSmartCategorizer
Spotlight
SpotlightFortunes

Declared In
`NSDictionary.h`

dictionaryWithContentsOfURL:

Creates and returns a dictionary using the keys and values found in a resource specified by a given URL.

```
+ (id)dictionaryWithContentsOfURL:(NSURL *)aURL
```

Parameters

aURL

An URL that identifies a resource containing a string representation of a property list whose root object is a dictionary. The dictionary must contain only property list objects (instances of `NSData`, `NSDate`, `NSNumber`, `NSString`, `NSArray`, or `NSDictionary`). For more details, see *Property List Programming Guide*.

Return Value
A new dictionary that contains the dictionary at *aURL*, or `nil` if there is an error or if the contents of the resource are an invalid representation of a dictionary.

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

See Also
`- initWithContentsOfURL:` (page 515)

Declared In
`NSDictionary.h`

dictionaryWithDictionary:

Creates and returns a dictionary containing the keys and values from another given dictionary.

```
+ (id)dictionaryWithDictionary:(NSDictionary *)otherDictionary
```

Parameters

otherDictionary

A dictionary containing keys and values for the new dictionary.

Return Value
A new dictionary containing the keys and values found in *otherDictionary*.

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

See Also
`- initWithDictionary:` (page 516)

Related Sample Code
QTSSInspector

Declared In
`NSDictionary.h`

dictionaryWithObject:forKey:

Creates and returns a dictionary containing a given key and value.

```
+ (id)dictionaryWithObject:(id)anObject forKey:(id)aKey
```

Parameters

anObject

The value corresponding to *aKey*.

aKey

The key for *anObject*.

Return Value

A new dictionary containing a single object, *anObject*, for a single key, *aKey*.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ dictionaryWithObjects:forKeys:` (page 501)

`+ dictionaryWithObjects:forKeys:count:` (page 502)

`+ dictionaryWithObjectsAndKeys:` (page 503)

Related Sample Code

iSpend

PDF Annotation Editor

QTCoreVideo301

Quartz Composer WWDC 2005 TextEdit

WhackedTV

Declared In

`NSDictionary.h`

dictionaryWithObjects:forKeys:

Creates and returns a dictionary containing entries constructed from the contents of an array of keys and an array of values.

```
+ (id)dictionaryWithObjects:(NSArray *)objects forKeys:(NSArray *)keys
```

Parameters

objects

An array containing the values for the new dictionary.

keys

An array containing the keys for the new dictionary. Each key is copied (using `copyWithZone:` (page 2042); keys must conform to the `NSCopying` protocol), and the copy is added to the dictionary.

Return Value

A new dictionary containing entries constructed from the contents of *objects* and *keys*.

Discussion

This method steps through the *objects* and *keys* arrays, creating entries in the new dictionary as it goes. An `NSInvalidArgumentException` is raised if objects and keys don't have the same number of elements.

Availability

Available in Mac OS X v10.0 and later.

See Also
- `initWithObjects:forKeys:` (page 517)
+ `dictionaryWithObject:forKey:` (page 500)
+ `dictionaryWithObjects:forKeys:count:` (page 502)
+ `dictionaryWithObjectsAndKeys:` (page 503)

Related Sample Code
ImageMapExample
TimelineToTC

Declared In
`NSDictionary.h`

dictionaryWithObjects:forKeys:count:

Creates and returns a dictionary containing *count* objects from the *objects* array.

```
+ (id)dictionaryWithObjects:(id *)objects forKeys:(id *)keys count:(NSUInteger)count
```

Parameters

objects
A C array of values for the new dictionary.

keys
A C array of keys for the new dictionary. Each key is copied (using `copyWithZone:` (page 2042); keys must conform to the `NSCopying` protocol), and the copy is added to the new dictionary.

count
The number of elements to use from the *keys* and *objects* arrays. *count* must not exceed the number of elements in *objects* or *keys*.

Discussion
This method steps through the *objects* and *keys* arrays, creating entries in the new dictionary as it goes. An `NSInvalidArgumentException` is raised if a key or value object is `nil`.

The following code fragment illustrates how to create a dictionary that associates the alphabetic characters with their ASCII values:

```
static const NSInteger N_ENTRIES = 26;
NSDictionary *asciiDict;
NSString *keyArray[N_ENTRIES];
NSNumber *valueArray[N_ENTRIES];
NSInteger i;

for (i = 0; i < N_ENTRIES; i++) {

    char charValue = 'a' + i;
    keyArray[i] = [NSString stringWithFormat:@"%c", charValue];
    valueArray[i] = [NSNumber numberWithChar:charValue];
}

asciiDict = [NSDictionary dictionaryWithObjects:(id *)valueArray
                          forKeys:(id *)keyArray count:N_ENTRIES];
```

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

See Also
- `initWithObjects:forKeys:count:` (page 517)
+ `dictionaryWithObject:forKey:` (page 500)
+ `dictionaryWithObjects:forKeys:` (page 501)
+ `dictionaryWithObjectsAndKeys:` (page 503)

Declared In
`NSDictionary.h`

dictionaryWithObjectsAndKeys:

Creates and returns a dictionary containing entries constructed from the specified set of values and keys.

```
+ (id)dictionaryWithObjectsAndKeys:(id)firstObject , ...
```

Parameters

firstObject
The first value to add to the new dictionary.

`...`
First the key for *firstObject*, then a null-terminated list of alternating values and keys. If any key is `nil`, an `NSInvalidArgumentException` is raised.

Discussion
This method is similar to `dictionaryWithObjects:forKeys:` (page 501), differing only in the way key-value pairs are specified.

For example:

```
NSDictionary *dict = [NSDictionary dictionaryWithObjectsAndKeys:
    @"value1", @"key1", @"value2", @"key2", nil];
```

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

See Also
- `initWithObjectsAndKeys:` (page 518)
+ `dictionaryWithObject:forKey:` (page 500)
+ `dictionaryWithObjects:forKeys:` (page 501)
+ `dictionaryWithObjects:forKeys:count:` (page 502)

Related Sample Code
CIAnnotation
iSpend
Quartz Composer WWDC 2005 TextEdit
StickiesExample
TextEditPlus

Declared In
`NSDictionary.h`

Instance Methods

allKeys

Returns a new array containing the receiver's keys.

```
- (NSArray *)allKeys
```

Return Value
A new array containing the receiver's keys, or an empty array if the receiver has no entries.

Discussion
The order of the elements in the array is not defined.

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

See Also
- `allValues` (page 505)
- `allKeysForObject:` (page 504)
- `getObjects:andKeys:` (page 514)

Related Sample Code
Core Data HTML Store
CoreRecipes
EnhancedAudioBurn
ImageMapExample
StickiesExample

Declared In
`NSDictionary.h`

allKeysForObject:

Returns a new array containing the keys corresponding to all occurrences of a given object in the receiver.

```
- (NSArray *)allKeysForObject:(id)anObject
```

Parameters
anObject
The value to look for in the receiver.

Return Value
A new array containing the keys corresponding to all occurrences of *anObject* in the receiver. If no object matching *anObject* is found, returns an empty array.

Discussion
Each object in the receiver is sent an `isEqual:` (page 2101) message to determine if it's equal to *anObject*.

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

See Also
- `allKeys` (page 504)
- `keyEnumerator` (page 519)

Declared In
`NSDictionary.h`

allValues

Returns a new array containing the receiver's values.

```
- (NSArray *)allValues
```

Return Value
A new array containing the receiver's values, or an empty array if the receiver has no entries.

Discussion
The order of the values in the array isn't defined.

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

See Also
- `allKeys` (page 504)
- `getObjects:andKeys:` (page 514)
- `objectEnumerator` (page 520)

Related Sample Code
ImageMapExample

Declared In
`NSDictionary.h`

count

Returns the number of entries in the receiver.

```
- (NSUInteger)count
```

Return Value
The number of entries in the receiver.

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

Declared In
`NSDictionary.h`

description

Returns a string that represents the contents of the receiver, formatted as a property list.

```
- (NSString *)description
```

Return Value
A string that represents the contents of the receiver, formatted as a property list.

Discussion
If each key in the receiver is an `NSString` object, the entries are listed in ascending order by key, otherwise the order in which the entries are listed is undefined. This method is intended to produce readable output for debugging purposes, not for serializing data. If you want to store dictionary data for later retrieval, see *Property List Programming Guide* and *Archives and Serializations Programming Guide for Cocoa*.

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

See Also
- `descriptionWithLocale:` (page 506)
- `descriptionWithLocale:indent:` (page 507)

Related Sample Code
Sketch-112
TextLinks

Declared In
`NSDictionary.h`

descriptionInStringsFileFormat

Returns a string that represents the contents of the receiver, formatted in `.strings` file format.

```
- (NSString *)descriptionInStringsFileFormat
```

Return Value
A string that represents the contents of the receiver, formatted in `.strings` file format.

Discussion
The order in which the entries are listed is undefined.

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

Declared In
`NSDictionary.h`

descriptionWithLocale:

Returns a string object that represents the contents of the receiver, formatted as a property list.

```
- (NSString *)descriptionWithLocale:(id)locale
```

Parameters

locale

An object that specifies options used for formatting each of the receiver's keys and values; pass `nil` if you don't want them formatted.

Prior to Mac OS X v10.5, locale must be an instance of `NSDictionary`. With Mac OS X v10.5 and later, it may also be an `NSLocale` object.

Discussion

For a description of how *locale* is applied to each element in the receiver, see `descriptionWithLocale:indent:` (page 507).

If each key in the dictionary responds to `compare:`, the entries are listed in ascending order by key, otherwise the order in which the entries are listed is undefined.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `description` (page 505)
- `descriptionWithLocale:indent:` (page 507)

Declared In

`NSDictionary.h`

descriptionWithLocale:indent:

Returns a string object that represents the contents of the receiver, formatted as a property list.

```
- (NSString *)descriptionWithLocale:(id)locale indent:(NSUInteger)level
```

Parameters

locale

An object that specifies options used for formatting each of the receiver's keys and values; pass `nil` if you don't want them formatted.

Prior to Mac OS X v10.5, locale must be an instance of `NSDictionary`. With Mac OS X v10.5 and later, it may also be an `NSLocale` object.

level

Specifies a level of indent, to make the output more readable: set *level* to `0` to use four spaces to indent, or `1` to indent the output with a tab character

Return Value

A string object that represents the contents of the receiver, formatted as a property list.

Discussion

The returned `NSString` object contains the string representations of each of the receiver's entries. `descriptionWithLocale:indent:` obtains the string representation of a given key or value as follows:

- If the object is an `NSString` object, it is used as is.
- If the object responds to `descriptionWithLocale:indent:`, that method is invoked to obtain the object's string representation.
- If the object responds to `descriptionWithLocale:`, that method is invoked to obtain the object's string representation.

- If none of the above conditions is met, the object's string representation is obtained by invoking its `description` method.

If each key in the dictionary responds to `compare:`, the entries are listed in ascending order, by key. Otherwise, the order in which the entries are listed is undefined.

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

See Also
`- description` (page 505)
`- descriptionWithLocale:` (page 506)

Declared In
`NSDictionary.h`

fileCreationDate

Returns the value for the `NSFileCreationDate` key.

```
- (NSDate *)fileCreationDate
```

Return Value
The value for the `NSFileCreationDate` key, or `nil` if the receiver doesn't have an entry for the key.

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

Declared In
`NSFileManager.h`

fileExtensionHidden

Returns the value for the `NSFileExtensionHidden` key.

```
- (BOOL)fileExtensionHidden
```

Return Value
The value for the `NSFileExtensionHidden` key, or `NO` if the receiver doesn't have an entry for the key.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`NSFileManager.h`

fileGroupOwnerAccountID

Returns the value for the `NSFileGroupOwnerAccountID` key.

```
- (NSNumber *)fileGroupOwnerAccountID
```

Return Value
The value for the `NSFileGroupOwnerAccountID` key, or `nil` if the receiver doesn't have an entry for the key.

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

Declared In
`NSFileManager.h`

fileGroupOwnerAccountName

Returns the value for the `NSFileGroupOwnerAccountName` key.

```
- (NSString *)fileGroupOwnerAccountName
```

Return Value
The value for the key `NSFileGroupOwnerAccountName`, or `nil` if the receiver doesn't have an entry for the key.

Discussion
This and the other `file...` methods are for use with a dictionary, such as those returned from the methods `fileAttributesAtPath:traverseLink:` (page 645) (`NSFileManager`), `directoryAttributes` (page 526) (`NSDirectoryEnumerator`), and `fileAttributes` (page 526) (`NSDirectoryEnumerator`), that represents the POSIX attributes of a file or directory. This method returns the name of the corresponding file's group.

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

Declared In
`NSFileManager.h`

fileHFSCreatorCode

Returns the value for the `NSFileHFSCreatorCode` key.

```
- (OSType)fileHFSCreatorCode
```

Return Value
The value for the `NSFileHFSCreatorCode` key, or `0` if the receiver doesn't have an entry for the key.

Discussion
See HFS File Types for details on the `OSType` data type.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`NSFileManager.h`

fileHFSTypeCode

Returns the value for the `NSFileHFSTypeCode` key.

```
- (OSType)fileHFSTypeCode
```

Return Value
The value for the `NSFileHFSTypeCode` key, or `0` if the receiver doesn't have an entry for the key.

Discussion
See HFS File Types for details on the `OSType` data type.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`NSFileManager.h`

fileIsAppendOnly

Returns the value for the `NSFileAppendOnly` key.

```
- (BOOL)fileIsAppendOnly
```

Return Value
The value for the `NSFileAppendOnly` key, or `NO` if the receiver doesn't have an entry for the key.

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

Declared In
`NSFileManager.h`

fileIsImmutable

Returns the value for the `NSFileImmutable` key.

```
- (BOOL)fileIsImmutable
```

Return Value
The value for the `NSFileImmutable` key, or `NO` if the receiver doesn't have an entry for the key.

Discussion
This and the other `file...` methods are for use with a dictionary, such as those returned from the methods `fileAttributesAtPath:traverseLink:` (page 645) (`NSFileManager`), `directoryAttributes` (page 526) (`NSDirectoryEnumerator`), and `fileAttributes` (page 526) (`NSDirectoryEnumerator`), that represents the POSIX attributes of a file or directory.

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

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
NSFileManager.h

fileModificationDate

Returns the value for the key NSFileModificationDate.

```
- (NSDate *)fileModificationDate
```

Return Value
The value for the key NSFileModificationDate, or nil if the receiver doesn't have an entry for the key.

Discussion
This and the other file... methods are for use with a dictionary, such as those returned from the methods fileAttributesAtPath:traverseLink: (page 645) (NSFileManager), directoryAttributes (page 526) (NSDirectoryEnumerator), and fileAttributes (page 526) (NSDirectoryEnumerator), that represents the POSIX attributes of a file or directory. This method returns the date that the file's data was last modified.

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

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
NSFileManager.h

fileOwnerAccountID

Returns the value for the NSFileOwnerAccountID key.

```
- (NSNumber *)fileOwnerAccountID
```

Return Value
The value for the NSFileOwnerAccountID key, or nil if the receiver doesn't have an entry for the key.

Discussion
This and the other file... methods are for use with a dictionary, such as those returned from the methods fileAttributesAtPath:traverseLink: (page 645) (NSFileManager), directoryAttributes (page 526) (NSDirectoryEnumerator), and fileAttributes (page 526) (NSDirectoryEnumerator), that represents the POSIX attributes of a file or directory. This method returns the account name of the file's owner.

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

Declared In
NSFileManager.h

fileOwnerAccountName

Returns the value for the key `NSFileOwnerAccountName`.

```
- (NSString *)fileOwnerAccountName
```

Return Value
The value for the key `NSFileOwnerAccountName`, or `nil` if the receiver doesn't have an entry for the key.

Discussion
This and the other `file...` methods are for use with a dictionary, such as those returned from the methods `fileAttributesAtPath:traverseLink:` (page 645) (`NSFileManager`), `directoryAttributes` (page 526) (`NSDirectoryEnumerator`), and `fileAttributes` (page 526) (`NSDirectoryEnumerator`), that represents the POSIX attributes of a file or directory. This method returns the account name of the file's owner.

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

Declared In
`NSFileManager.h`

filePosixPermissions

Returns the value for the key `NSFilePosixPermissions`.

```
- (NSUInteger)filePosixPermissions
```

Return Value
The value, as an `unsigned long`, for the key `NSFilePosixPermissions`, or `0` if the receiver doesn't have an entry for the key.

Discussion
This and the other `file...` methods are for use with a dictionary, such as those returned from the methods `fileAttributesAtPath:traverseLink:` (page 645) (`NSFileManager`), `directoryAttributes` (page 526) (`NSDirectoryEnumerator`), and `fileAttributes` (page 526) (`NSDirectoryEnumerator`), that represents the POSIX attributes of a file or directory. This method returns the file's permissions.

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

Declared In
`NSFileManager.h`

fileSize

Returns the value for the key `NSFileSize`.

```
- (unsigned long long)fileSize
```

Return Value
The value, as an `unsigned long long`, for the key `NSFileSize`, or `0` if the receiver doesn't have an entry for the key.

Discussion
This and the other `file...` methods are for use with a dictionary such, as those returned from the methods `fileAttributesAtPath:traverseLink:` (page 645) (`NSFileManager`), `directoryAttributes` (page 526) (`NSDirectoryEnumerator`), and `fileAttributes` (page 526) (`NSDirectoryEnumerator`), that represents the POSIX attributes of a file or directory. This method returns the file's size.

Special Considerations

If the file has a resource fork, the returned value does *not* include the size of the resource fork.

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

Declared In
`NSFileManager.h`

fileSystemFileNumber

Returns the value for the key `NSFileSystemFileNumber`.

```
- (NSUInteger)fileSystemFileNumber
```

Return Value
The value, as an `unsigned long`, for the key `NSFileSystemFileNumber`, or `0` if the receiver doesn't have an entry for the key

Discussion
This and the other `file...` methods are for use with a dictionary, such as those returned from the methods `fileAttributesAtPath:traverseLink:` (page 645) (`NSFileManager`), `directoryAttributes` (page 526) (`NSDirectoryEnumerator`), and `fileAttributes` (page 526) (`NSDirectoryEnumerator`), that represents the POSIX attributes of a file or directory. This method returns the file's inode.

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

Declared In
`NSFileManager.h`

fileSystemNumber

Returns the value for the key `NSFileSystemNumber`.

```
- (NSInteger)fileSystemNumber
```

Return Value
The value, as an `unsigned long`, for the key `NSFileSystemNumber`, or `0` if the receiver doesn't have an entry for the key

Discussion
This and the other `file...` methods are for use with a dictionary, such as those returned from the methods `fileAttributesAtPath:traverseLink:` (page 645) (`NSFileManager`), `directoryAttributes` (page 526) (`NSDirectoryEnumerator`), and `fileAttributes` (page 526) (`NSDirectoryEnumerator`), that represents the POSIX attributes of a file or directory. This method returns the ID of the device containing the file.

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

Declared In
`NSFileManager.h`

fileType

Returns the value for the key `NSFileType`.

```
- (NSString *)fileType
```

Return Value
The value for the key `NSFileType`, or `nil` if the receiver doesn't have an entry for the key.

Discussion
This and the other `file...` methods are for use with a dictionary, such as those returned from the methods `fileAttributesAtPath:traverseLink:` (page 645) (`NSFileManager`), `directoryAttributes` (page 526) (`NSDirectoryEnumerator`), and `fileAttributes` (page 526) (`NSDirectoryEnumerator`), that represents the POSIX attributes of a file or directory. This method returns the file's type. Possible return values are described in the "Constants" section of `NSFileManager`.

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

Declared In
`NSFileManager.h`

getObjects:andKeys:

Returns by reference C arrays of the keys and values in the receiver.

```
- (void)getObjects:(id *)objects andKeys:(id *)keys
```

Parameters

objects
Upon return, contains a C array of the values in the receiver.

keys
Upon return, contains a C array of the keys in the receiver.

Discussion
The elements in the returned arrays are ordered such that the first element in *objects* is the value for the first key in *keys* and so on.

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

See Also
- `allKeys` (page 504)
- `allValues` (page 505)
- `objectForKey:` (page 521)
- `objectsForKeys:notFoundMarker:` (page 522)

Declared In
NSDictionary.h

initWithContentsOfFile:

Initializes a newly allocated dictionary using the keys and values found in a file at a given path.

```
- (id)initWithContentsOfFile:(NSString *)path
```

Parameters

path

A full or relative pathname. The file identified by *path* must contain a string representation of a property list whose root object is a dictionary. The dictionary must contain only property list objects (instances of NSData, NSDate, NSNumber, NSString, NSArray, or NSDictionary). For more details, see *Property List Programming Guide*.

Return Value
An initialized object—which might be different than the original receiver—that contains the dictionary at *path*, or nil if there is a file error or if the contents of the file are an invalid representation of a dictionary.

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

See Also
+ dictionaryWithContentsOfFile: (page 499)

Declared In
NSDictionary.h

initWithContentsOfURL:

Initializes a newly allocated dictionary using the keys and values found at a given URL.

```
- (id)initWithContentsOfURL:(NSURL *)aURL
```

Parameters

aURL

An URL that identifies a resource containing a string representation of a property list whose root object is a dictionary. The dictionary must contain only property list objects (instances of NSData, NSDate, NSNumber, NSString, NSArray, or NSDictionary). For more details, see *Property List Programming Guide*.

Return Value
An initialized object—which might be different than the original receiver—that contains the dictionary at *aURL*, or nil if there is an error or if the contents of the resource are an invalid representation of a dictionary.

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

See Also
+ dictionaryWithContentsOfURL: (page 500)

Declared In
NSDictionary.h

initWithDictionary:

Initializes a newly allocated dictionary by placing in it the keys and values contained in another given dictionary.

```
- (id)initWithDictionary:(NSDictionary *)otherDictionary
```

Parameters

otherDictionary

A dictionary containing keys and values for the new dictionary.

Return Value

An initialized object—which might be different than the original receiver—containing the keys and values found in *otherDictionary*.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ dictionaryWithDictionary:` (page 500)

Declared In

`NSDictionary.h`

initWithDictionary:copyItems:

Initializes a newly allocated dictionary using the objects contained in another given dictionary.

```
- (id)initWithDictionary:(NSDictionary *)otherDictionary copyItems:(BOOL)flag
```

Parameters

otherDictionary

A dictionary containing keys and values for the new dictionary.

flag

A flag that specifies whether values in *otherDictionary* should be copied. If `YES`, the members of *otherDictionary* are copied, and the copies are added to the receiver. If `NO`, the values of *otherDictionary* are retained by the new dictionary.

Return Value

An initialized object—which might be different than the original receiver—containing the keys and values found in *otherDictionary*.

Discussion

Note that `copyWithZone:` (page 2042) is used to make copies. Thus, the receiver's new member objects may be immutable, even though their counterparts in *otherDictionary* were mutable. Also, members must conform to the `NSCopying` protocol.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- initWithDictionary:` (page 516)

Declared In

`NSDictionary.h`

initWithObjects:forKeys:

Initializes a newly allocated dictionary with entries constructed from the contents of the *objects* and *keys* arrays.

```
- (id)initWithObjects:(NSArray *)objects forKeys:(NSArray *)keys
```

Parameters

objects

An array containing the values for the new dictionary.

keys

An array containing the keys for the new dictionary. Each key is copied (using `copyWithZone:` (page 2042); keys must conform to the `NSCopying` protocol), and the copy is added to the new dictionary.

Discussion

This method steps through the *objects* and *keys* arrays, creating entries in the new dictionary as it goes. An `NSInvalidArgumentException` is raised if the objects and keys arrays do not have the same number of elements.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ dictionaryWithObjects:forKeys:` (page 501)
`- initWithObjects:forKeys:count:` (page 517)
`- initWithObjectsAndKeys:` (page 518)

Related Sample Code

QTCoreVideo301

Declared In

`NSDictionary.h`

initWithObjects:forKeys:count:

Initializes a newly allocated dictionary with *count* entries.

```
- (id)initWithObjects:(id *)objects forKeys:(id *)keys count:(NSUInteger)count
```

Parameters

objects

A C array of values for the new dictionary.

keys

A C array of keys for the new dictionary. Each key is copied (using `copyWithZone:` (page 2042); keys must conform to the `NSCopying` protocol), and the copy is added to the new dictionary.

count

The number of elements to use from the *keys* and *objects* arrays. *count* must not exceed the number of elements in *objects* or *keys*.

Discussion

This method steps through the *objects* and *keys* arrays, creating entries in the new dictionary as it goes. An `NSInvalidArgumentException` is raised if a key or value object is `nil`.

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

See Also
+ `dictionaryWithObjects:forKeys:count:` (page 502)
- `initWithObjects:forKeys:` (page 517)
- `initWithObjectsAndKeys:` (page 518)

Declared In
`NSDictionary.h`

initWithObjectsAndKeys:

Initializes a newly allocated dictionary with entries constructed from the specified set of values and keys.

```
- (id)initWithObjectsAndKeys:(id)firstObject , ...
```

Parameters

firstObject
The first value to add to the new dictionary.

`...`
First the key for *firstObject*, then a null-terminated list of alternating values and keys. If any key is `nil`, an `NSInvalidArgumentException` is raised.

Discussion
This method is similar to `initWithObjects:forKeys:` (page 517), differing only in the way in which the key-value pairs are specified.

For example:

```
NSDictionary *dict = [[NSDictionary alloc] initWithObjectsAndKeys:
    @"value1", @"key1", @"value2", @"key2", nil];
```

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

See Also
+ `dictionaryWithObjectsAndKeys:` (page 503)
- `initWithObjects:forKeys:` (page 517)
- `initWithObjects:forKeys:count:` (page 517)

Related Sample Code
GLChildWindowDemo
QTRecorder
Quartz Composer WWDC 2005 TextEdit
SpeedometerView
TextEditPlus

Declared In
`NSDictionary.h`

isEqualToDictionary:

Returns a Boolean value that indicates whether the contents of the receiver are equal to the contents of another given dictionary.

```
- (BOOL)isEqualToDictionary:(NSDictionary *)otherDictionary
```

Parameters

otherDictionary

The dictionary with which to compare the receiver.

Return Value

`YES` if the contents of *otherDictionary* are equal to the contents of the receiver, otherwise `NO`.

Discussion

Two dictionaries have equal contents if they each hold the same number of entries and, for a given key, the corresponding value objects in each dictionary satisfy the `isEqual:` (page 2101) test.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- isEqual:` (page 2101) (`NSObject` protocol)

Declared In

`NSDictionary.h`

keyEnumerator

Returns an enumerator object that lets you access each key in the receiver.

```
- (NSEnumerator *)keyEnumerator
```

Return Value

An enumerator object that lets you access each key in the receiver.

Discussion

The following code fragment illustrates how you might use this method.

```
NSEnumerator *enumerator = [myDictionary keyEnumerator];
id key;

while ((key = [enumerator nextObject])) {
    /* code that uses the returned key */
}
```

If you use this method with instances of mutable subclasses of `NSDictionary`, your code should not modify the entries during enumeration. If you intend to modify the entries, use the `allKeys` (page 504) method to create a "snapshot" of the dictionary's keys. Then use this snapshot to traverse the entries, modifying them along the way.

Note that the `objectEnumerator` (page 520) method provides a convenient way to access each value in the dictionary.

Availability

Available in Mac OS X v10.0 and later.

See Also
- `allKeys` (page 504)
- `allKeysForObject:` (page 504)
- `getObjects:andKeys:` (page 514)
- `objectEnumerator` (page 520)
- `nextObject` (page 558) (NSEnumerator)

Related Sample Code
ColorSyncDevices-Cocoa
LSMSmartCategorizer
StickiesExample

Declared In
`NSDictionary.h`

keysSortedByValueUsingSelector:

Returns an array of the receiver's keys, in the order they would be in if the receiver were sorted by its values.

```
- (NSArray *)keysSortedByValueUsingSelector:(SEL)comparator
```

Parameters

comparator

A selector that specifies the method to use to compare the values in the receiver.

The *comparator* method should return `NSOrderedAscending` if the receiver is smaller than the argument, `NSOrderedDescending` if the receiver is larger than the argument, and `NSOrderedSame` if they are equal.

Return Value
An array of the receiver's keys, in the order they would be in if the receiver were sorted by its values.

Discussion
Pairs of dictionary values are compared using the comparison method specified by *comparator*; the *comparator* message is sent to one of the values and has as its single argument the other value from the dictionary.

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

See Also
- `allKeys` (page 504)
- `sortedArrayUsingSelector:` (page 138) (`NSArray`)

Declared In
`NSDictionary.h`

objectEnumerator

Returns an enumerator object that lets you access each value in the receiver.

```
- (NSEnumerator *)objectEnumerator
```

Return Value
An enumerator object that lets you access each value in the receiver.

Discussion
The following code fragment illustrates how you might use the method.

```
NSEnumerator *enumerator = [myDictionary objectEnumerator];
id value;

while ((value = [enumerator nextObject])) {
    /* code that acts on the dictionary's values */
}
```

If you use this method with instances of mutable subclasses of `NSDictionary`, your code should not modify the entries during enumeration. If you intend to modify the entries, use the `allValues` (page 505) method to create a "snapshot" of the dictionary's values. Work from this snapshot to modify the values.

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

See Also
- `keyEnumerator` (page 519)
- `nextObject` (page 558) (`NSEnumerator`)

Declared In
`NSDictionary.h`

objectForKey:

Returns the value associated with a given key.

```
- (id)objectForKey:(id)aKey
```

Parameters

aKey
The key for which to return the corresponding value.

Return Value
The value associated with *aKey*, or `nil` if no value is associated with *aKey*.

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

See Also
- `allKeys` (page 504)
- `allValues` (page 505)
- `getObjects:andKeys:` (page 514)

Related Sample Code
iSpend
People
QTCoreVideo301
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
NSDictionary.h

objectsForKeys:notFoundMarker:

Returns the set of objects from the receiver that corresponds to the specified *keys* as an NSArray.

```
- (NSArray *)objectsForKeys:(NSArray *)keys notFoundMarker:(id)anObject
```

Parameters

keys

The keys for which to return corresponding values.

anObject

The marker object to place in the corresponding element of the returned array if an object isn't found in the receiver to correspond to a given key.

Discussion
The objects in the returned array and the *keys* array have a one-for-one correspondence, so that the *n*th object in the returned array corresponds to the *n*th key in *keys*.

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

See Also
- `allKeys` (page 504)
- `allValues` (page 505)
- `getObjects:andKeys:` (page 514)

Declared In
NSDictionary.h

valueForKey:

Returns the value associated with a given key.

```
- (id)valueForKey:(NSString *)key
```

Parameters

key

The key for which to return the corresponding value. Note that when using key-value coding, the key must be a string (see Key-Value Coding Fundamentals).

Return Value
The value associated with *key*.

Discussion
If *key* does not start with "@", invokes `objectForKey:` (page 521). If *key* does start with "@", strips the "@" and invokes `[super valueForKey:]` with the rest of the key.

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

See Also
- `setValue:forKey:` (page 962) (`NSMutableDictionary`)
- `getObjects:andKeys:` (page 514)

Related Sample Code
CustomAtomicStoreSubclass
ImageMapExample
NSOperationSample
SimpleCalendar
StickiesExample

Declared In
`NSKeyValueCoding.h`

writeToFile:atomically:

Writes a property list representation of the contents of the receiver to a given path.

```
- (BOOL)writeToFile:(NSString *)path atomically:(BOOL)flag
```

Parameters

path

The path at which to write the file.

If *path* contains a tilde (~) character, you must expand it with `stringByExpandingTildeInPath` (page 1602) before invoking this method.

flag

A flag that specifies whether the file should be written atomically.

If *flag* is `YES`, the receiver is written to an auxiliary file, and then the auxiliary file is renamed to *path*. If *flag* is `NO`, the dictionary is written directly to *path*. The `YES` option guarantees that *path*, if it exists at all, won't be corrupted even if the system should crash during writing.

Return Value
`YES` if the file is written successfully, otherwise `NO`.

Discussion
This method recursively validates that all the contained objects are property list objects (instances of `NSData`, `NSDate`, `NSNumber`, `NSString`, `NSArray`, or `NSDictionary`) before writing out the file, and returns `NO` if all the objects are not property list objects, since the resultant file would not be a valid property list.

If the receiver's contents are all property list objects, the file written by this method can be used to initialize a new dictionary with the class method `dictionaryWithContentsOfFile:` (page 499) or the instance method `initWithContentsOfFile:` (page 515).

For more information about property lists, see *Property List Programming Guide*.

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

Declared In
`NSDictionary.h`

writeToURL:atomically:

Writes a property list representation of the contents of the receiver to a given URL.

```
- (BOOL)writeToURL:(NSURL *)aURL atomically:(BOOL)flag
```

Parameters

aURL

The URL to which to write the receiver.

flag

A flag that specifies whether the output should be written atomically.

If *flag* is YES, the receiver is written to an auxiliary location, and then the auxiliary location is renamed to *aURL*. If *flag* is NO, the dictionary is written directly to *aURL*. The YES option guarantees that *aURL*, if it exists at all, won't be corrupted even if the system should crash during writing. *flag* is ignored if *aURL* is of a type that cannot be written atomically.

Return Value

YES if the location is written successfully, otherwise NO.

Discussion

This method recursively validates that all the contained objects are property list objects (instances of NSData, NSDate, NSNumber, NSString, NSArray, or NSDictionary) before writing out the file, and returns NO if all the objects are not property list objects, since the resultant output would not be a valid property list.

If the receiver's contents are all property list objects, the location written by this method can be used to initialize a new dictionary with the class method dictionaryWithContentsOfURL: (page 500) or the instance method initWithContentsOfURL: (page 515).

For more information about property lists, see *Property List Programming Guide*.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSDictionary.h

NSDirectoryEnumerator Class Reference

Inherits from	NSEnumerator : NSObject
Conforms to	NSFastEnumeration (NSEnumerator) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSFileManager.h
Companion guide	Low-Level File Management Programming Topics
Related sample code	BundleLoader DeskPictAppDockMenu NSOperationSample

Overview

An `NSDirectoryEnumerator` object enumerates the contents of a directory, returning the pathnames of all files and directories contained within that directory. These pathnames are relative to the directory.

You obtain a directory enumerator using `NSFileManager`'s `enumeratorAtPath:` (page 644) method. For more details, see *Low-Level File Management Programming Topics*.

An enumeration is recursive, including the files of all subdirectories, and crosses device boundaries. An enumeration does not resolve symbolic links, or attempt to traverse symbolic links that point to directories.

Tasks

Getting File and Directory Attributes

`- directoryAttributes` (page 526)
: Returns an `NSDictionary` object that contains the attributes of the directory at which enumeration started.

`- fileAttributes` (page 526)
: Returns an `NSDictionary` object that contains the attributes of the most recently returned file or subdirectory (as referenced by the pathname).

Skipping Subdirectories

`- skipDescendents` (page 527)
Causes the receiver to skip recursion into the most recently obtained subdirectory.

Instance Methods

directoryAttributes

Returns an `NSDictionary` object that contains the attributes of the directory at which enumeration started.

```
- (NSDictionary *)directoryAttributes
```

Return Value
An `NSDictionary` object that contains the attributes of the directory at which enumeration started.

Discussion
See the description of the `fileAttributesAtPath:traverseLink:` (page 645) method of `NSFileManager` for details on obtaining the attributes from the dictionary.

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

See Also
`createDirectoryAtPath:attributes:` (page 637) (`NSFileManager`)

Declared In
`NSFileManager.h`

fileAttributes

Returns an `NSDictionary` object that contains the attributes of the most recently returned file or subdirectory (as referenced by the pathname).

```
- (NSDictionary *)fileAttributes
```

Return Value
An `NSDictionary` object that contains the attributes of the most recently returned file or subdirectory (as referenced by the pathname).

Discussion
See the description of the `fileAttributesAtPath:traverseLink:` (page 645) method of `NSFileManager` for details on obtaining the attributes from the dictionary.

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

Related Sample Code
NSOperationSample

Declared In
NSFileManager.h

skipDescendents

Causes the receiver to skip recursion into the most recently obtained subdirectory.

```
- (void)skipDescendents
```

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

Declared In
NSFileManager.h

NSDistantObject Class Reference

Inherits from	NSProxy
Conforms to	NSCoding NSObject (NSProxy)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSDistantObject.h
Companion guide	Distributed Objects Programming Topics

Overview

`NSDistantObject` is a concrete subclass of `NSProxy` that defines proxies for objects in other applications or threads. When a distant object receives a message, in most cases it forwards the message through its `NSConnection` object to the real object in another application, supplying the return value to the sender of the message if one is received, and propagating any exception back to the invoker of the method that raised it.

`NSDistantObject` adds two useful instance methods to those defined by `NSProxy`: `connectionForProxy` (page 532) returns the `NSConnection` object that handles the receiver; `setProtocolForProxy:` (page 533) establishes the set of methods the real object is known to respond to, saving the network traffic required to determine the argument and return types the first time a particular selector is forwarded to the remote proxy.

There are two kinds of distant object: local proxies and remote proxies. A local proxy is created by an `NSConnection` object the first time an object is sent to another application. It is used by the connection for bookkeeping purposes and should be considered private. The local proxy is transmitted over the network using the `NSCoding` protocol to create the remote proxy, which is the object that the other application uses. `NSDistantObject` defines methods for an `NSConnection` object to create instances, but they're intended only for subclasses to override—you should never invoke them directly. Use the `rootProxyForConnectionWithRegisteredName:host:` (page 332) method of `NSConnection`, which sets up all the required state for an object-proxy pair.

Important: `NSDistantObject` conforms to the `NSCoding` protocol, but only supports coding by an `NSPortCoder`. `NSDistantObject` and its subclasses do not support archiving.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 2034)
- `initWithCoder:` (page 2034)

Tasks

Creating a Local Proxy

`+ proxyWithLocal:connection:` (page 531)
Returns a local proxy for a given object and connection, creating the proxy if necessary.

`- initWithLocal:connection:` (page 532)
Initializes an `NSDistantObject` object as a local proxy for a given object.

Creating a Remote Proxy

`+ proxyWithTarget:connection:` (page 531)
Returns a remote proxy for a given object and connection, creating the proxy if necessary.

`- initWithTarget:connection:` (page 533)
Initializes a newly allocated NSDistantObject as a remote proxy for *`remoteObject`*, which is an `id` in another thread or another application's address space.

Getting a Proxy's NSConnection

`- connectionForProxy` (page 532)
Returns the connection used by the receiver.

Setting a Proxy's Protocol

`- setProtocolForProxy:` (page 533)
Sets the methods known to be handled by the receiver to those in a given protocol.

Class Methods

proxyWithLocal:connection:

Returns a local proxy for a given object and connection, creating the proxy if necessary.

```
+ (NSDistantObject *)proxyWithLocal:(id)anObject connection:(NSConnection
    *)aConnection
```

Parameters

anObject

An object in the receiver's address space.

aConnection

The connection for the returned proxy.

Return Value

A local proxy for *anObject* and *aConnection*, creating it if necessary.

Discussion

Other applications connect to the proxy using the `NSConnection` `connectionWithRegisteredName:host:` (page 330) class method.

Local proxies should be considered private to their `NSConnection` objects. Only an `NSConnection` object should use this method to create them, and your code shouldn't retain or otherwise use local proxies.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- initWithLocal:connection:` (page 532)

Declared In

`NSDistantObject.h`

proxyWithTarget:connection:

Returns a remote proxy for a given object and connection, creating the proxy if necessary.

```
+ (NSDistantObject *)proxyWithTarget:(id)remoteObject connection:(NSConnection
    *)aConnection
```

Parameters

remoteObject

An object in another thread or another application's address space.

aConnection

The connection to set as the `NSConnection` object for the returned proxy—it should have been created using the `NSConnection` `connectionWithRegisteredName:host:` (page 330) class method.

Return Value

A remote proxy for *remoteObject* and *aConnection*, creating the proxy if necessary

Discussion
A remote proxy cannot be used until its connection's peer has a local proxy representing *remoteObject* in the other application.

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

See Also
- initWithTarget:connection: (page 533)

Declared In
NSDistantObject.h

Instance Methods

connectionForProxy

Returns the connection used by the receiver.

```
- (NSConnection *)connectionForProxy
```

Return Value
The connection used by the receiver.

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

Declared In
NSDistantObject.h

initWithLocal:connection:

Initializes an NSDistantObject object as a local proxy for a given object.

```
- (id)initWithLocal:(id)anObject connection:(NSConnection *)aConnection
```

Parameters

anObject
An object in the receiver's address space.

aConnection
The connection for the returned proxy.

Return Value
An initialized NSDistantObject object that serves as a local proxy for *anObject*. If a proxy for *anObject* and *aConnection* already exists, the receiver is released and the existing proxy is retained and returned.

Discussion
Other applications connect to the proxy using the NSConnectionconnectionWithRegisteredName:host: (page 330) class method.

Local proxies should be considered private to their `NSConnection` objects. Only an `NSConnection` object should use this method to create them, and your code shouldn't retain or otherwise use local proxies.

This is the designated initializer for local proxies. It returns an initialized object, which might be different than the original receiver

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

See Also
`+ proxyWithLocal:connection:` (page 531)

Declared In
`NSDistantObject.h`

initWithTarget:connection:

Initializes a newly allocated NSDistantObject as a remote proxy for *`remoteObject`*, which is an `id` in another thread or another application's address space.

```
- (id)initWithTarget:(id)remoteObject connection:(NSConnection *)aConnection
```

Parameters

`remoteObject`
An object in another thread or another application's address space.

`aConnection`
The connection to set as the `NSConnection` object for the returned proxy—it should have been created using the `NSConnectionconnectionWithRegisteredName:host:` (page 330) class method.

Return Value
An `NSDistantObject` object initialized as a remote proxy for *`remoteObject`*. If a proxy for *`remoteObject`* and *`aConnection`* already exists, the receiver is released and the existing proxy is retained and returned.

Discussion
A remote proxy can't be used until its connection's peer has a local proxy representing *`remoteObject`* in the other application.

This is the designated initializer for remote proxies. It returns an initialized object, which might be different than the original receiver.

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

See Also
`+ proxyWithTarget:connection:` (page 531)

Declared In
`NSDistantObject.h`

setProtocolForProxy:

Sets the methods known to be handled by the receiver to those in a given protocol.

```
- (void)setProtocolForProxy:(Protocol *)aProtocol
```

Parameters

aProtocol

The protocol for the receiver.

Discussion

Setting a protocol for a remote proxy reduces network traffic needed to determine method argument and return types.

In order to encode a message's arguments for transmission over the network, the types of those arguments must be known in advance. When they're not known, the distributed objects system must send an initial message just to get those types, doubling the network traffic for every new message sent. Setting a protocol alleviates this need for methods defined by the protocol. You can still send messages that aren't declared in *aProtocol*—in this case the initial message is sent to determine the types, and then the real message is sent.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

SimpleThreads

Declared In

`NSDistantObject.h`

NSDistantObjectRequest Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSConnection.h
Companion guide	Distributed Objects Programming Topics

Overview

`NSDistantObjectRequest` objects are used by the distributed objects system to help handle invocations between different processes. You should never create `NSDistantObjectRequest` objects directly. Unless you are getting involved with the low-level details of distributed objects, there should never be a need to access an `NSDistantObjectRequest`. To intercept and possibly process requests yourself, implement the `NSConnection` delegate method `connection:handleRequest:` (page 350).

Tasks

Getting Information About a Request

- `connection` (page 536)
  Returns the `NSConnection` object involved in the request.
- `conversation` (page 536)
  Returns the token object representing the conversation in which the receiver was created.
- `invocation` (page 536)
  Returns the `NSInvocation` object for the request.

Raising a Remote Exception

- `replyWithException:` (page 537)
  Sends a reply back to the remote object making the distant object request.

Instance Methods

connection

Returns the `NSConnection` object involved in the request.

```
- (NSConnection *)connection
```

Return Value
The `NSConnection` object involved in the request.

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

Declared In
`NSConnection.h`

conversation

Returns the token object representing the conversation in which the receiver was created.

```
- (id)conversation
```

Return Value
The token object representing the conversation in which the receiver was created.

Discussion
If both ends of the distributed objects connection has `independentConversationQueueing` (page 336) set to `NO` (the default), the conversation object is always `nil`. Otherwise, it is either a proxy (or a copy) of the object created by the sender of the message or a locally created object, depending which end of the connection has independent queueing on.

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

See Also
`createConversationForConnection:` (page 351) (NSConnection)

Declared In
`NSConnection.h`

invocation

Returns the `NSInvocation` object for the request.

```
- (NSInvocation *)invocation
```

Return Value
The `NSInvocation` object for the request.

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

Declared In
`NSConnection.h`

replyWithException:

Sends a reply back to the remote object making the distant object request.

```
- (void)replyWithException:(NSException *)exception
```

Parameters

exception
The exception to send.

Discussion
If *exception* is `nil`, the return value of the receiver's invocation is sent; otherwise, *exception* is sent and is automatically raised when it arrives at its destination.

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

Declared In
`NSConnection.h`

NSDistributedLock Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSDistributedLock.h
Companion guide	Threading Programming Guide

Overview

The `NSDistributedLock` class defines an object that multiple applications on multiple hosts can use to restrict access to some shared resource, such as a file.

The lock is implemented by an entry (such as a file or directory) in the file system. For multiple applications to use an `NSDistributedLock` object to coordinate their activities, the lock must be writable on a file system accessible to all hosts on which the applications might be running.

Use the `tryLock` (page 542) method to attempt to acquire a lock. You should generally use the `unlock` (page 543) method to release the lock rather than `breakLock` (page 541).

`NSDistributedLock` doesn't conform to the `NSLocking` protocol, nor does it have a `lock` method. The protocol's `lock` (page 2091) method is intended to block the execution of the thread until successful. For an `NSDistributedLock` object, this could mean polling the file system at some predetermined rate. A better solution is to provide the `tryLock` (page 542) method and let you determine the polling frequency that makes sense for your application.

Tasks

Creating an NSDistributedLock

`+ lockWithPath:` (page 540)
Returns an `NSDistributedLock` object initialized to use as the locking object the file-system entry specified by a given path.

- `initWithPath:` (page 541)
  Initializes an `NSDistributedLock` object to use as the lock the file-system entry specified by a given path.

Acquiring a Lock

- `tryLock` (page 542)
  Attempts to acquire the receiver and immediately returns a Boolean value that indicates whether the attempt was successful.

Relinquishing a Lock

- `breakLock` (page 541)
  Forces the lock to be relinquished.
- `unlock` (page 543)
  Relinquishes the receiver.

Getting Lock Information

- `lockDate` (page 542)
  Returns the time the receiver was acquired by any of the `NSDistributedLock` objects using the same path.

Class Methods

lockWithPath:

Returns an `NSDistributedLock` object initialized to use as the locking object the file-system entry specified by a given path.

```
+ (NSDistributedLock *)lockWithPath:(NSString *)aPath
```

Parameters

aPath

All of *aPath* up to the last component itself must exist. You can use `NSFileManager` to create (and set permissions) for any nonexistent intermediate directories.

Return Value

An `NSDistributedLock` object initialized to use as the locking object the file-system entry specified by *aPath*.

Discussion

For applications to use the lock, *aPath* must be accessible to—and writable by—all hosts on which the applications might be running.

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

See Also
- `initWithPath:` (page 541)

Declared In
`NSDistributedLock.h`

Instance Methods

breakLock

Forces the lock to be relinquished.

```
- (void)breakLock
```

Discussion
This method always succeeds unless the lock has been damaged. If another process has already unlocked or broken the lock, this method has no effect. You should generally use `unlock` (page 543) rather than `breakLock` to relinquish a lock.

Warning: Because `breakLock` can release another process's lock, it should be used with great caution.

Even if you break a lock, there's no guarantee that you will then be able to acquire the lock—another process might get it before your `tryLock` (page 542) is invoked.

Raises an `NSGenericException` if the lock could not be removed.

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

See Also
- `unlock` (page 543)

Declared In
`NSDistributedLock.h`

initWithPath:

Initializes an `NSDistributedLock` object to use as the lock the file-system entry specified by a given path.

```
- (id)initWithPath:(NSString *)aPath
```

Parameters

aPath

All of *aPath* up to the last component itself must exist. You can use `NSFileManager` to create (and set permissions) for any nonexistent intermediate directories.

Return Value
An `NSDistributedLock` object initialized to use as the locking object the file-system entry specified by *aPath*.

Discussion
For applications to use the lock, *aPath* must be accessible to—and writable by—all hosts on which the applications might be running.

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

See Also
`+ lockWithPath:` (page 540)

Declared In
`NSDistributedLock.h`

lockDate

Returns the time the receiver was acquired by any of the `NSDistributedLock` objects using the same path.

```
- (NSDate *)lockDate
```

Return Value
The time the receiver was acquired by any of the `NSDistributedLock` objects using the same path. Returns `nil` if the lock doesn't exist.

Discussion
This method is potentially useful to applications that want to use an age heuristic to decide if a lock is too old and should be broken.

If the creation date on the lock isn't the date on which you locked it, you've lost the lock: it's been broken since you last checked it.

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

Declared In
`NSDistributedLock.h`

tryLock

Attempts to acquire the receiver and immediately returns a Boolean value that indicates whether the attempt was successful.

```
- (BOOL)tryLock
```

Return Value
`YES` if the attempt to acquire the receiver was successful, otherwise `NO`.

Discussion
Raises `NSGenericException` if a file-system error occurs.

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

See Also
- `unlock` (page 543)

Declared In
`NSDistributedLock.h`

unlock

Relinquishes the receiver.

```
- (void)unlock
```

Discussion
You should generally use the `unlock` method rather than `breakLock` (page 541) to release a lock.

An `NSGenericException` is raised if the receiver doesn't already exist.

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

See Also
- `breakLock` (page 541)

Declared In
`NSDistributedLock.h`

NSDistributedNotificationCenter Class Reference

Inherits from	NSNotificationCenter : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSDistributedNotificationCenter.h
Companion guide	Notification Programming Topics for Cocoa
Related sample code	StickiesExample

Class at a Glance

The `NSDistributedNotificationCenter` class provides a way to send notifications to objects in other tasks. It takes `NSNotification` objects and broadcasts them to any objects in other tasks that have registered for the notification with their task's default distributed notification center.

Principal Attributes

- **Notification dispatch table.** See "Class at a Glance" > "Principal Attributes" in *NSNotificationCenter Class Reference* for information about the dispatch table.

  In addition to the notification name and sender, dispatch table entries for distributed notification centers specify when the notification center delivers notifications to its observers. See the `postNotificationName:object:userInfo:deliverImmediately:` (page 551) method, "Suspending and Resuming Notification Delivery" (page 547), and `NSNotificationSuspensionBehavior` (page 555) for details.

Commonly Used Methods

`defaultCenter` (page 547)
: Accesses the default distributed notification center.

`addObserver:selector:name:object:suspensionBehavior:` (page 549)
: Registers an object to receive a notification with a specified behavior when notification delivery is suspended.

`postNotificationName:object:userInfo:deliverImmediately:` (page 551)
Creates and posts a notification.

`removeObserver:name:object:` (page 552)
Specifies that an object no longer wants to receive certain notifications.

Overview

The `NSDistributedNotificationCenter` class implements a notification center that can distribute notifications asynchronously to tasks other than the one in which the notification was posted. An instance of this class are known as a **distributed notification center**.

Each task has a default distributed notification center that you access with the `defaultCenter` (page 547) class method. There may be different types of distributed notification centers. Currently there is a single type—`NSLocalNotificationCenterType`. This type of distributed notification center handles notifications that can be sent between tasks on a single computer. For communication between tasks on different computers, use *Distributed Objects Programming Topics*.

Posting a *distributed notification* is an expensive operation. The notification gets sent to a system-wide server that distributes it to all the tasks that have objects registered for distributed notifications. The latency between posting the notification and the notification's arrival in another task is unbounded. In fact, when too many notifications are posted and the server's queue fills up, notifications may be dropped.

Distributed notifications are delivered via a task's run loop. A task must be running a run loop in one of the "common" modes, such as `NSDefaultRunLoopMode`, to receive a distributed notification. For multithreaded applications running in Mac OS X v10.3 and later, distributed notifications are always delivered to the main thread. For multithreaded applications running in Mac OS X v10.2.8 and earlier, notifications are delivered to the thread that first used the distributed notifications API, which in most cases is the main thread.

Note: `NSDistributedNotificationCenter` objects should not be used to send notifications between threads within the same task. Use *Distributed Objects Programming Topics* or the `NSObject` method `performSelectorOnMainThread:withObject:waitUntilDone:` (page 1188), instead. You can also setup an `NSPort` object to receive and distribute messages from other threads.

Tasks

Getting Distributed Notification Centers

`+ defaultCenter` (page 547)
Returns the default distributed notification center, representing the local notification center for the computer.

`+ notificationCenterForType:` (page 548)
Returns the distributed notification center for a particular notification center type.

Managing Observers

- `addObserver:selector:name:object:` (page 548)
  Adds an entry to the receiver's dispatch table with an observer, a notification selector and optional criteria: notification name and sender.
- `addObserver:selector:name:object:suspensionBehavior:` (page 549)
  Adds an entry to the receiver's dispatch table with a specific observer and suspended-notifications behavior, and optional notification name and sender.
- `removeObserver:name:object:` (page 552)
  Removes matching entries from the receiver's dispatch table.

Posting Notifications

- `postNotificationName:object:` (page 550)
  Creates a notification, and posts it to the receiver.
- `postNotificationName:object:userInfo:` (page 550)
  Creates a notification with information, and posts it to the receiver.
- `postNotificationName:object:userInfo:deliverImmediately:` (page 551)
  Creates a notification with information and an immediate-delivery specifier, and posts it to the receiver.
- `postNotificationName:object:userInfo:options:` (page 552)
  Creates a notification with information, and posts it to the receiver.

Suspending and Resuming Notification Delivery

- `suspended` (page 553)
  Returns a Boolean value that indicates whether notification delivery is suspended.
- `setSuspended:` (page 553)
  Suspends or resumes notification delivery.

Class Methods

defaultCenter

Returns the default distributed notification center, representing the local notification center for the computer.

```
+ (id)defaultCenter
```

Return Value
Default distributed notification center for the computer.

Discussion
This method calls `notificationCenterForType:` (page 548) with an argument of `NSLocalNotificationCenterType`.

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

Related Sample Code
StickiesExample

Declared In
`NSDistributedNotificationCenter.h`

notificationCenterForType:

Returns the distributed notification center for a particular notification center type.

```
+ (NSDistributedNotificationCenter *)notificationCenterForType:(NSString
    *)notificationCenterType
```

Parameters

notificationCenterType
Notification center type being inquired about.

Return Value
Distributed notification center for *notificationCenterType*.

Discussion
Currently only one type, `NSLocalNotificationCenterType`, is supported.

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

Declared In
`NSDistributedNotificationCenter.h`

Instance Methods

addObserver:selector:name:object:

Adds an entry to the receiver's dispatch table with an observer, a notification selector and optional criteria: notification name and sender.

```
- (void)addObserver:(id)notificationObserver selector:(SEL)notificationSelector
    name:(NSString *)notificationName object:(NSString *)notificationSender
```

Parameters

notificationObserver
Object registering as an observer. Must not be `nil`.

notificationSelector
Selector that specifies the message the receiver sends *notificationObserver* to notify it of the notification posting. Must not be `0`.

notificationName
The name of the notification for which to register the observer; that is, only notifications with this name are delivered to the observer. When `nil`, the notification center doesn't use a notification's name to decide whether to deliver it to the observer.

notificationSender
The object whose notifications the observer wants to receive; that is, only notifications sent by this sender are delivered to the observer. When `nil`, the notification center doesn't use a notification's sender to decide whether to deliver it to the observer.

Discussion
This method calls `addObserver:selector:name:object:suspensionBehavior:` (page 549) with `suspensionBehavior:NSNotificationSuspensionBehaviorCoalesce` (described in "Constants" (page 554)).

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

Declared In
`NSDistributedNotificationCenter.h`

addObserver:selector:name:object:suspensionBehavior:

Adds an entry to the receiver's dispatch table with a specific observer and suspended-notifications behavior, and optional notification name and sender.

```
- (void)addObserver:(id)notificationObserver selector:(SEL)notificationSelector
    name:(NSString *)notificationName object:(NSString *)notificationSender
    suspensionBehavior:(NSNotificationSuspensionBehavior)suspendedDeliveryBehavior
```

Parameters

notificationObserver
Object registering as an observer. Must not be `nil`.

notificationSelector
Selector that specifies the message the receiver sends *notificationObserver* to notify it of the notification posting. Must not be `0`.

notificationName
The name of the notification for which to register the observer; that is, only notifications with this name are delivered to the observer. When `nil`, the notification center doesn't use a notification's name to decide whether to deliver it to the observer.

notificationSender
The object whose notifications the observer wants to receive; that is, only notifications sent by this sender are delivered to the observer. When `nil`, the notification center doesn't use a notification's sender to decide whether to deliver it to the observer.

suspendedDeliveryBehavior
Notification posting behavior when notification delivery is suspended.

Discussion
The receiver does not retain *notificationObserver*. Therefore, you should always send `removeObserver:` (page 1043) or `removeObserver:name:object:` (page 552) to the receiver before releasing *notificationObserver*.

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

See Also
- `postNotificationName:object:userInfo:deliverImmediately:` (page 551)

Declared In
`NSDistributedNotificationCenter.h`

postNotificationName:object:

Creates a notification, and posts it to the receiver.

```
- (void)postNotificationName:(NSString *)notificationName object:(NSString
    *)notificationSender
```

Parameters

notificationName
Name of the notification to post. Must not be `nil`.

notificationSender
Sender of the notification. May be `nil`.

Discussion
This method invokes `postNotificationName:object:userInfo:deliverImmediately:` (page 551) with `userInfo:nil deliverImmediately:NO`.

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

See Also
- `postNotificationName:object:userInfo:` (page 550)
- `postNotificationName:object:userInfo:deliverImmediately:` (page 551)
- `postNotificationName:object:userInfo:options:` (page 552)

Declared In
`NSDistributedNotificationCenter.h`

postNotificationName:object:userInfo:

Creates a notification with information, and posts it to the receiver.

```
- (void)postNotificationName:(NSString *)notificationName object:(NSString
    *)notificationSender userInfo:(NSDictionary *)notificationInfo
```

Parameters

notificationName
Name of the notification to post. Must not be `nil`.

notificationSender
Sender of the notification. May be `nil`.

notificationInfo
Dictionary containing additional information. May be `nil`.

Discussion
This method invokes `postNotificationName:object:userInfo:deliverImmediately:` (page 551) with `deliverImmediately:NO`.

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

See Also
- `postNotificationName:object:` (page 550)
- `postNotificationName:object:userInfo:deliverImmediately:` (page 551)
- `postNotificationName:object:userInfo:options:` (page 552)

Declared In
`NSDistributedNotificationCenter.h`

postNotificationName:object:userInfo:deliverImmediately:

Creates a notification with information and an immediate-delivery specifier, and posts it to the receiver.

```
- (void)postNotificationName:(NSString *)notificationName object:(NSString
    *)notificationSender userInfo:(NSDictionary *)userInfo
    deliverImmediately:(BOOL)deliverImmediately
```

Parameters

notificationName
Name of the notification to post. Must not be `nil`.

notificationSender
Sender of the notification. May be `nil`.

userInfo
Dictionary containing additional information. May be `nil`.

deliverImmediately
Specifies when to deliver the notification. When `NO`, the receiver delivers notifications to their observers according to the suspended-notification behavior specified in the corresponding dispatch table entry. When `YES`, the receiver delivers the notification immediately to its observers.

Discussion
This is the preferred method for posting notifications.

The *notificationInfo* dictionary is serialized as a property list, so it can be passed to another task. In the receiving task, it is deserialized back into a dictionary. This serialization imposes some restrictions on the objects that can be placed in the *notificationInfo* dictionary. See XML Property Lists for details.

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

See Also
- `postNotificationName:object:` (page 550)
- `postNotificationName:object:userInfo:` (page 550)
- `postNotificationName:object:userInfo:options:` (page 552)
- `encodeRootObject:` (page 101) (NSArchiver)
+ `unarchiveObjectWithData:` (page 1684) (NSUnarchiver)

Declared In
NSDistributedNotificationCenter.h

postNotificationName:object:userInfo:options:

Creates a notification with information, and posts it to the receiver.

```
- (void)postNotificationName:(NSString *)notificationName object:(NSString
    *)notificationSender userInfo:(NSDictionary *)userInfo
    options:(NSUInteger)notificationOptions
```

Parameters

notificationName
Name of the notification to post. Must not be `nil`.

notificationSender
Sender of the notification. May be `nil`.

userInfo
Dictionary containing additional information. May be `nil`.

notificationOptions
Specifies how the notification is posted to the task and when to deliver it to its observers. See "Notification Posting Behavior" (page 554) for details.

Discussion
The *userInfo* dictionary is serialized as a property list, so it can be passed to another task. In the receiving task, it is deserialized back into a dictionary. This serialization imposes some restrictions on the objects that can be placed in the *notificationInfo* dictionary. See XML Property Lists for details.

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

See Also
- `postNotificationName:object:` (page 550)
- `postNotificationName:object:userInfo:` (page 550)
- `postNotificationName:object:userInfo:deliverImmediately:` (page 551)

Declared In
NSDistributedNotificationCenter.h

removeObserver:name:object:

Removes matching entries from the receiver's dispatch table.

```
- (void)removeObserver:(id)notificationObserver name:(NSString *)notificationName
    object:(NSString *)notificationSender
```

Parameters

notificationObserver
Observer to remove from the dispatch table. Specify an observer to remove only entries for this observer. When `nil`, the receiver does not use notification observers as criteria for removal.

notificationName
Name of the notification to remove from dispatch table. Specify a notification name to remove only entries that specify this notification name. When `nil`, the receiver does not use notification names as criteria for removal.

notificationSender
Sender to remove from the dispatch table. Specify a notification sender to remove only entries that specify this sender. When `nil`, the receiver does not use notification senders as criteria for removal.

Discussion
Be sure to invoke this method with `notificationName:nil notificationSender:nil` (or `removeObserver:` (page 1043)) before deallocating the observer object.

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

Declared In
`NSDistributedNotificationCenter.h`

setSuspended:

Suspends or resumes notification delivery.

```
- (void)setSuspended:(BOOL)suspended
```

Parameters
suspended
`YES` suspends notification delivery, `NO` resumes it.

Discussion
See `NSNotificationSuspensionBehavior` (page 555) for details on how the receiver delivers notifications to their observers when normal notification delivery is suspended.

The `NSApplication` class automatically suspends distributed notification delivery when the application is not active. Applications based on the Application Kit framework should let AppKit manage the suspension of notification delivery. Foundation-only programs may have occasional need to use this method.

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

See Also
- `addObserver:selector:name:object:suspensionBehavior:` (page 549)
- `postNotificationName:object:userInfo:deliverImmediately:` (page 551)
- `suspended` (page 553)

Declared In
`NSDistributedNotificationCenter.h`

suspended

Returns a Boolean value that indicates whether notification delivery is suspended.

```
- (BOOL)suspended
```

Return Value
`YES` when notification delivery is suspended, otherwise `NO`.

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

See Also
`- setSuspended:` (page 553)

Declared In
`NSDistributedNotificationCenter.h`

Constants

Notification Center Type

This constant specifies the notification center type.

```
FOUNDATION_EXPORT NSString * const NSLocalNotificationCenterType;
```

Constants
`NSLocalNotificationCenterType`
Distributes notifications to all tasks on the sender's computer.

Available in Mac OS X v10.0 and later.

Declared in `NSDistributedNotificationCenter.h`.

Declared In
`NSDistributedNotificationCenter.h`

Notification Posting Behavior

These constants specify the behavior of notifications posted using the `postNotificationName:object:userInfo:options:` (page 552) method.

```
enum {
    NSNotificationDeliverImmediately = (1 << 0),
    NSNotificationPostToAllSessions = (1 << 1)
};
```

Constants
`NSNotificationDeliverImmediately`
When set, the notification is delivered immediately to all observers, regardless of their suspension behavior or suspension state. When not set, allows the normal suspension behavior of notification observers to take place.

Available in Mac OS X v10.3 and later.

Declared in `NSDistributedNotificationCenter.h`.

`NSNotificationPostToAllSessions`
When set, the notification is posted to all sessions. When not set, the notification is sent only to applications within the same login session as the posting task.

Available in Mac OS X v10.3 and later.

Declared in `NSDistributedNotificationCenter.h`.

Declared In
`NSDistributedNotificationCenter.h`

NSNotificationSuspensionBehavior

These constants specify the types of notification delivery suspension behaviors.

```
typedef enum {
    NSNotificationSuspensionBehaviorDrop = 1,
    NSNotificationSuspensionBehaviorCoalesce = 2,
    NSNotificationSuspensionBehaviorHold = 3,
    NSNotificationSuspensionBehaviorDeliverImmediately = 4
} NSNotificationSuspensionBehavior;
```

Constants
`NSNotificationSuspensionBehaviorDrop`
The server does not queue any notifications with this name and object until `setSuspended:` (page 553) with an argument of `NO` is called.

Available in Mac OS X v10.0 and later.

Declared in `NSDistributedNotificationCenter.h`.

`NSNotificationSuspensionBehaviorCoalesce`
The server only queues the last notification of the specified name and object; earlier notifications are dropped. In cover methods for which suspension behavior is not an explicit argument, `NSNotificationSuspensionBehaviorCoalesce` is the default.

Available in Mac OS X v10.0 and later.

Declared in `NSDistributedNotificationCenter.h`.

`NSNotificationSuspensionBehaviorHold`
The server holds all matching notifications until the queue has been filled (queue size determined by the server), at which point the server may flush queued notifications.

Available in Mac OS X v10.0 and later.

Declared in `NSDistributedNotificationCenter.h`.

`NSNotificationSuspensionBehaviorDeliverImmediately`
The server delivers notifications matching this registration irrespective of whether `setSuspended:` (page 553) with an argument of `YES` has been called. When a notification with this suspension behavior is matched, it has the effect of first flushing any queued notifications. The effect is as if `setSuspended:` (page 553) with an argument of `NO` were first called if the application is suspended, followed by the notification in question being delivered, followed by a transition back to the previous suspended or unsuspended state.

Available in Mac OS X v10.0 and later.

Declared in `NSDistributedNotificationCenter.h`.

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

Declared In
`NSDistributedNotificationCenter.h`

NSEnumerator Class Reference

Inherits from	NSObject
Conforms to	NSFastEnumeration NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSEnumerator.h
Companion guide	Collections Programming Topics for Cocoa
Related sample code	CoreRecipes iSpend LSMSmartCategorizer SimpleCalendar StickiesExample

Overview

`NSEnumerator` is an abstract class, instances of whose subclasses enumerate collections of other objects, such as arrays and dictionaries.

All creation methods are defined in the collection classes—such as `NSArray`, `NSSet`, and `NSDictionary`—which provide special `NSEnumerator` objects with which to enumerate their contents. For example, `NSArray` has two methods that return an `NSEnumerator` object: `objectEnumerator` (page 1458) and `reverseObjectEnumerator` (page 134). `NSDictionary` also has two methods that return an `NSEnumerator` object: `keyEnumerator` (page 519) and `objectEnumerator` (page 520). These methods let you enumerate the contents of a dictionary by key or by value, respectively.

You send `nextObject` (page 558) repeatedly to a newly created `NSEnumerator` object to have it return the next object in the original collection. When the collection is exhausted, `nil` is returned. You cannot "reset" an enumerator after it has exhausted its collection. To enumerate a collection again, you need a new enumerator.

The enumerator subclasses used by `NSArray`, `NSDictionary`, and `NSSet` retain the collection during enumeration. When the enumeration is exhausted, the collection is released.

Note: It is not safe to modify a mutable collection while enumerating through it. Some enumerators may currently allow enumeration of a collection that is modified, but this behavior is not guaranteed to be supported in the future.

Tasks

Getting the Enumerated Objects

- `allObjects` (page 558)
  Returns an array of objects the receiver has yet to enumerate.
- `nextObject` (page 558)
  Returns the next object from the collection being enumerated.

Instance Methods

allObjects

Returns an array of objects the receiver has yet to enumerate.

```
- (NSArray *)allObjects
```

Return Value
An array of objects the receiver has yet to enumerate.

Discussion
Put another way, the array returned by this method does not contain objects that have already been enumerated with previous `nextObject` (page 558) messages.

Invoking this method exhausts the enumerator's collection so that subsequent invocations of `nextObject` return `nil`.

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

Declared In
`NSEnumerator.h`

nextObject

Returns the next object from the collection being enumerated.

```
- (id)nextObject
```

Return Value
The next object from the collection being enumerated, or `nil` when all objects have been enumerated.

Discussion
The following code illustrates how this method works using an array:

```
NSArray *anArray = // ... ;
NSEnumerator *enumerator = [anArray objectEnumerator];
id object;

while ((object = [enumerator nextObject])) {
    // do something with object...
}
```

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

Related Sample Code
ColorSyncDevices-Cocoa
QTAudioExtractionPanel
SillyFrequencyLevels
SimpleCalendar
StickiesExample

Declared In
`NSEnumerator.h`

NSError Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSError.h Foundation/NSURLError.h
Availability	Available in Mac OS X v10.2 with Safari 1.0 installed. Available in Mac OS X v10.2.7 and later. Available in iPhone OS 2.0 and later.
Companion guide	Error Handling Programming Guide For Cocoa
Related sample code	CoreRecipes CustomAtomicStoreSubclass iSpend Quartz Composer WWDC 2005 TextEdit TextEditPlus

Overview

An `NSError` object encapsulates richer and more extensible error information than is possible using only an error code or error string. The core attributes of an `NSError` object are an error domain (represented by a string), a domain-specific error code and a user info dictionary containing application specific information.

Several well-known domains are defined corresponding to Mach, POSIX, and `OSStatus` errors. Foundation error codes are found in the Cocoa error domain and documented in the *Foundation Constants Reference*. In addition, `NSError` allows you to attach an arbitrary user info dictionary to an error object, and provides the means to return a human-readable description for the error.

`NSError` is not an abstract class, and can be used directly. Applications may choose to create subclasses of `NSError` to provide better localized error strings by overriding `localizedDescription` (page 565).

In general, a method should signal an error condition by—for example—returning `NO` or `nil` rather than by the simple presence of an error object. The method can then optionally return an `NSError` object by reference, in order to further describe the error.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 2034)
- `initWithCoder:` (page 2034)

NSCopying
- `copyWithZone:` (page 2042)

Tasks

Creating Error Objects

`+ errorWithDomain:code:userInfo:` (page 563)
Creates and initializes an `NSError` object for a given domain and code with a given `userInfo` dictionary.

`- initWithDomain:code:userInfo:` (page 565)
Returns an `NSError` object initialized for a given domain and code with a given `userInfo` dictionary.

Getting Error Properties

`- code` (page 563)
Returns the receiver's error code.

`- domain` (page 564)
Returns the receiver's error domain.

`- userInfo` (page 568)
Returns the receiver's user info dictionary.

Getting a Localized Error Description

`- localizedDescription` (page 565)
Returns a string containing the localized description of the error.

`- localizedRecoveryOptions` (page 567)
Returns an array containing the localized titles of buttons appropriate for displaying in an alert panel.

`- localizedRecoverySuggestion` (page 567)
Returns a string containing the localized recovery suggestion for the error.

`- localizedFailureReason` (page 566)
Returns a string containing the localized explanation of the reason for the error.

Getting the Error Recovery Attempter

`- recoveryAttempter` (page 567)
Returns an object that conforms to the `NSErrorRecoveryAttempting` informal protocol.

Class Methods

errorWithDomain:code:userInfo:

Creates and initializes an `NSError` object for a given domain and code with a given `userInfo` dictionary.

```
+ (id)errorWithDomain:(NSString *)domain code:(NSInteger)code userInfo:(NSDictionary
    *)dict
```

Parameters

domain
The error domain—this can be one of the predefined `NSError` domains, or an arbitrary string describing a custom domain. *domain* must not be `nil`.

code
The error code for the error.

dict
The `userInfo` dictionary for the error. *userInfo* may be `nil`.

Return Value
An `NSError` object for *domain* with the specified error *code* and the dictionary of arbitrary data *userInfo*.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.
Available in iPhone OS 2.0 and later.

Related Sample Code
CoreRecipes
CustomAtomicStoreSubclass
iSpend
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSError.h`

Instance Methods

code

Returns the receiver's error code.

```
- (NSInteger)code
```

Return Value
The receiver's error code.

Discussion
Note that errors are domain specific.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.
Available in iPhone OS 2.0 and later.

See Also
- `localizedDescription` (page 565)
- `domain` (page 564)
- `userInfo` (page 568)

Related Sample Code
Core Data HTML Store
Departments and Employees
ExtractMovieAudioToAIFF
QTExtractAndConvertToAIFF
QTExtractAndConvertToMovieFile

Declared In
`NSError.h`

domain

Returns the receiver's error domain.

```
- (NSString *)domain
```

Return Value
A string containing the receiver's error domain.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.
Available in iPhone OS 2.0 and later.

See Also
- `code` (page 563)
- `localizedDescription` (page 565)
- `userInfo` (page 568)

Related Sample Code
Departments and Employees

Declared In
`NSError.h`

initWithDomain:code:userInfo:

Returns an `NSError` object initialized for a given domain and code with a given `userInfo` dictionary.

```
- (id)initWithDomain:(NSString *)domain code:(NSInteger)code userInfo:(NSDictionary
    *)dict
```

Parameters

domain

The error domain—this can be one of the predefined `NSError` domains, or an arbitrary string describing a custom domain. *domain* must not be `nil`.

code

The error code for the error.

dict

The `userInfo` dictionary for the error. *userInfo* may be `nil`.

Return Value
An `NSError` object initialized for *domain* with the specified error *code* and the dictionary of arbitrary data *userInfo*.

Discussion
This is the designated initializer for `NSError`.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.
Available in iPhone OS 2.0 and later.

See Also
`+ errorWithDomain:code:userInfo:` (page 563)

Related Sample Code
BindingsJoystick
CocoaEcho
CocoaHTTPServer
CocoaSOAP

Declared In
`NSError.h`

localizedDescription

Returns a string containing the localized description of the error.

```
- (NSString *)localizedDescription
```

Return Value
A string containing the localized description of the error.

By default this method returns the object in the user info dictionary for the key `NSLocalizedDescriptionKey`. If the user info dictionary doesn't contain a value for `NSLocalizedDescriptionKey`, a default string is constructed from the domain and code.

Discussion
This method can be overridden by subclasses to present customized error strings.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.
Available in iPhone OS 2.0 and later.

See Also
- `code` (page 563)
- `domain` (page 564)
- `userInfo` (page 568)

Related Sample Code
CoreRecipes
Departments and Employees
NewsReader

Declared In
`NSError.h`

localizedFailureReason

Returns a string containing the localized explanation of the reason for the error.

```
- (NSString *)localizedFailureReason
```

Return Value
A string containing the localized explanation of the reason for the error. By default this method returns the object in the user info dictionary for the key `NSLocalizedFailureReasonErrorKey`.

Discussion
This method can be overridden by subclasses to present customized error strings.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `code` (page 563)
- `domain` (page 564)
- `userInfo` (page 568)

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSError.h`

localizedRecoveryOptions

Returns an array containing the localized titles of buttons appropriate for displaying in an alert panel.

```
- (NSArray *)localizedRecoveryOptions
```

Return Value
An array containing the localized titles of buttons appropriate for displaying in an alert panel. By default this method returns the object in the user info dictionary for the key `NSLocalizedRecoveryOptionsErrorKey`. If the user info dictionary doesn't contain a value for `NSLocalizedRecoveryOptionsErrorKey`, `nil` is returned.

Discussion
The first string is the title of the right-most and default button, the second the one to the left of that, and so on. The recovery options should be appropriate for the recovery suggestion returned by `localizedRecoverySuggestion` (page 567). If the user info dictionary doesn't contain a value for `NSLocalizedRecoveryOptionsErrorKey`, only an OK button is displayed.

This method can be overridden by subclasses to present customized recovery suggestion strings.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSError.h`

localizedRecoverySuggestion

Returns a string containing the localized recovery suggestion for the error.

```
- (NSString *)localizedRecoverySuggestion
```

Return Value
A string containing the localized recovery suggestion for the error. By default this method returns the object in the user info dictionary for the key `NSLocalizedRecoverySuggestionErrorKey`. If the user info dictionary doesn't contain a value for `NSLocalizedRecoverySuggestionErrorKey`, `nil` is returned.

Discussion
The returned string is suitable for displaying as the secondary message in an alert panel.

This method can be overridden by subclasses to present customized recovery suggestion strings.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSError.h`

recoveryAttempter

Returns an object that conforms to the `NSErrorRecoveryAttempting` informal protocol.

```
- (id)recoveryAttempter
```

Return Value
An object that conforms to the `NSErrorRecoveryAttempting` informal protocol. By default this method returns the object for the user info dictionary for the key `NSRecoveryAttempterErrorKey`. If the user info dictionary doesn't contain a value for `NSRecoveryAttempterErrorKey`, `nil` is returned.

Discussion
The recovery attempter must be an object that can correctly interpret an index into the array returned by `localizedRecoveryOptions` (page 567).

Availability
Available in Mac OS X v10.4 and later.

See Also
- `localizedRecoveryOptions` (page 567)

Declared In
`NSError.h`

userInfo

Returns the receiver's user info dictionary.

```
- (NSDictionary *)userInfo
```

Return Value
The receiver's user info dictionary, or `nil` if the user info dictionary has not been set.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.
Available in iPhone OS 2.0 and later.

See Also
- `code` (page 563)
- `domain` (page 564)
- `localizedDescription` (page 565)

Related Sample Code
CoreRecipes
Departments and Employees
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSError.h`

Constants

User info dictionary keys

These keys may exist in the user info dictionary.

```
extern NSString *NSLocalizedDescriptionKey;
extern NSString *NSErrorFailingURLStringKey;
const NSString *NSFilePathErrorKey;
const NSString *NSStringEncodingErrorKey;
const NSString *NSUnderlyingErrorKey;
const NSString *NSURLErrorKey;
const NSString *NSLocalizedFailureReasonErrorKey;
const NSString *NSLocalizedRecoverySuggestionErrorKey;
const NSString *NSLocalizedRecoveryOptionsErrorKey;
const NSString *NSRecoveryAttempterErrorKey;
```

Constants

`NSLocalizedDescriptionKey`

The corresponding value is a localized string representation of the error that, if present, will be returned by `localizedDescription` (page 565).

Available in Mac OS X v10.2 and later.

Declared in `NSError.h`.

`NSErrorFailingURLStringKey`

The corresponding value is the URL that caused the error. This key is only present in the `NSURLErrorDomain`.

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`.

`NSFilePathErrorKey`

Contains the file path of the error.

The corresponding value is an `NSString` object.

Available in Mac OS X v10.4 and later.

Declared in `NSError.h`.

`NSStringEncodingErrorKey`

The corresponding value is an `NSNumber` object containing the `NSStringEncoding` value.

Available in Mac OS X v10.4 and later.

Declared in `NSError.h`.

`NSUnderlyingErrorKey`

The corresponding value is an error that was encountered in an underlying implementation and caused the error that the receiver represents to occur.

Available in Mac OS X v10.3 and later.

Declared in `NSError.h`.

`NSURLErrorKey`
The corresponding value is an `NSURL` object.

Available in Mac OS X v10.4 and later.

Declared in `NSError.h`.

`NSLocalizedFailureReasonErrorKey`
The corresponding value is a localized string representation containing the reason for the failure that, if present, will be returned by `localizedFailureReason` (page 566).

This string provides a more detailed explanation of the error than the description.

Available in Mac OS X v10.4 and later.

Declared in `NSError.h`.

`NSLocalizedRecoverySuggestionErrorKey`
The corresponding value is a string containing the localized recovery suggestion for the error.

This string is suitable for displaying as the secondary message in an alert panel.

Available in Mac OS X v10.4 and later.

Declared in `NSError.h`.

`NSLocalizedRecoveryOptionsErrorKey`
The corresponding value is an array containing the localized titles of buttons appropriate for displaying in an alert panel.

The first string is the title of the right-most and default button, the second the one to the left, and so on. The recovery options should be appropriate for the recovery suggestion returned by `localizedRecoverySuggestion` (page 567).

Available in Mac OS X v10.4 and later.

Declared in `NSError.h`.

`NSRecoveryAttempterErrorKey`
The corresponding value is an object that conforms to the `NSErrorRecoveryAttempting` informal protocol.

The recovery attempter must be an object that can correctly interpret an index into the array returned by `recoveryAttempter` (page 567).

Available in Mac OS X v10.4 and later.

Declared in `NSError.h`.

Declared In
`NSError.h`

Error Domains

The following error domains are predefined.

```
const NSString *NSPOSIXErrorDomain;
const NSString *NSOSStatusErrorDomain;
const NSString *NSMachErrorDomain;
```

Constants
`NSPOSIXErrorDomain`
POSIX/BSD errors

Available in Mac OS X v10.2 and later.

Declared in `NSError.h`.

`NSOSStatusErrorDomain`

Mac OS 9/Carbon errors

Available in Mac OS X v10.2 and later.

Declared in `NSError.h`.

`NSMachErrorDomain`

Mach errors

Available in Mac OS X v10.2 and later.

Declared in `NSError.h`.

Discussion

Additionally, the following error domain is defined by Core Foundation:

`CFStreamErrorDomain`	Defines constants for values returned in the domain field of the CFStreamError structure.

Declared In

`NSError.h`

NSException Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSException.h
Companion guide	Exception Programming Topics for Cocoa
Related sample code	CoreRecipes EnhancedAudioBurn QTSSConnectionMonitor QTSSInspector Sketch-112

Overview

`NSException` is used to implement exception handling and contains information about an exception. An exception is a special condition that interrupts the normal flow of program execution. Each application can interrupt the program for different reasons. For example, one application might interpret saving a file in a directory that is write-protected as an exception. In this sense, the exception is equivalent to an error. Another application might interpret the user's keypress (for example, Control-C) as an exception: an indication that a long-running process should be aborted.

Note: The exception handling mechanism uses `longjmp` to control the flow of execution. Any code written for an application that uses exception handling is therefore subject to the restrictions associated with this functionality. See your compiler documentation for more information on the `longjmp` function.

Adopted Protocols

NSCoding

- `encodeWithCoder:` (page 2034)
- `initWithCoder:` (page 2034)

NSCopying
- copyWithZone: (page 2042)

Tasks

Creating and Raising an NSException Object

+ exceptionWithName:reason:userInfo: (page 574)
Creates and returns an exception object .

+ raise:format: (page 575)
A convenience method that creates and raises an exception.

+ raise:format:arguments: (page 576)
Creates and raises an exception with the specified name, reason, and arguments.

- initWithName:reason:userInfo: (page 577)
Initializes and returns a newly allocated exception object.

- raise (page 578)
Raises the receiver, causing program flow to jump to the local exception handler.

Querying an NSException Object

- name (page 577)
Returns an NSString object used to uniquely identify the receiver.

- reason (page 578)
Returns an NSString object containing a “human-readable” reason for the receiver.

- userInfo (page 579)
Returns an NSDictionary object containing application-specific data pertaining to the receiver.

Getting Exception Stack Frames

- callStackReturnAddresses (page 576)
Returns the call return addresses related to a raised exception.

Class Methods

exceptionWithName:reason:userInfo:

Creates and returns an exception object .

```
+ (NSException *)exceptionWithName:(NSString *)name reason:(NSString *)reason
    userInfo:(NSDictionary *)userInfo
```

Parameters

name

The name of the exception.

reason

A human-readable message string summarizing the reason for the exception.

userInfo

A dictionary containing user-defined information relating to the exception

Return Value

The created `NSException` object or `nil` if the object couldn't be created.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `initWithName:reason:userInfo:` (page 577)
- `name` (page 577)
- `reason` (page 578)
- `userInfo` (page 579)

Related Sample Code

Core Data HTML Store

CoreRecipes

Declared In

`NSException.h`

raise:format:

A convenience method that creates and raises an exception.

```
+ (void)raise:(NSString *)name format:(NSString *)format, ...
```

Parameters

name

The name of the exception.

format,

A human-readable message string (that is, the exception reason) with conversion specifications for the variable arguments that follow.

...

Variable information to be inserted into the formatted exception reason (in the manner of `printf`).

Discussion

The user-defined information is `nil` for the generated exception object.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `raise:format:arguments:` (page 576)
- `raise` (page 578)

Related Sample Code
EnhancedAudioBurn
QTSSConnectionMonitor
QTSSInspector
Sketch-112
TemperatureTester

Declared In
NSException.h

raise:format:arguments:

Creates and raises an exception with the specified name, reason, and arguments.

```
+ (void)raise:(NSString *)name format:(NSString *)format arguments:(va_list)argList
```

Parameters

name
The name of the exception.

format
A human-readable message string (that is, the exception reason) with conversion specifications for the variable arguments in `argList`.

argList
Variable information to be inserted into the formatted exception reason (in the manner of `vprintf`).

Discussion
The user-defined dictionary of the generated object is `nil`.

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

See Also
`+ raise:format:` (page 575)
`- raise` (page 578)

Declared In
NSException.h

Instance Methods

callStackReturnAddresses

Returns the call return addresses related to a raised exception.

```
- (NSArray *)callStackReturnAddresses
```

Return Value
An array of `NSNumber` objects encapsulating `NSUInteger` (page 2283) values. Each value is a call frame return address. The array of stack frames starts at the point at which the exception was first raised, with the first items being the most recent stack frames.

Discussion
`NSException` subclasses posing as the `NSException` class or subclasses or other API elements that interfere with the exception-raising mechanism may not get this information.

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

Declared In
`NSException.h`

initWithName:reason:userInfo:

Initializes and returns a newly allocated exception object.

```
- (id)initWithName:(NSString *)name reason:(NSString *)reason userInfo:(NSDictionary
    *)userInfo
```

Parameters

name
The name of the exception.

reason
A human-readable message string summarizing the reason for the exception.

userInfo
A dictionary containing user-defined information relating to the exception

Return Value
The created `NSException` object or `nil` if the object couldn't be created.

Discussion
This is the designated initializer.

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

See Also
`+ exceptionWithName:reason:userInfo:` (page 574)

Declared In
`NSException.h`

name

Returns an `NSString` object used to uniquely identify the receiver.

```
- (NSString *)name
```

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

See Also
+ exceptionWithName:reason:userInfo: (page 574)
- initWithName:reason:userInfo: (page 577)

Declared In
NSException.h

raise

Raises the receiver, causing program flow to jump to the local exception handler.

```
- (void)raise
```

Discussion
All other methods that raise an exception invoke this method, so set a breakpoint here if you are debugging exceptions. When there are no exception handlers in the exception handler stack, unless the exception is raised during the posting of a notification, this method calls the uncaught exception handler, in which last-minute logging can be performed. The program then terminates, regardless of the actions taken by the uncaught exception handler.

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

See Also
+ raise:format: (page 575)
+ raise:format:arguments: (page 576)

Related Sample Code
Core Data HTML Store

Declared In
NSException.h

reason

Returns an NSString object containing a "human-readable" reason for the receiver.

```
- (NSString *)reason
```

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

See Also
+ exceptionWithName:reason:userInfo: (page 574)
- initWithName:reason:userInfo: (page 577)

Declared In
NSException.h

userInfo

Returns an `NSDictionary` object containing application-specific data pertaining to the receiver.

```
- (NSDictionary *)userInfo
```

Discussion
Returns `nil` if no application-specific data exists. As an example, if a method's return value caused the exception to be raised, the return value might be available to the exception handler through this method.

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

See Also
`+ exceptionWithName:reason:userInfo:` (page 574)
`- initWithName:reason:userInfo:` (page 577)

Declared In
`NSException.h`

Constants

The string constants for exceptions are listed and described in the "Constants" (page 2287) chapter.

NSExistsCommand Class Reference

Inherits from	NSScriptCommand : NSObject
Conforms to	NSCoding (NSScriptCommand) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptStandardSuiteCommands.h
Companion guide	Cocoa Scripting Guide

Overview

An instance of `NSExistsCommand` determines whether a specified scriptable object, such as a word, paragraph, or image, exists.

When an instance of `NSExistsCommand` is executed, it evaluates the receiver specifier for the command to determine if it specifies any objects.

`NSExistsCommand` is part of Cocoa's built-in scripting support. Most applications don't need to subclass `NSExistsCommand`.

NSExpression Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSExpression.h
Availability	Available in Mac OS X v10.4 and later.
Companion guide	Predicate Programming Guide
Related sample code	iSpend

Overview

`NSExpression` is used to represent expressions in a predicate.

Comparison operations in an `NSPredicate` are based on two expressions, as represented by instances of the `NSExpression` class. Expressions are created for constant values, key paths, and so on.

Generally, anywhere in the `NSExpression` class hierarchy where there is composite API and subtypes that may only reasonably respond to a subset of that API, invoking a method that does not make sense for that subtype will cause an exception to be thrown.

Expression Types

In Mac OS X v10.5, `NSExpression` introduces several new expression types: `NSSubqueryExpressionType`, `NSAggregateExpressionType`, `NSUnionSetExpressionType`, `NSIntersectSetExpressionType`, and `NSMinusSetExpressionType`.

Aggregate Expressions

The aggregate expression allows you to create predicates containing expressions that evaluate to collections that contain further expressions. The collection may be an `NSArray`, `NSSet`, or `NSDictionary` object.

For example, consider the BETWEEN operator (`NSBetweenPredicateOperatorType`); its right hand side is a collection containing two elements. Using just the Mac OS X v10.4 API, these elements must be constants, as there is no way to populate them using variable expressions. On Mac OS X v10.4, it is not possible to create a predicate template to the effect of `date between {$YESTERDAY, $TOMORROW}`; instead you must create a new predicate each time.

Aggregate expressions are not supported by Core Data.

Subquery Expressions

The `NSSubqueryExpressionType` (page 601) creates a sub-expression, evaluation of which returns a subset of a collection of objects. It allows you to create sophisticated queries across relationships, such as a search for multiple correlated values on the destination object of a relationship.

Set Expressions

The set expressions (`NSUnionSetExpressionType` (page 601), `NSIntersectSetExpressionType` (page 601), and `NSMinusSetExpressionType` (page 601)) combine results in a manner similar to the `NSSet` methods.

Both sides of these expressions must evaluate to a collection; the left-hand side must evaluate to an `NSSet` object, the right-hand side can be any other collection type.

```
(expression UNION expression)
(expression INTERSECT expression)
(expression MINUS expression)
```

Set expressions are not supported by Core Data.

Function Expressions

On Mac OS X v10.4, `NSExpression` only supported a predefined set of functions: `sum`, `count`, `min`, `max`, and `average`. These predefined functions were accessed in the predicate syntax using custom keywords (for example, `MAX(1, 5, 10)`).

In Mac OS X v10.5, function expressions have been extended to support arbitrary method invocations as well. To use this extended functionality, you can now use the syntax `FUNCTION(receiver, selectorName, arguments, ...)`, for example:

```
FUNCTION(@"/Developer/Tools/otest", @"lastPathComponent") => @"otest"
```

All methods must take 0 or more `id` arguments and return an `id` value, although you can use the `CAST` expression to convert datatypes with lossy string representations (for example, `CAST(####, "NSDate")`). The `CAST` expression is extended in Mac OS X v10.5 to provide support for casting to classes for use in creating receivers for function expressions.

Note that although Core Data supports evaluation of the predefined functions, it does not support the evaluation of custom predicate functions in the persistent stores (during a fetch).

Tasks

Initializing an Expression

`- initWithExpressionType:` (page 597)
Initializes the receiver with the specified expression type.

Creating an Expression for a Value

`+ expressionForConstantValue:` (page 587)
Returns a new expression that represents a given constant value.

`+ expressionForEvaluatedObject` (page 587)
Returns a new expression that represents the object being evaluated.

`+ expressionForKeyPath:` (page 592)
Returns a new expression that invokes `valueForKeyPath:` with a given key path.

`+ expressionForVariable:` (page 595)
Returns a new expression that extracts a value from the variable bindings dictionary for a given key.

Creating a Collection Expression

`+ expressionForAggregate:` (page 586)
Returns a new aggregate expression for a given collection.

`+ expressionForUnionSet:with:` (page 594)
Returns a new `NSExpression` object that represent the union of a given set and collection.

`+ expressionForIntersectSet:with:` (page 592)
Returns a new `NSExpression` object that represent the intersection of a given set and collection.

`+ expressionForMinusSet:with:` (page 593)
Returns a new `NSExpression` object that represent the subtraction of a given collection from a given set.

Creating a Subquery

`+ expressionForSubquery:usingIteratorVariable:predicate:` (page 593)
Returns an expression that filters a collection by storing elements in the collection in a given variable and keeping the elements for which qualifier returns true.

Creating an Expression for a Function

`+ expressionForFunction:arguments:` (page 588)
Returns a new expression that will invoke one of the predefined functions.

+ `expressionForFunction:selectorName:arguments:` (page 591)
Returns an expression which will return the result of invoking on a given target a selector with a given name using given arguments.

Getting Information About an Expression

- `arguments` (page 595)
Returns the arguments for the receiver.
- `collection` (page 595)
Returns the collection of expressions in an aggregate expression, or the collection element of a subquery expression.
- `constantValue` (page 596)
Returns the constant value of the receiver.
- `expressionType` (page 596)
Returns the expression type for the receiver.
- `function` (page 597)
Returns the function for the receiver.
- `keyPath` (page 598)
Returns the key path for the receiver.
- `leftExpression` (page 598)
Returns the left expression of an aggregate expression.
- `operand` (page 598)
Returns the operand for the receiver.
- `predicate` (page 599)
Return the predicate of a subquery expression.
- `rightExpression` (page 599)
Returns the right expression of an aggregate expression.
- `variable` (page 599)
Returns the variable for the receiver.

Evaluating an Expression

- `expressionValueWithObject:context:` (page 596)
Evaluates an expression using a given object and context.

Class Methods

expressionForAggregate:

Returns a new aggregate expression for a given collection.

```
+ (NSExpression *)expressionForAggregate:(NSArray *)collection
```

Parameters

collection

A collection object (an instance of `NSArray`, `NSSet`, or `NSDictionary`) that contains further expressions.

Return Value
A new expression that contains the expressions in *collection*.

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

Declared In
`NSExpression.h`

expressionForConstantValue:

Returns a new expression that represents a given constant value.

```
+ (NSExpression *)expressionForConstantValue:(id)obj
```

Parameters

obj

The constant value the new expression is to represent.

Return Value
A new expression that represents the constant value, *obj*.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
iSpend

Declared In
`NSExpression.h`

expressionForEvaluatedObject

Returns a new expression that represents the object being evaluated.

```
+ (NSExpression *)expressionForEvaluatedObject
```

Return Value
A new expression that represents the object being evaluated.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSExpression.h`

expressionForFunction:arguments:

Returns a new expression that will invoke one of the predefined functions.

```
+ (NSExpression *)expressionForFunction:(NSString *)name arguments:(NSArray
    *)parameters
```

Parameters

name

The name of the function to invoke.

parameters

An array containing `NSExpression` objects that will be used as parameters during the invocation of selector.

For a selector taking no parameters, the array should be empty. For a selector taking one or more parameters, the array should contain one `NSExpression` object which will evaluate to an instance of the appropriate type for each parameter.

If there is a mismatch between the number of parameters expected and the number you provide during evaluation, an exception may be raised or missing parameters may simply be replaced by `nil` (which occurs depends on how many parameters are provided, and whether you have over- or underflow).

Return Value

A new expression that invokes the function *name* using the parameters in *parameters*.

Discussion

The *name* parameter can be one of the following predefined functions.

Function	Parameter	Returns	Availability
`average:`	An `NSArray` object containing `NSExpression` objects representing numbers	An `NSNumber` object (the average of values in the array)	Mac OS X v10.4 and later
`sum:`	An `NSArray` object containing `NSExpression` objects representing numbers	An `NSNumber` object (the sum of values in the array)	Mac OS X v10.4 and later
`count:`	An `NSArray` object containing `NSExpression` objects representing numbers	An `NSNumber` object (the number of elements in the array)	Mac OS X v10.4 and later
`min:`	An `NSArray` object containing `NSExpression` objects representing numbers	An `NSNumber` object (the minimum of the values in the array)	Mac OS X v10.4 and later
`max:`	An `NSArray` object containing `NSExpression` objects representing numbers	An `NSNumber` object (the maximum of the values in the array)	Mac OS X v10.4 and later
`median:`	An `NSArray` object containing `NSExpression` objects representing numbers	An `NSNumber` object (the median of the values in the array)	Mac OS X v10.5 and later

Function	Parameter	Returns	Availability
`mode:`	An `NSArray` object containing `NSExpression` objects representing numbers	An `NSArray` object (the mode of the values in the array)	Mac OS X v10.5 and later
`stddev:`	An `NSArray` object containing `NSExpression` objects representing numbers	An `NSNumber` object (the standard deviation of the values in the array)	Mac OS X v10.5 and later
`add:to:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the sum of the values in the array)	Mac OS X v10.5 and later
`from:subtract:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the result of subtracting the second value in the array from the first value in the array)	Mac OS X v10.5 and later
`multiply:by:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the result of multiplying the values in the array)	Mac OS X v10.5 and later
`divide:by:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the result of dividing the first value in the array by the second value in the array)	Mac OS X v10.5 and later
`modulus:by:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the remainder of dividing the first value in the array by the second value in the array)	Mac OS X v10.5 and later
`sqrt:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the square root of the value in the array)	Mac OS X v10.5 and later
`log:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the log of the value in the array)	Mac OS X v10.5 and later
`ln:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the natural log of the value in the array)	Mac OS X v10.5 and later
`raise:toPower:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the result of raising the first value in the array to the power of the second value in the array)	Mac OS X v10.5 and later
`exp:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the base-e exponential of the value in the aray)	Mac OS X v10.5 and later

Function	Parameter	Returns	Availability
`ceiling:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the smallest integral value not less than the value in the array)	Mac OS X v10.5 and later
`abs:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the absolute value of the value in the array)	Mac OS X v10.5 and later
`trunc:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the integral value nearest to but no greater than the value in the array)	Mac OS X v10.5 and later
`random`	`nil`	An `NSNumber` object (a random integer value)	Mac OS X v10.5 and later
`random:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (a random integer value between `0` and the value in the array (exclusive))	Mac OS X v10.5 and later
`now`	`nil`	An `[NSDate]` object (the current date and time)	Mac OS X v10.5 and later

This method raises an exception immediately if the selector is invalid; it raises an exception at runtime if the parameters are incorrect.

The *`parameters`* argument is a collection containing an expression which evaluates to a collection, as illustrated in the following examples:

```
NSNumber *number1 = [NSNumber numberWithInteger:20];
NSNumber *number2 = [NSNumber numberWithInteger:40];
NSArray *numberArray = [NSArray arrayWithObjects: number1, number2, nil];

NSExpression *arrayExpression = [NSExpression expressionForConstantValue:
numberArray];
NSArray *argumentArray = [NSArray arrayWithObject: arrayExpression];

NSExpression* expression =
    [NSExpression expressionForFunction:@"sum:" arguments:argumentArray];
id result = [expression expressionValueWithObject: nil context: nil];

BOOL ok = [result isEqual: [NSNumber numberWithInt: 60]]; // ok == YES

[NSExpression expressionForFunction:@"random" arguments:nil];

[NSExpression expressionForFunction:@"max:"
    arguments: [NSArray arrayWithObject:
        [NSExpression expressionForConstantValue:
            [NSArray arrayWithObjects:
                [NSNumber numberWithInt: 5], [NSNumber numberWithInt: 10],
nil]]]];
```

```
[NSExpression expressionForFunction:@"subtract:from:"
    arguments: [NSArray arrayWithObjects:
        [NSExpression expressionForConstantValue: [NSNumber numberWithInt: 5]],
       [NSExpression expressionForConstantValue: [NSNumber numberWithInt: 10]],
 nil]];
```

Special Considerations

This method throws an exception immediately if the selector is unknown; it throws at runtime if the parameters are incorrect.

Availability
Available in Mac OS X v10.4 and later.

See Also
`+ expressionForFunction:selectorName:arguments:` (page 591)

Declared In
`NSExpression.h`

expressionForFunction:selectorName:arguments:

Returns an expression which will return the result of invoking on a given target a selector with a given name using given arguments.

```
+ (NSExpression *)expressionForFunction:(NSExpression *)target selectorName:(NSString
     *)name arguments:(NSArray *)parameters
```

Parameters

target
: An `NSExpression` object which will evaluate an object on which the selector identified by *name* may be invoked.

name
: The name of the method to be invoked.

parameters
: An array containing `NSExpression` objects which can be evaluated to provide parameters for the method specified by *name*.

Return Value
An expression which will return the result of invoking the selector named *name* on the result of evaluating the target expression with the parameters specified by evaluating the elements of *parameters*.

Discussion
See the description of `expressionForFunction:arguments:` (page 588) for examples of how to construct the parameter array.

Special Considerations

This method throws an exception immediately if the selector is unknown; it throws at runtime if the parameters are incorrect.

This expression effectively allows your application to invoke any method on any object it can navigate to at runtime. You must consider the security implications of this type of evaluation.

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

See Also
+ expressionForFunction:arguments: (page 588)

Declared In
NSExpression.h

expressionForIntersectSet:with:

Returns a new NSExpression object that represent the intersection of a given set and collection.

```
+ (NSExpression *)expressionForIntersectSet:(NSExpression *)left with:(NSExpression
    *)right
```

Parameters

left
An expression that evaluates to an NSSet object.

right
An expression that evaluates to a collection object (an instance of NSArray, NSSet, or NSDictionary).

Return Value
A new NSExpression object that represents the intersection of *left* and *right*.

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

Declared In
NSExpression.h

expressionForKeyPath:

Returns a new expression that invokes valueForKeyPath: with a given key path.

```
+ (NSExpression *)expressionForKeyPath:(NSString *)keyPath
```

Parameters

keyPath
The key path that the new expression should evaluate.

Return Value
A new expression that invokes valueForKeyPath: (page 2071) with *keyPath*.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
iSpend

Declared In
NSExpression.h

expressionForMinusSet:with:

Returns a new `NSExpression` object that represent the subtraction of a given collection from a given set.

```
+ (NSExpression *)expressionForMinusSet:(NSExpression *)left with:(NSExpression
    *)right
```

Parameters

left

An expression that evaluates to an `NSSet` object.

right

An expression that evaluates to a collection object (an instance of `NSArray`, `NSSet`, or `NSDictionary`).

Return Value

A new `NSExpression` object that represents the subtraction of *right* from *left*.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSExpression.h`

expressionForSubquery:usingIteratorVariable:predicate:

Returns an expression that filters a collection by storing elements in the collection in a given variable and keeping the elements for which qualifier returns true.

```
+ (NSExpression *)expressionForSubquery:(NSExpression *)expression
    usingIteratorVariable:(NSString *)variable predicate:(id)predicate
```

Parameters

expression

A predicate expression that evaluates to a collection.

variable

Used as a local variable, and will shadow any instances of variable in the bindings dictionary. The variable is removed or the old value replaced once evaluation completes.

predicate

The predicate used to determine whether the element belongs in the result collection.

Return Value

An expression that filters a collection by storing elements in the collection in the variable variable and keeping the elements for which qualifier returns true

Discussion

This method creates a sub-expression, evaluation of which returns a subset of a collection of objects. It allows you to create sophisticated queries across relationships, such as a search for multiple correlated values on the destination object of a relationship.

For example, suppose you have an Apartment entity that has a to-many relationship to a Resident entity, and that you want to create a query for all apartments inhabited by a resident whose first name is "Jane" and whose last name is "Doe". Using only API available for Mac OS X v 10.4, you could try the predicate:

```
resident.firstname == "Jane" && resident.lastname == "Doe"
```

but this will always return false since `resident.firstname` and `resident.lastname` both return collections. You could also try:

```
resident.firstname CONTAINS "Jane" && resident.lastname CONTAINS "Doe"
```

but this is also flawed—it returns true if there are two residents, one of whom is John Doe and one of whom is Jane Smith. The only way to find the desired apartments is to do two passes: one through residents to find "Jane Doe", and one through apartness to find the ones where our Jane Does reside.

Subquery expressions provide a way to encapsulate this type of qualification into a single query.

The string format for a subquery expression is:

```
SUBQUERY(collection_expression, variable_expression, predicate);
```

where `expression` is a predicate expression that evaluates to a collection, `variableExpression` is an expression which will be used to contain each individual element of `collection`, and `predicate` is the predicate used to determine whether the element belongs in the result collection.

Using subqueries, the apartment query could be reformulated as

```
(SUBQUERY(residents, $x, $x.firstname == "Jane" && $x.lastname == "Doe").@count
 != 0)
```

or

```
(SUBQUERY(residents, $x, $x.firstname == "Jane" && $x.lastname == "Doe")[size]
 != 0)
```

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

Declared In
`NSExpression.h`

expressionForUnionSet:with:

Returns a new `NSExpression` object that represent the union of a given set and collection.

```
+ (NSExpression *)expressionForUnionSet:(NSExpression *)left with:(NSExpression
    *)right
```

Parameters

`left`
An expression that evaluates to an `NSSet` object.

`right`
An expression that evaluates to a collection object (an instance of `NSArray`, `NSSet`, or `NSDictionary`).

Return Value
An new `NSExpression` object that represents the union of *`left`* and *`right`*.

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

Declared In
NSExpression.h

expressionForVariable:

Returns a new expression that extracts a value from the variable bindings dictionary for a given key.

```
+ (NSExpression *)expressionForVariable:(NSString *)string
```

Parameters

string
The key for the variable to extract from the variable bindings dictionary.

Return Value
A new expression that extracts from the variable bindings dictionary the value for the key *string*.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSExpression.h

Instance Methods

arguments

Returns the arguments for the receiver.

```
- (NSArray *)arguments
```

Return Value
The arguments for the receiver—that is, the array of expressions that will be passed as parameters during invocation of the selector on the operand of a function expression.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSExpression.h

collection

Returns the collection of expressions in an aggregate expression, or the collection element of a subquery expression.

```
- (id)collection
```

Return Value
Returns the collection of expressions in an aggregate expression, or the collection element of a subquery expression.

Discussion
This method raises an exception if it is not applicable to the receiver.

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

Declared In
`NSExpression.h`

constantValue

Returns the constant value of the receiver.

```
- (id)constantValue
```

Return Value
The constant value of the receiver.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSExpression.h`

expressionType

Returns the expression type for the receiver.

```
- (NSExpressionType)expressionType
```

Return Value
The expression type for the receiver.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSExpression.h`

expressionValueWithObject:context:

Evaluates an expression using a given object and context.

```
- (id)expressionValueWithObject:(id)object context:(NSMutableDictionary *)context
```

Parameters

object

The object against which the receiver is evaluated.

context

A dictionary that the expression can use to store temporary state for one predicate evaluation.

Note that *context* is mutable, and that it can only be accessed during the evaluation of the expression. You must not attempt to retain it for use elsewhere.]

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSExpression.h

function

Returns the function for the receiver.

```
- (NSString *)function
```

Return Value
The function for the receiver.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSExpression.h

initWithExpressionType:

Initializes the receiver with the specified expression type.

```
- (id)initWithExpressionType:(NSExpressionType)type
```

Parameters

type

The type of the new expression, as defined by NSExpressionType (page 600).

Return Value
An initialized NSExpression object of the type *type*.

Special Considerations

This method is the designated initializer for NSExpression.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSExpression.h

keyPath

Returns the key path for the receiver.

```
- (NSString *)keyPath
```

Return Value
The key path for the receiver.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSExpression.h

leftExpression

Returns the left expression of an aggregate expression.

```
- (NSExpression *)leftExpression
```

Return Value
The left expression of a set expression.

Discussion
This method raises an exception if it is not applicable to the receiver.

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

Declared In
NSExpression.h

operand

Returns the operand for the receiver.

```
- (NSExpression *)operand
```

Return Value
The operand for the receiver—that is, the object on which the selector will be invoked.

Discussion
The object is the result of evaluating a key path or one of the defined functions. This method raises an exception if it is not applicable to the receiver.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSExpression.h`

predicate

Return the predicate of a subquery expression.

```
- (NSPredicate *)predicate
```

Return Value
The predicate of a subquery expression.

Discussion
This method raises an exception if it is not applicable to the receiver.

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

Declared In
`NSExpression.h`

rightExpression

Returns the right expression of an aggregate expression.

```
- (NSExpression *)rightExpression
```

Return Value
The right expression of a set expression.

Discussion
This method raises an exception if it is not applicable to the receiver.

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

Declared In
`NSExpression.h`

variable

Returns the variable for the receiver.

```
- (NSString *)variable
```

Return Value
The variable for the receiver.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSExpression.h`

Constants

NSExpressionType

Defines the possible types of `NSExpression`.

```
typedef enum {
    NSConstantValueExpressionType = 0,
    NSEvaluatedObjectExpressionType,
    NSVariableExpressionType,
    NSKeyPathExpressionType,
    NSFunctionExpressionType,
    NSAggregateExpressionType,
    NSSubqueryExpressionType,
    NSUnionSetExpressionType,
    NSIntersectSetExpressionType,
    NSMinusSetExpressionType
} NSExpressionType;
```

Constants

`NSConstantValueExpressionType`
An expression that always returns the same value.
Available in Mac OS X v10.4 and later.
Declared in `NSExpression.h`.

`NSEvaluatedObjectExpressionType`
An expression that always returns the parameter object itself.
Available in Mac OS X v10.4 and later.
Declared in `NSExpression.h`.

`NSVariableExpressionType`
An expression that always returns whatever value is associated with the key specified by ‘variable’ in the bindings dictionary.
Available in Mac OS X v10.4 and later.
Declared in `NSExpression.h`.

`NSKeyPathExpressionType`
An expression that returns something that can be used as a key path.
Available in Mac OS X v10.4 and later.
Declared in `NSExpression.h`.

`NSFunctionExpressionType`
An expression that returns the result of evaluating a function.
Available in Mac OS X v10.4 and later.
Declared in `NSExpression.h`.

`NSAggregateExpressionType`
: An expression that defines an aggregate of `NSExpression` objects.

  Available in Mac OS X v10.5 and later.

  Declared in `NSExpression.h`.

`NSSubqueryExpressionType`
: An expression that filters a collection using a subpredicate.

  Available in Mac OS X v10.5 and later.

  Declared in `NSExpression.h`.

`NSUnionSetExpressionType`
: An expression that creates a union of the results of two nested expressions.

  Available in Mac OS X v10.5 and later.

  Declared in `NSExpression.h`.

`NSIntersectSetExpressionType`
: An expression that creates an intersection of the results of two nested expressions.

  Available in Mac OS X v10.5 and later.

  Declared in `NSExpression.h`.

`NSMinusSetExpressionType`
: An expression that combines two nested expression results by set subtraction.

  Available in Mac OS X v10.5 and later.

  Declared in `NSExpression.h`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSExpression.h`

NSFileHandle Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSFileHandle.h
Companion guide	Low-Level File Management Programming Topics
Related sample code	AudioBurn PictureSharing PictureSharingBrowser

Overview

`NSFileHandle` objects provide an object-oriented wrapper for accessing open files or communications channels.

See the *PictureSharing* example project to examine code that creates an NSFileHandle object to listen for incoming connections; the file-handle object is initialized from a socket obtained through BSD calls.

Note: The deallocation of an `NSFileHandle` object deletes its descriptor and closes the represented file or channel unless the `NSFileHandle` object was created with `initWithFileDescriptor:` (page 612) or `initWithFileDescriptor:closeOnDealloc:` (page 612) with `NO` as the parameter argument.

Tasks

Getting a File Handle

`+ fileHandleForReadingAtPath:` (page 606)
Returns a file handle initialized for reading the file, device, or named socket at the specified path.

`+ fileHandleForWritingAtPath:` (page 607)
Returns a file handle initialized for writing to the file, device, or named socket at the specified path.

+ `fileHandleForUpdatingAtPath:` (page 606)
  Returns a file handle initialized for reading and writing to the file, device, or named socket at the specified path.
+ `fileHandleWithStandardError` (page 608)
  Returns the file handle associated with the standard error file.
+ `fileHandleWithStandardInput` (page 608)
  Returns the file handle associated with the standard input file.
+ `fileHandleWithStandardOutput` (page 609)
  Returns the file handle associated with the standard output file.
+ `fileHandleWithNullDevice` (page 607)
  Returns a file handle associated with a null device.

Creating a File Handle

- `initWithFileDescriptor:` (page 612)
  Returns a file handle initialized with a file descriptor.
- `initWithFileDescriptor:closeOnDealloc:` (page 612)
  Returns a file handle initialized with a file handle, using a specified deallocation policy.

Getting a File Descriptor

- `fileDescriptor` (page 611)
  Returns the file descriptor associated with the receiver.

Reading from a File Handle

- `availableData` (page 610)
  Returns the data available through the receiver.
- `readDataToEndOfFile` (page 614)
  Returns the data available through the receiver up to the end of file or maximum number of bytes.
- `readDataOfLength:` (page 613)
  Reads data up to a specified number of bytes from the receiver.

Writing to a File Handle

- `writeData:` (page 619)
  Synchronously writes data to the file, device, pipe, or socket represented by the receiver.

Communicating Asynchronously

- `acceptConnectionInBackgroundAndNotify` (page 609)
  Accepts a socket connection (for stream-type sockets only) in the background and creates a file handle for the "near" (client) end of the communications channel.

- `acceptConnectionInBackgroundAndNotifyForModes:` (page 610)
  Accepts a socket connection (for stream-type sockets only) in the background and creates a file handle for the "near" (client) end of the communications channel.
- `readInBackgroundAndNotify` (page 614)
  Reads from the file or communications channel in the background and posts a notification when finished.
- `readInBackgroundAndNotifyForModes:` (page 615)
  Reads from the file or communications channel in the background and posts a notification when finished.
- `readToEndOfFileInBackgroundAndNotify` (page 616)
  Reads to the end of file from the file or communications channel in the background and posts a notification when finished.
- `readToEndOfFileInBackgroundAndNotifyForModes:` (page 616)
  Reads to the end of file from the file or communications channel in the background and posts a notification when finished.
- `waitForDataInBackgroundAndNotify` (page 618)
  Checks to see if data is available in a background thread.
- `waitForDataInBackgroundAndNotifyForModes:` (page 619)
  Checks to see if data is available in a background thread.

Seeking Within a File

- `offsetInFile` (page 613)
  Returns the position of the file pointer within the file represented by the receiver.
- `seekToEndOfFile` (page 617)
  Puts the file pointer at the end of the file referenced by the receiver and returns the new file offset.
- `seekToFileOffset:` (page 617)
  Moves the file pointer to the specified offset within the file represented by the receiver.

Operating on a File

- `closeFile` (page 611)
  Disallows further access to the represented file or communications channel and signals end of file on communications channels that permit writing.
- `synchronizeFile` (page 618)
  Causes all in-memory data and attributes of the file represented by the receiver to be written to permanent storage.
- `truncateFileAtOffset:` (page 618)
  Truncates or extends the file represented by the receiver to a specified offset within the file and puts the file pointer at that position.

Class Methods

fileHandleForReadingAtPath:

Returns a file handle initialized for reading the file, device, or named socket at the specified path.

```
+ (id)fileHandleForReadingAtPath:(NSString *)path
```

Parameters

path

The path to the file, device, or named socket to access.

Return Value

The initialized file handle, or `nil` if no file exists at *path*.

Discussion

The file pointer is set to the beginning of the file. The returned object responds only to `NSFileHandle read...` messages.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `availableData` (page 610)
- `initWithFileDescriptor:` (page 612)
- `readDataOfLength:` (page 613)
- `readDataToEndOfFile` (page 614)

Related Sample Code

AudioBurn

Declared In

`NSFileHandle.h`

fileHandleForUpdatingAtPath:

Returns a file handle initialized for reading and writing to the file, device, or named socket at the specified path.

```
+ (id)fileHandleForUpdatingAtPath:(NSString *)path
```

Parameters

path

The path to the file, device, or named socket to access.

Return Value

The initialized file handle, or `nil` if no file exists at *path*.

Discussion

The file pointer is set to the beginning of the file. The returned object responds to both `NSFileHandle read...` messages and `writeData:` (page 619).

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

See Also
- `availableData` (page 610)
- `initWithFileDescriptor:` (page 612)
- `readDataOfLength:` (page 613)
- `readDataToEndOfFile` (page 614)

Declared In
`NSFileHandle.h`

fileHandleForWritingAtPath:

Returns a file handle initialized for writing to the file, device, or named socket at the specified path.

```
+ (id)fileHandleForWritingAtPath:(NSString *)path
```

Parameters
path
The path to the file, device, or named socket to access.

Return Value
The initialized file handle, or `nil` if no file exists at *path*.

Discussion
The file pointer is set to the beginning of the file. The returned object responds only to `writeData:` (page 619).

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

See Also
- `initWithFileDescriptor:` (page 612)

Declared In
`NSFileHandle.h`

fileHandleWithNullDevice

Returns a file handle associated with a null device.

```
+ (id)fileHandleWithNullDevice
```

Return Value
A file handle associated with a null device.

Discussion
You can use null-device file handles as "placeholders" for standard-device file handles or in collection objects to avoid exceptions and other errors resulting from messages being sent to invalid file handles. Read messages sent to a null-device file handle return an end-of-file indicator (an empty `NSData` object) rather than raise an exception. Write messages are no-ops, whereas `fileDescriptor` (page 611) returns an illegal value. Other methods are no-ops or return "sensible" values.

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

See Also
- `initWithFileDescriptor:` (page 612)

Declared In
`NSFileHandle.h`

fileHandleWithStandardError

Returns the file handle associated with the standard error file.

```
+ (id)fileHandleWithStandardError
```

Return Value
The shared file handle associated with the standard error file.

Discussion
Conventionally this is a terminal device to which error messages are sent. There is one standard error file handle per process; it is a shared instance.

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

See Also
- `+ fileHandleWithNullDevice` (page 607)
- `- initWithFileDescriptor:` (page 612)

Declared In
`NSFileHandle.h`

fileHandleWithStandardInput

Returns the file handle associated with the standard input file.

```
+ (id)fileHandleWithStandardInput
```

Return Value
The shared file handle associated with the standard input file.

Discussion
Conventionally this is a terminal device on which the user enters a stream of data. There is one standard input file handle per process; it is a shared instance.

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

See Also
- `+ fileHandleWithNullDevice` (page 607)
- `- initWithFileDescriptor:` (page 612)

Declared In
NSFileHandle.h

fileHandleWithStandardOutput

Returns the file handle associated with the standard output file.

```
+ (id)fileHandleWithStandardOutput
```

Return Value
The shared file handle associated with the standard output file.

Discussion
Conventionally this is a terminal device that receives a stream of data from a program. There is one standard output file handle per process; it is a shared instance.

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

See Also
+ fileHandleWithNullDevice (page 607)
- initWithFileDescriptor: (page 612)

Declared In
NSFileHandle.h

Instance Methods

acceptConnectionInBackgroundAndNotify

Accepts a socket connection (for stream-type sockets only) in the background and creates a file handle for the "near" (client) end of the communications channel.

```
- (void)acceptConnectionInBackgroundAndNotify
```

Discussion
This method is asynchronous. In a separate "safe" thread it accepts a connection, creates a file handle for the other end of the connection, and returns that object to the client by posting an NSFileHandleConnectionAcceptedNotification (page 621) in the run loop of the client. The notification includes as data a *userInfo* dictionary containing the created NSFileHandle object; access this object using the NSFileHandleNotificationFileHandleItem key.

The receiver must be created by an initWithFileDescriptor: (page 612) message that takes as an argument a stream-type socket created by the appropriate system routine. The object that will write data to the returned file handle must add itself as an observer of NSFileHandleConnectionAcceptedNotification (page 621).

Note that this method does not continue to listen for connection requests after it posts NSFileHandleConnectionAcceptedNotification. If you want to keep getting notified, you need to call acceptConnectionInBackgroundAndNotify again in your observer method.

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

See Also
- `enqueueNotification:postingStyle:coalesceMask:forModes:` (page 1047) (`NSNotificationQueue`)
- `readInBackgroundAndNotify` (page 614)
- `readToEndOfFileInBackgroundAndNotify` (page 616)

Related Sample Code
PictureSharing

Declared In
`NSFileHandle.h`

acceptConnectionInBackgroundAndNotifyForModes:

Accepts a socket connection (for stream-type sockets only) in the background and creates a file handle for the "near" (client) end of the communications channel.

```
- (void)acceptConnectionInBackgroundAndNotifyForModes:(NSArray *)modes
```

Parameters

modes

The runloop modes in which the connection accepted notification can be posted.

Discussion
See `acceptConnectionInBackgroundAndNotify` (page 609) for details of how this method operates. This method differs from `acceptConnectionInBackgroundAndNotify` (page 609) in that *modes* specifies the run-loop mode (or modes) in which `NSFileHandleConnectionAcceptedNotification` (page 621) can be posted.

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

See Also
- `enqueueNotification:postingStyle:coalesceMask:forModes:` (page 1047) (`NSNotificationQueue`)
- `readInBackgroundAndNotifyForModes:` (page 615)
- `readToEndOfFileInBackgroundAndNotifyForModes:` (page 616)

Declared In
`NSFileHandle.h`

availableData

Returns the data available through the receiver.

```
- (NSData *)availableData
```

Return Value
The data currently available through the receiver.

Discussion
If the receiver is a file, returns the data obtained by reading the file from the file pointer to the end of the file. If the receiver is a communications channel, reads up to a buffer of data and returns it; if no data is available, the method blocks. Returns an empty data object if the end of file is reached. Raises `NSFileHandleOperationException` if attempts to determine file-handle type fail or if attempts to read from the file or channel fail.

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

See Also
- `readDataOfLength:` (page 613)
- `readDataToEndOfFile` (page 614)

Declared In
`NSFileHandle.h`

closeFile

Disallows further access to the represented file or communications channel and signals end of file on communications channels that permit writing.

```
- (void)closeFile
```

Discussion
The file or communications channel is available for other uses after the file handle represented by the receiver is closed. Further read and write messages sent to a file handle to which `closeFile` has been sent raises an exception.

Sending `closeFile` to a file handle does not cause its deallocation. The deallocation of an `NSFileHandle` object deletes its descriptor and closes the represented file or channel unless the `NSFileHandle` object was created with `initWithFileDescriptor:` (page 612) or `initWithFileDescriptor:closeOnDealloc:` (page 612) with `NO` as the parameter argument.

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

Related Sample Code
PictureSharing

Declared In
`NSFileHandle.h`

fileDescriptor

Returns the file descriptor associated with the receiver.

```
- (int)fileDescriptor
```

Return Value
The POSIX file descriptor associated with the receiver.

Discussion
You can send this message to file handles originating from both file descriptors and file handles and receive a valid file descriptor so long as the file handle is open. If the file handle has been closed by sending it `closeFile` (page 611), this method raises an exception.

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

See Also
`- initWithFileDescriptor:` (page 612)

Declared In
`NSFileHandle.h`

initWithFileDescriptor:

Returns a file handle initialized with a file descriptor.

```
- (id)initWithFileDescriptor:(int)fileDescriptor
```

Parameters
fileDescriptor
The POSIX file descriptor with which to initialize the file handle.

Return Value
A file handle initialized with *fileDescriptor*.

Discussion
You can create a file handle for a socket by using the result of a `socket` call as *fileDescriptor*.

Special Considerations
The object creating a file handle using this method owns *fileDescriptor* and is responsible for its disposition.

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

See Also
`- closeFile` (page 611)

Declared In
`NSFileHandle.h`

initWithFileDescriptor:closeOnDealloc:

Returns a file handle initialized with a file handle, using a specified deallocation policy.

```
- (id)initWithFileDescriptor:(int)fileDescriptor closeOnDealloc:(BOOL)flag
```

Parameters
fileDescriptor
The POSIX file descriptor with which to initialize the file handle.

flag

YES if the file descriptor should be closed when the receiver is deallocated, otherwise NO.

Return Value
A file handle initialized with *fileDescriptor* with a deallocation policy specified by *flag*.

Special Considerations

If *flag* is NO, the object creating a file handle using this method owns *fileDescriptor* and is responsible for its disposition.

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

See Also
- closeFile (page 611)

Declared In
NSFileHandle.h

offsetInFile

Returns the position of the file pointer within the file represented by the receiver.

```
- (unsigned long long)offsetInFile
```

Return Value
The position of the file pointer within the file represented by the receiver.

Special Considerations

Raises an exception if the message is sent to a file handle representing a pipe or socket or if the file descriptor is closed.

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

See Also
- seekToEndOfFile (page 617)
- seekToFileOffset: (page 617)

Related Sample Code
AudioBurn

Declared In
NSFileHandle.h

readDataOfLength:

Reads data up to a specified number of bytes from the receiver.

```
- (NSData *)readDataOfLength:(NSUInteger)length
```

Parameters

length

The number of bytes to read from the receiver.

Return Value

The data available through the receiver up to a maximum of *length* bytes.

Discussion

If the receiver is a file, returns the data obtained by reading from the file pointer to *length* or to the end of the file, whichever comes first. If the receiver is a communications channel, the method reads data from the channel up to *length*. Returns an empty `NSData` object if the file is positioned at the end of the file or if an end-of-file indicator is returned on a communications channel. Raises `NSFileHandleOperationException` if attempts to determine file-handle type fail or if attempts to read from the file or channel fail.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `availableData` (page 610)
- `readDataToEndOfFile` (page 614)

Declared In

`NSFileHandle.h`

readDataToEndOfFile

Returns the data available through the receiver up to the end of file or maximum number of bytes.

```
- (NSData *)readDataToEndOfFile
```

Return Value

The data available through the receiver up to `UINT_MAX` bytes (the maximum value for unsigned integers) or, if a communications channel, until an end-of-file indicator is returned.

Discussion

This method invokes `readDataOfLength:` (page 613) as part of its implementation.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `availableData` (page 610)

Declared In

`NSFileHandle.h`

readInBackgroundAndNotify

Reads from the file or communications channel in the background and posts a notification when finished.

```
- (void)readInBackgroundAndNotify
```

Discussion
This method performs an asynchronous `availableData` (page 610) operation on a file or communications channel and posts an `NSFileHandleReadCompletionNotification` (page 622) to the client process's run loop.

The length of the data is limited to the buffer size of the underlying operating system. The notification includes a *`userInfo`* dictionary that contains the data read; access this object using the `NSFileHandleNotificationDataItem` key.

Any object interested in receiving this data asynchronously must add itself as an observer of `NSFileHandleReadCompletionNotification` (page 622). In communication via stream-type sockets, the receiver is often the object returned in the *`userInfo`* dictionary of `NSFileHandleConnectionAcceptedNotification` (page 621).

Note that this method does not cause a continuous stream of notifications to be sent. If you wish to keep getting notified, you'll also need to call `readInBackgroundAndNotify` in your observer method.

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

See Also
- `acceptConnectionInBackgroundAndNotify` (page 609)
- `readToEndOfFileInBackgroundAndNotifyForModes:` (page 616)
- `enqueueNotification:postingStyle:coalesceMask:forModes:` (page 1047) (`NSNotificationQueue`)

Related Sample Code
Moriarity

Declared In
`NSFileHandle.h`

readInBackgroundAndNotifyForModes:

Reads from the file or communications channel in the background and posts a notification when finished.

```
- (void)readInBackgroundAndNotifyForModes:(NSArray *)modes
```

Parameters
`modes`
The runloop modes in which the read completion notification can be posted.

Discussion
See `readInBackgroundAndNotify` (page 614) for details of how this method operates. This method differs from `readInBackgroundAndNotify` (page 614) in that *`modes`* specifies the run-loop mode (or modes) in which `NSFileHandleReadCompletionNotification` (page 622) can be posted.

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

See Also
- `acceptConnectionInBackgroundAndNotifyForModes:` (page 610)
- `enqueueNotification:postingStyle:coalesceMask:forModes:` (page 1047) (`NSNotificationQueue`)

Declared In
NSFileHandle.h

readToEndOfFileInBackgroundAndNotify

Reads to the end of file from the file or communications channel in the background and posts a notification when finished.

```
- (void)readToEndOfFileInBackgroundAndNotify
```

Discussion
This method performs an asynchronous `readToEndOfFile` operation on a file or communications channel and posts an `NSFileHandleReadToEndOfFileCompletionNotification` (page 623) to the client process's run loop.

The notification includes a *`userInfo`* dictionary that contains the data read; access this object using the `NSFileHandleNotificationDataItem` key.

Any object interested in receiving this data asynchronously must add itself as an observer of `NSFileHandleReadToEndOfFileCompletionNotification` (page 623). In communication via stream-type sockets, the receiver is often the object returned in the *`userInfo`* dictionary of `NSFileHandleConnectionAcceptedNotification` (page 621).

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

See Also
- `acceptConnectionInBackgroundAndNotify` (page 609)
- `readToEndOfFileInBackgroundAndNotifyForModes:` (page 616)
- `enqueueNotification:postingStyle:coalesceMask:forModes:` (page 1047) (`NSNotificationQueue`)

Related Sample Code
PictureSharingBrowser

Declared In
NSFileHandle.h

readToEndOfFileInBackgroundAndNotifyForModes:

Reads to the end of file from the file or communications channel in the background and posts a notification when finished.

```
- (void)readToEndOfFileInBackgroundAndNotifyForModes:(NSArray *)modes
```

Parameters
`modes`
The runloop modes in which the read completion notification can be posted.

Discussion
See `readToEndOfFileInBackgroundAndNotify` (page 616) for details of this method's operation. The method differs from `readToEndOfFileInBackgroundAndNotify` (page 616) in that *`modes`* specifies the run-loop mode (or modes) in which `NSFileHandleReadToEndOfFileCompletionNotification` (page 623) can be posted.

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

See Also
- `acceptConnectionInBackgroundAndNotifyForModes:` (page 610)
- `enqueueNotification:postingStyle:coalesceMask:forModes:` (page 1047) (`NSNotificationQueue`)

Declared In
`NSFileHandle.h`

seekToEndOfFile

Puts the file pointer at the end of the file referenced by the receiver and returns the new file offset.

```
- (unsigned long long)seekToEndOfFile
```

Return Value
The file offset with the file pointer at the end of the file. This is therefore equal to the size of the file.

Special Considerations

Raises an exception if the message is sent to an `NSFileHandle` object representing a pipe or socket or if the file descriptor is closed.

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

See Also
- `offsetInFile` (page 613)

Declared In
`NSFileHandle.h`

seekToFileOffset:

Moves the file pointer to the specified offset within the file represented by the receiver.

```
- (void)seekToFileOffset:(unsigned long long)offset
```

Parameters
offset
The offset to seek to.

Special Considerations

Raises an exception if the message is sent to an `NSFileHandle` object representing a pipe or socket, if the file descriptor is closed, or if any other error occurs in seeking.

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

See Also
- `offsetInFile` (page 613)

Related Sample Code
AudioBurn

Declared In
`NSFileHandle.h`

synchronizeFile

Causes all in-memory data and attributes of the file represented by the receiver to be written to permanent storage.

```
- (void)synchronizeFile
```

Discussion
This method should be invoked by programs that require the file to always be in a known state. An invocation of this method does not return until memory is flushed.

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

Declared In
`NSFileHandle.h`

truncateFileAtOffset:

Truncates or extends the file represented by the receiver to a specified offset within the file and puts the file pointer at that position.

```
- (void)truncateFileAtOffset:(unsigned long long)offset
```

Parameters

offset
The offset within the file that will mark the new end of the file.

Discussion
If the file is extended (if *offset* is beyond the current end of file), the added characters are null bytes.

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

Declared In
`NSFileHandle.h`

waitForDataInBackgroundAndNotify

Checks to see if data is available in a background thread.

```
- (void)waitForDataInBackgroundAndNotify
```

Discussion
When the data becomes available, the thread notifies all observers with `NSFileHandleDataAvailableNotification` (page 622). After the notification has been posted, the thread is terminated.

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

See Also
`- waitForDataInBackgroundAndNotifyForModes:` (page 619)

Declared In
`NSFileHandle.h`

waitForDataInBackgroundAndNotifyForModes:

Checks to see if data is available in a background thread.

```
- (void)waitForDataInBackgroundAndNotifyForModes:(NSArray *)modes
```

Parameters

modes

The runloop modes in which the data available notification can be posted.

Discussion
When the data becomes available, the thread notifies all observers with `NSFileHandleDataAvailableNotification` (page 622). After the notification has been posted, the thread is terminated. This method differs from `waitForDataInBackgroundAndNotify` (page 618) in that *modes* specifies the run-loop mode (or modes) in which `NSFileHandleDataAvailableNotification` (page 622) can be posted.

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

See Also
`- waitForDataInBackgroundAndNotify` (page 618)

Declared In
`NSFileHandle.h`

writeData:

Synchronously writes data to the file, device, pipe, or socket represented by the receiver.

```
- (void)writeData:(NSData *)data
```

Parameters

data

The data to be written.

Discussion
If the receiver is a file, writing takes place at the file pointer's current position. After it writes the data, the method advances the file pointer by the number of bytes written. Raises an exception if the file descriptor is closed or is not valid, if the receiver represents an unconnected pipe or socket endpoint, if no free space is left on the file system, or if any other writing error occurs.

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

See Also
- `- availableData` (page 610)
- `- readDataOfLength:` (page 613)
- `- readDataToEndOfFile` (page 614)

Related Sample Code
PictureSharing

Declared In
`NSFileHandle.h`

Constants

Keys for Notification UserInfo Dictionary

Strings that are used as keys in a userinfo dictionary in a file handle notification.

```
NSString * const NSFileHandleNotificationFileHandleItem;
NSString * const NSFileHandleNotificationDataItem;
```

Constants

`NSFileHandleNotificationFileHandleItem`

A key in the userinfo dictionary in a NSFileHandleConnectionAcceptedNotification (page 621) notification.

The corresponding value is the `NSFileHandle` object representing the "near" end of a socket connection.

Available in Mac OS X v10.0 and later.

Declared in `NSFileHandle.h`.

`NSFileHandleNotificationDataItem`

A key in the userinfo dictionary in a NSFileHandleReadCompletionNotification (page 622) and NSFileHandleReadToEndOfFileCompletionNotification (page 623).

The corresponding value is an `NSData` object containing the available data read from a socket connection.

Available in Mac OS X v10.0 and later.

Declared in `NSFileHandle.h`.

Declared In
`NSFileHandle.h`

Exception Names

Constant that defines the name of a file operation exception.

```
extern NSString *NSFileHandleOperationException;
```

Constants

`NSFileHandleOperationException`

Raised by `NSFileHandle` if attempts to determine file-handle type fail or if attempts to read from a file or channel fail.

Available in Mac OS X v10.0 and later.

Declared in `NSFileHandle.h`.

Declared In

`NSFileHandle.h`

Unused Constant

Constant that is currently unused.

```
NSString * const NSFileHandleNotificationMonitorModes;
```

Constants

`NSFileHandleNotificationMonitorModes`

Currently unused.

Available in Mac OS X v10.0 and later.

Declared in `NSFileHandle.h`.

Declared In

`NSFileHandle.h`

Notifications

`NSFileHandle` posts several notifications related to asynchronous background I/O operations. They are set to post when the run loop of the thread that started the asynchronous operation is idle.

NSFileHandleConnectionAcceptedNotification

This notification is posted when an `NSFileHandle` object establishes a socket connection between two processes, creates an `NSFileHandle` object for one end of the connection, and makes this object available to observers by putting it in the *`userInfo`* dictionary. To cause the posting of this notification, you must send either `acceptConnectionInBackgroundAndNotify` (page 609) or `acceptConnectionInBackgroundAndNotifyForModes:` (page 610) to an `NSFileHandle` object representing a server stream-type socket.

The notification object is the `NSFileHandle` object that sent the notification. The *`userInfo`* dictionary contains the following information:

Key	Value
`NSFileHandleNotificationFileHandleItem`	The `NSFileHandle` object representing the "near" end of a socket connection
`@"NSFileHandleError"`	An `NSNumber` object containing an integer representing the UNIX-type error which occurred

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

Declared In
`NSFileHandle.h`

NSFileHandleDataAvailableNotification

This notification is posted when the background thread determines that data is currently available for reading in a file or at a communications channel. The observers can then issue the appropriate messages to begin reading the data. To cause the posting of this notification, you must send either `waitForDataInBackgroundAndNotify` (page 618) or `waitForDataInBackgroundAndNotifyForModes:` (page 619) to an appropriate `NSFileHandle` object.

The notification object is the `NSFileHandle` object that sent the notification. This notification does not contain a *`userInfo`* dictionary.

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

Declared In
`NSFileHandle.h`

NSFileHandleReadCompletionNotification

This notification is posted when the background thread reads the data currently available in a file or at a communications channel. It makes the data available to observers by putting it in the *`userInfo`* dictionary. To cause the posting of this notification, you must send either `readInBackgroundAndNotify` (page 614) or `readInBackgroundAndNotifyForModes:` (page 615) to an appropriate `NSFileHandle` object.

The notification object is the `NSFileHandle` object that sent the notification. The *`userInfo`* dictionary contains the following information:

Key	Value
`NSFileHandleNotificationDataItem`	An `NSData` object containing the available data read from a socket connection
`@"NSFileHandleError"`	An `NSNumber` object containing an integer representing the UNIX-type error which occurred

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

Declared In
NSFileHandle.h

NSFileHandleReadToEndOfFileCompletionNotification

This notification is posted when the background thread reads all data in the file or, if a communications channel, until the other process signals the end of data. It makes the data available to observers by putting it in the *userInfo* dictionary. To cause the posting of this notification, you must send either readToEndOfFileInBackgroundAndNotify (page 616) or readToEndOfFileInBackgroundAndNotifyForModes: (page 616) to an appropriate NSFileHandle object.

The notification object is the NSFileHandle object that sent the notification. The *userInfo* dictionary contains the following information:

Key	Value
NSFileHandleNotificationDataItem	An NSData object containing the available data read from a socket connection
@"NSFileHandleError"	An NSNumber object containing an integer representing the UNIX-type error which occurred

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

Declared In
NSFileHandle.h

NSFileManager Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSFileManager.h
Companion guide	Low-Level File Management Programming Topics
Related sample code	Core Data HTML Store CoreRecipes MyPhoto Quartz Composer WWDC 2005 TextEdit TextEditPlus

Overview

`NSFileManager` enables you to perform many generic file-system operations and insulates an application from the underlying file system.

Tasks

Getting the Default Manager

`+ defaultManager` (page 630)
Returns the default `NSFileManager` object for the file system.

Moving an Item

`- movePath:toPath:handler:` (page 654)
Moves the directory or file specified by a given path to a different location in the file system identified by another path.

- `fileManager:shouldMoveItemAtPath:toPath:` (page 662) *delegate method*
  An `NSFileManager` object sends this message immediately before attempting to move to a given path.
- `moveItemAtPath:toPath:error:` (page 654)
  Moves the directory or file specified by a given path to a different location in the file system identified by another path.
- `fileManager:shouldProceedAfterError:movingItemAtPath:toPath:` (page 665) *delegate method*
  An `NSFileManager` object sends this message if an error occurs during an attempt to move to a given path.

Copying an Item

- `copyPath:toPath:handler:` (page 636)
  Copies the directory or file specified in a given path to a different location in the file system identified by another path.
- `fileManager:shouldCopyItemAtPath:toPath:` (page 661) *delegate method*
  An `NSFileManager` object sends this message immediately before attempting to copy to a given path.
- `copyItemAtPath:toPath:error:` (page 635)
  Copies the directory or file specified in a given path to a different location in the file system identified by another path.
- `fileManager:shouldProceedAfterError:copyingItemAtPath:toPath:` (page 664) *delegate method*
  An `NSFileManager` object sends this message if an error occurs during an attempt to copy to a given path.

Removing an Item

- `removeFileAtPath:handler:` (page 656)
  Deletes the file, link, or directory (including, recursively, all subdirectories, files, and links in the directory) identified by a given path.
- `fileManager:shouldRemoveItemAtPath:` (page 666) *delegate method*
  An `NSFileManager` object sends this message immediately before attempting to delete an item at a given path.
- `removeItemAtPath:error:` (page 657)
  Deletes the file, link, or directory (including, recursively, all subdirectories, files, and links in the directory) identified by a given path.
- `fileManager:shouldProceedAfterError:removingItemAtPath:` (page 666) *delegate method*
  An `NSFileManager` object sends this message if an error occurs during an attempt to delete a given path.

Creating an Item

- `createDirectoryAtPath:attributes:` (page 637)
  Creates a directory (without contents) at a given path with given attributes.

- `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 638)
  Creates a directory with given attributes at a specified path.
- `createFileAtPath:contents:attributes:` (page 639)
  Creates a file at a given path that has given attributes and contents.

Linking an Item

- `linkPath:toPath:handler:` (page 652)
  Creates a link from a source to a destination.
- `fileManager:shouldLinkItemAtPath:toPath:` (page 662) *delegate method*
  An `NSFileManager` object sends this message immediately before attempting to link to a given path.
- `linkItemAtPath:toPath:error:` (page 651)
  Creates a link from a source to a destination.
- `fileManager:shouldProceedAfterError:linkingItemAtPath:toPath:` (page 665) *delegate method*
  An `NSFileManager` object sends this message if an error occurs during an attempt to link to a given path.

Symbolic-Link Operations

- `createSymbolicLinkAtPath:pathContent:` (page 640)
  Creates a symbolic link identified by a given path that refers to a given location.
- `createSymbolicLinkAtPath:withDestinationPath:error:` (page 640)
  Creates a symbolic link identified by a given path that refers to a given location.
- `pathContentOfSymbolicLinkAtPath:` (page 656)
  Returns the path of the directory or file that a symbolic link at a given path refers to.
- `destinationOfSymbolicLinkAtPath:error:` (page 642)
  Returns an `NSString` object containing the path of the item pointed at by the symlink specified by a given path.

Handling File Operations

The methods described in this section are methods to be implemented by the callback handler passed to several methods of `NSFileManager`.

- `fileManager:shouldProceedAfterError:` (page 663) *delegate method*
  An `NSFileManager` object sends this message to its handler for each error it encounters when copying, moving, removing, or linking files or directories.
- `fileManager:willProcessPath:` (page 667) *delegate method*
  An `NSFileManager` object sends this message to a handler immediately before attempting to move, copy, rename, or delete, or before attempting to link to a given path.

Getting and Comparing File Contents

- `contentsAtPath:` (page 633)
  Returns as an `NSData` object the contents of the file at at given path.
- `contentsEqualAtPath:andPath:` (page 634)
  Returns a Boolean value that indicates whether the files or directories in specified paths have the same contents.

Discovering Directory Contents

- `directoryContentsAtPath:` (page 642)
  Returns an array of `NSString` objects identifying the directories and files (including symbolic links) contained in a given directory.
- `contentsOfDirectoryAtPath:error:` (page 634)
  Returns an array of `NSString` objects identifying the directories and files (including symbolic links) contained in a given directory.
- `enumeratorAtPath:` (page 644)
  Creates and returns an `NSDirectoryEnumerator` object that enumerates the contents of the directory at a given path.
- `subpathsAtPath:` (page 659)
  Returns an array that contains (as `NSString` objects) the contents of the directory identified by a given path.
- `subpathsOfDirectoryAtPath:error:` (page 660)
  Returns an array that contains the filenames of the items in the directory specified by a given path and all its subdirectories recursively.

Determining Access to Files

- `fileExistsAtPath:` (page 646)
  Returns a Boolean value that indicates whether a file or directory exists at a specified path.
- `fileExistsAtPath:isDirectory:` (page 647)
  Returns a Boolean value that indicates whether a file or directory exists at a specified path.
- `isReadableFileAtPath:` (page 650)
  Returns a Boolean value that indicates whether the invoking object appears able to read a specified file.
- `isWritableFileAtPath:` (page 651)
  Returns a Boolean value that indicates whether the invoking object appears able to write to a specified file.
- `isExecutableFileAtPath:` (page 650)
  Returns a Boolean value that indicates whether the operating system appears able to execute a specified file.
- `isDeletableFileAtPath:` (page 649)
  Returns a Boolean value that indicates whether the invoking object appears able to delete a specified file.

Getting and Setting Attributes

- `componentsToDisplayForPath:` (page 633)
  Returns an array of `NSString` objects representing the user-visible components of a given path.
- `displayNameAtPath:` (page 643)
  Returns the name of the file or directory at a given path in a localized form appropriate for presentation to the user.
- `fileAttributesAtPath:traverseLink:` (page 645)
  Returns a dictionary that describes the POSIX attributes of the file specified at a given.
- `attributesOfItemAtPath:error:` (page 631)
  An `NSDictionary` object containing the attributes of the item at a given path.
- `fileSystemAttributesAtPath:` (page 648)
  Returns a dictionary that describes the attributes of the mounted file system on which a given path resides.
- `attributesOfFileSystemForPath:error:` (page 630)
  Returns a dictionary that describes the attributes of the mounted file system on which a given path resides.
- `changeFileAttributes:atPath:` (page 632)
  Changes the attributes of a given file or directory.
- `setAttributes:ofItemAtPath:error:` (page 658)
  Sets the attributes of a given file or directory.

Getting Representations of File Paths

- `fileSystemRepresentationWithPath:` (page 649)
  Returns a C-string representation of a given path that properly encodes Unicode strings for use by the file system.
- `stringWithFileSystemRepresentation:length:` (page 659)
  Returns an `NSString` object converted from the C-string representation of a pathname in the current file system.

Managing the Delegate

- `setDelegate:` (page 659)
  Sets the delegate for the receiver.
- `delegate` (page 641)
  Returns the delegate for the receiver.

Managing the Current Directory

- `changeCurrentDirectoryPath:` (page 631)
  Changes the path of the current directory for the current process to a given path.
- `currentDirectoryPath` (page 641)
  Returns the path of the program's current directory.

Class Methods

defaultManager

Returns the default `NSFileManager` object for the file system.

```
+ (NSFileManager *)defaultManager
```

Return Value
The default `NSFileManager` object for the file system.

Discussion
You invoke all `NSFileManager` instance methods with this object as the receiver.

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

Related Sample Code
Core Data HTML Store
CoreRecipes
ImageBrowser
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSFileManager.h`

Instance Methods

attributesOfFileSystemForPath:error:

Returns a dictionary that describes the attributes of the mounted file system on which a given path resides.

```
- (NSDictionary *)attributesOfFileSystemForPath:(NSString *)path error:(NSError
    **)error
```

Parameters

path
Any pathname within the mounted file system.

error
If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value
An `NSDictionary` object that describes the attributes of the mounted file system on which *path* resides. See "File-System Attribute Keys" (page 672) for a description of the keys available in the dictionary.

Discussion
This method does not traverse an initial symbolic link.

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

See Also
- `- fileSystemAttributesAtPath:` (page 648)
- `- fileAttributesAtPath:traverseLink:` (page 645)
- `- changeFileAttributes:atPath:` (page 632)

Declared In
`NSFileManager.h`

attributesOfItemAtPath:error:

An `NSDictionary` object containing the attributes of the item at a given path.

```
- (NSDictionary *)attributesOfItemAtPath:(NSString *)path error:(NSError **)error
```

Parameters

path

The path of a file or directory.

error

If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value
An `NSDictionary` object that describes the attributes (file, directory, symlink, and so on) of the file specified by *path*. The keys in the dictionary are described in "`File Attribute Keys`" (page 668).

Discussion
This method does not traverse an initial symbolic link.

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

See Also
- `- fileAttributesAtPath:traverseLink:` (page 645)
- `- changeFileAttributes:atPath:` (page 632)

Declared In
`NSFileManager.h`

changeCurrentDirectoryPath:

Changes the path of the current directory for the current process to a given path.

```
- (BOOL)changeCurrentDirectoryPath:(NSString *)path
```

Parameters

path

The path of the directory to which to change.

Return Value
`YES` if successful, otherwise `NO`.

Discussion
All relative pathnames refer implicitly to the current working directory. The current working directory is stored per process.

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

See Also
- `currentDirectoryPath` (page 641)
- `fileExistsAtPath:isDirectory:` (page 647)
- `directoryContentsAtPath:` (page 642)
- `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 638)
- `createDirectoryAtPath:attributes:` (page 637)

Declared In
`NSFileManager.h`

changeFileAttributes:atPath:

Changes the attributes of a given file or directory.

```
- (BOOL)changeFileAttributes:(NSDictionary *)attributes  atPath:(NSString *)path
```

Parameters

attributes

A dictionary containing as keys the attributes to set for *path* and as values the corresponding value for the attribute. You can set following: `NSFileBusy`, `NSFileCreationDate`, `NSFileExtensionHidden`, `NSFileGroupOwnerAccountID`, `NSFileGroupOwnerAccountName`, `NSFileHFSCreatorCode`, `NSFileHFSTypeCode`, `NSFileImmutable`, `NSFileModificationDate`, `NSFileOwnerAccountID`, `NSFileOwnerAccountName`, `NSFilePosixPermissions`. You can change single attributes or any combination of attributes; you need not specify keys for all attributes.

For the `NSFilePosixPermissions` value, specify a file mode from the OR'd permission bit masks defined in `sys/stat.h`. See the man page for the `chmod` function (`man 2 chmod`) for an explanation.

path

A path to a file or directory.

Return Value
`YES` if *all* changes succeed. If any change fails, returns `NO`, but it is undefined whether any changes actually occurred.

Discussion
As in the POSIX standard, the application either must own the file or directory or must be running as superuser for attribute changes to take effect. The method attempts to make all changes specified in attributes and ignores any rejection of an attempted modification.

The `NSFilePosixPermissions` value must be initialized with the code representing the POSIX file-permissions bit pattern. `NSFileHFSCreatorCode` and `NSFileHFSTypeCode` will only be heeded when *path* specifies a file.

Special Considerations

On Mac OS X v10.5 and later, use `setAttributes:ofItemAtPath:error:` (page 658) instead.

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

See Also
- `fileAttributesAtPath:traverseLink:` (page 645)
- `setAttributes:ofItemAtPath:error:` (page 658)

Related Sample Code
File Wrappers with Core Data Documents
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSFileManager.h`

componentsToDisplayForPath:

Returns an array of `NSString` objects representing the user-visible components of a given path.

```
- (NSArray *)componentsToDisplayForPath:(NSString *)path
```

Parameters
path
A pathname.

Return Value
An array of `NSString` objects representing the user-visible (for the Finder, Open and Save panels, and so on) components of *path*.

Discussion
These components cannot be used for path operations and are only suitable for display to the user.

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

Related Sample Code
QTAudioExtractionPanel

Declared In
`NSFileManager.h`

contentsAtPath:

Returns as an `NSData` object the contents of the file at at given path.

```
- (NSData *)contentsAtPath:(NSString *)path
```

Parameters
path
The path of a file.

Return Value
The contents of the file specified by *path* as an `NSData` object. If *path* specifies a directory, or if some other error occurs, returns `nil`.

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

See Also
- `contentsEqualAtPath:andPath:` (page 634)
- `createFileAtPath:contents:attributes:` (page 639)

Declared In
`NSFileManager.h`

contentsEqualAtPath:andPath:

Returns a Boolean value that indicates whether the files or directories in specified paths have the same contents.

```
- (BOOL)contentsEqualAtPath:(NSString *)path1  andPath:(NSString *)path2
```

Parameters

path1
The path of a file or directory to compare with the contents of *path2*.

path2
The path of a file or directory to compare with the contents of *path1*.

Return Value
`YES` if file or directory specified in *path1* has the same contents as that specified in *path2*, otherwise `NO`.

Discussion
If *path1* and *path2* are directories, the contents are the list of files and subdirectories each contains—contents of subdirectories are also compared. For files, this method checks to see if they're the same file, then compares their size, and finally compares their contents. This method does not traverse symbolic links, but compares the links themselves.

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

See Also
- `contentsAtPath:` (page 633)

Declared In
`NSFileManager.h`

contentsOfDirectoryAtPath:error:

Returns an array of `NSString` objects identifying the directories and files (including symbolic links) contained in a given directory.

```
- (NSArray *)contentsOfDirectoryAtPath:(NSString *)path error:(NSError **)error
```

Parameters

path

A path to a directory.

error

If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value

An array of `NSString` objects identifying the directories and files (including symbolic links) contained in *path*. Returns an empty array if the directory exists but has no contents. Returns `nil` if the directory specified at *path* does not exist or there is some other error accessing it.

Discussion

The search is shallow and therefore does not return the contents of any subdirectories. This returned array does not contain strings for the current directory ("."), parent directory (".."), or resource forks (begin with "._") and does not traverse symbolic links.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `directoryContentsAtPath:` (page 642)
- `currentDirectoryPath` (page 641)
- `fileExistsAtPath:isDirectory:` (page 647)
- `enumeratorAtPath:` (page 644)
- `subpathsAtPath:` (page 659)

Declared In

`NSFileManager.h`

copyItemAtPath:toPath:error:

Copies the directory or file specified in a given path to a different location in the file system identified by another path.

```
- (BOOL)copyItemAtPath:(NSString *)srcPath toPath:(NSString *)dstPath error:(NSError
    **)error
```

Parameters

srcPath

The path of a file or directory.

dstPath

The path of a file or directory.

error

If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value

`YES` if the operation was successful, otherwise `NO`.

Availability

Available in Mac OS X v10.5 and later.

See Also
- `fileManager:shouldCopyItemAtPath:toPath:` (page 661)
- `fileManager:shouldProceedAfterError:copyingItemAtPath:toPath:` (page 664)
- `linkItemAtPath:toPath:error:` (page 651)
- `moveItemAtPath:toPath:error:` (page 654)
- `removeItemAtPath:error:` (page 657)
- `copyPath:toPath:handler:` (page 636)

Declared In
`NSFileManager.h`

copyPath:toPath:handler:

Copies the directory or file specified in a given path to a different location in the file system identified by another path.

```
- (BOOL)copyPath:(NSString *)source  toPath:(NSString *)destination
    handler:(id)handler
```

Parameters

source
The location of the source file.

destination
The location to which to copy the file specified by *source*.

handler
An object that responds to the callback messages `fileManager:willProcessPath:` (page 667) and `fileManager:shouldProceedAfterError:` (page 663). You can specify `nil` for *handler*; if you do so and an error occurs, the method automatically returns `NO`.

Return Value
`YES` if the copy operation is successful. If the operation is not successful, but the callback handler of `fileManager:shouldProceedAfterError:` (page 663) returns `YES`, `copyPath:toPath:handler:` also returns `YES`. Otherwise this method returns `NO`. The method also attempts to make the attributes of the directory or file at *destination* identical to *source*, but ignores any failure at this attempt.

Discussion
If *source* is a file, the method creates a file at *destination* that holds the exact contents of the original file (this includes BSD special files). If *source* is a directory, the method creates a new directory at *destination* and recursively populates it with duplicates of the files and directories contained in *source*, preserving all links. The file specified in *source* must exist, while *destination* must not exist prior to the operation. When a file is being copied, the destination path must end in a filename—there is no implicit adoption of the source filename. Symbolic links are not traversed but are themselves copied. File or directory attributes—that is, metadata such as owner and group numbers, file permissions, and modification date—are also copied.

The handler callback mechanism is similar to delegation. `NSFileManager` sends `fileManager:willProcessPath:` (page 667) when it begins a copy, move, remove, or link operation. It sends `fileManager:shouldProceedAfterError:` (page 663) when it encounters any error in processing.

This code fragment verifies that the file to be copied exists and then copies that file to the user's `~/Library/Reports` directory:

```
NSString *source = @"/tmp/quarterly_report.rtf";
NSString *destination = [[[NSHomeDirectory()
        stringByAppendingPathComponent:@"Library"]
        stringByAppendingPathComponent:@"Reports"]
        stringByAppendingPathComponent:@"new_quarterly_report.rtf"];
NSFileManager *fileManager = [NSFileManager defaultManager];

if ([fileManager fileExistsAtPath:source]) {
    [fileManager copyPath:source toPath:destination handler:nil];
}
```

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

See Also
- `linkPath:toPath:handler:` (page 652)
- `movePath:toPath:handler:` (page 654)
- `fileManager:shouldProceedAfterError:` (page 663)
- `removeFileAtPath:handler:` (page 656)
- `fileManager:willProcessPath:` (page 667)

Related Sample Code
Core Data HTML Store

Declared In
`NSFileManager.h`

createDirectoryAtPath:attributes:

Creates a directory (without contents) at a given path with given attributes.

```
- (BOOL)createDirectoryAtPath:(NSString *)path attributes:(NSDictionary *)attributes
```

Parameters

path

The path at which to create the new directory. The directory to be created must not yet exist, but its parent directory must exist.

attributes

The file attributes for the new directory. The attributes you can set are owner and group numbers, file permissions, and modification date. If you specify `nil` for *attributes*, default values for these attributes are set (particularly write access for the creator and read access for others). The "Constants" (page 668) section lists the global constants used as keys in the *attributes* dictionary. Some of the keys, such as `NSFileHFSCreatorCode` and `NSFileHFSTypeCode`, do not apply to directories.

Return Value
`YES` if the operation was successful, otherwise `NO`.

Special Considerations

On Mac OS X v10.5 and later, use `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 638) instead.

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

See Also
- `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 638)
- `changeCurrentDirectoryPath:` (page 631)
- `changeFileAttributes:atPath:` (page 632)
- `createFileAtPath:contents:attributes:` (page 639)
- `currentDirectoryPath` (page 641)

Related Sample Code
Core Data HTML Store
CoreRecipes
GridCalendar
MyPhoto
SpotlightFortunes

Declared In
`NSFileManager.h`

createDirectoryAtPath:withIntermediateDirectories:attributes:error:

Creates a directory with given attributes at a specified path.

```
- (BOOL)createDirectoryAtPath:(NSString *)path
    withIntermediateDirectories:(BOOL)createIntermediates attributes:(NSDictionary
     *)attributes error:(NSError **)error
```

Parameters

path
The path at which to create the new directory. The directory to be created must not yet exist.

createIntermediates
If `YES`, then the method will also create any necessary intermediate directories; if `NO`, then the method will fail if any parent of the directory to be created does not exist.

attributes
The file attributes for the new directory. The attributes you can set are owner and group numbers, file permissions, and modification date. If you specify `nil` for *attributes*, the directory is created according to the umask of the process. The "Constants" (page 668) section lists the global constants used as keys in the *attributes* dictionary. Some of the keys, such as `NSFileHFSCreatorCode` and `NSFileHFSTypeCode`, do not apply to directories.

error
If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value
`YES` if the operation was successful, otherwise `NO`.

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

See Also
- `createDirectoryAtPath:attributes:` (page 637)
- `changeCurrentDirectoryPath:` (page 631)
- `setAttributes:ofItemAtPath:error:` (page 658)
- `createFileAtPath:contents:attributes:` (page 639)
- `currentDirectoryPath` (page 641)

Declared In
`NSFileManager.h`

createFileAtPath:contents:attributes:

Creates a file at a given path that has given attributes and contents.

```
- (BOOL)createFileAtPath:(NSString *)path   contents:(NSData *)contents
    attributes:(NSDictionary *)attributes
```

Parameters

path
The path for the new file.

contents
The contents for the new file.

attributes
A dictionary that describes the attributes of the new file. The file attributes you can set are owner and group numbers, file permissions, and modification date. "File Attribute Keys" (page 668) lists the global constants used as keys in the *attributes* dictionary. If you specify `nil` for *attributes*, the file is given a default set of attributes.

Return Value
`YES` if the operation was successful, otherwise `NO`.

Discussion
If a file already exists at *path*, then if the file can be overwritten (subject to user privileges) it will be.

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

See Also
- `contentsAtPath:` (page 633)
- `changeFileAttributes:atPath:` (page 632)
- `setAttributes:ofItemAtPath:error:` (page 658)
- `fileAttributesAtPath:traverseLink:` (page 645)
- `attributesOfItemAtPath:error:` (page 631)

Related Sample Code
Core Data HTML Store
CustomAtomicStoreSubclass
TimelineToTC

Declared In
`NSFileManager.h`

createSymbolicLinkAtPath:pathContent:

Creates a symbolic link identified by a given path that refers to a given location.

```
- (BOOL)createSymbolicLinkAtPath:(NSString *)path  pathContent:(NSString *)otherPath
```

Parameters

path

The path for a symbolic link.

otherPath

The path to which *path* should refer.

Return Value

`YES` if the operation is successful, otherwise `NO`. Returns `NO` if a file, directory, or symbolic link identical to *path* already exists.

Discussion

Creates a symbolic link identified by *path* that refers to the location *otherPath* in the file system.

Special Considerations

On Mac OS X v10.5 and later, use `createSymbolicLinkAtPath:withDestinationPath:error:` (page 640) instead.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `createSymbolicLinkAtPath:withDestinationPath:error:` (page 640)
- `pathContentOfSymbolicLinkAtPath:` (page 656)
- `linkPath:toPath:handler:` (page 652)

Declared In

`NSFileManager.h`

createSymbolicLinkAtPath:withDestinationPath:error:

Creates a symbolic link identified by a given path that refers to a given location.

```
- (BOOL)createSymbolicLinkAtPath:(NSString *)path withDestinationPath:(NSString
    *)destPath error:(NSError **)error
```

Parameters

path

The path for a symbolic link.

destPath

The path to which *path* should refer.

error

If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value

`YES` if the operation is successful, otherwise `NO`. Returns `NO` if a file, directory, or symbolic link identical to *path* already exists.

Discussion
Creates a symbolic link identified by *path* that refers to the location *destPath* in the file system.

This method does not traverse an initial symlink.

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

See Also
- `createSymbolicLinkAtPath:pathContent:` (page 640)
- `pathContentOfSymbolicLinkAtPath:` (page 656)
- `linkPath:toPath:handler:` (page 652)

Declared In
`NSFileManager.h`

currentDirectoryPath

Returns the path of the program's current directory.

```
- (NSString *)currentDirectoryPath
```

Return Value
The path of the program's current directory. If the program's current working directory isn't accessible, returns `nil`.

Discussion
The string returned by this method is initialized to the current working directory; you can change the working directory by invoking `changeCurrentDirectoryPath:` (page 631).

Relative pathnames refer implicitly to the current directory. For example, if the current directory is `/tmp`, and the relative pathname `reports/info.txt` is specified, the resulting full pathname is `/tmp/reports/info.txt`.

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

See Also
- `changeCurrentDirectoryPath:` (page 631)
- `createDirectoryAtPath:attributes:` (page 637)
- `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 638)

Declared In
`NSFileManager.h`

delegate

Returns the delegate for the receiver.

```
- (id)delegate
```

Return Value
The delegate for the receiver.

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

Declared In
NSFileManager.h

destinationOfSymbolicLinkAtPath:error:

Returns an NSString object containing the path of the item pointed at by the symlink specified by a given path.

```
- (NSString *)destinationOfSymbolicLinkAtPath:(NSString *)path error:(NSError
    **)error
```

Parameters

path

The path of a file or directory.

error

If an error occurs, upon return contains an NSError object that describes the problem. Pass NULL if you do not want error information.

Return Value
An NSString object containing the path of the directory or file to which the symbolic link *path* refers, or nil upon failure. If the symbolic link is specified as a relative path, that relative path is returned.

Discussion
This method does not traverse an initial symlink.

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

See Also
- pathContentOfSymbolicLinkAtPath: (page 656)
- createSymbolicLinkAtPath:withDestinationPath:error: (page 640)

Declared In
NSFileManager.h

directoryContentsAtPath:

Returns an array of NSString objects identifying the directories and files (including symbolic links) contained in a given directory.

```
- (NSArray *)directoryContentsAtPath:(NSString *)path
```

Parameters

path

A path to a directory.

Return Value
An array of NSString objects identifying the directories and files (including symbolic links) contained in *path*. Returns an empty array if the directory exists but has no contents. Returns nil if the directory specified at *path* does not exist or there is some other error accessing it.

Discussion
The search is shallow and therefore does not return the contents of any subdirectories. This returned array does not contain strings for the current directory (“.”), parent directory (“..”), or resource forks (begin with “._”) and does not traverse symbolic links.

Special Considerations
On Mac OS X v10.5 and later, use `contentsOfDirectoryAtPath:error:` (page 634) instead.

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

See Also
- `contentsOfDirectoryAtPath:error:` (page 634)
- `currentDirectoryPath` (page 641)
- `fileExistsAtPath:isDirectory:` (page 647)
- `enumeratorAtPath:` (page 644)
- `subpathsAtPath:` (page 659)

Related Sample Code
Core Data HTML Store
IKSlideshowDemo
ImageBrowser
LSMSmartCategorizer
ThreadsImportMovie

Declared In
`NSFileManager.h`

displayNameAtPath:

Returns the name of the file or directory at a given path in a localized form appropriate for presentation to the user.

```
- (NSString *)displayNameAtPath:(NSString *)path
```

Parameters
path
The path of a file or directory.

Return Value
The name of the file or directory at *path* in a localized form appropriate for presentation to the user. If there is no file or directory at *path*, or if an error occurs, returns `[path lastPathComponent]`.

Discussion
The returned value is localized where appropriate. For example, if you have selected French as your preferred language, the following code fragment logs “Bibliothèque”:

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSLibraryDirectory,
NSUserDomainMask, YES);
if ([paths count] > 0)
{
    NSString *documentsDirectory = [paths objectAtIndex:0];
    NSFileManager *fileManager = [NSFileManager defaultManager];
```

```
    NSString *displayNameAtPath = [fileManager
displayNameAtPath:documentsDirectory];
    NSLog(@"%@", displayNameAtPath);
}
```

Availability
Available in Mac OS X v10.1 and later.

See Also
`- lastPathComponent` (page 1579) (`NSString`)

Related Sample Code
AlbumToSlideshow
AutomatorHandsOn
DeskPictAppDockMenu
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSFileManager.h`

enumeratorAtPath:

Creates and returns an `NSDirectoryEnumerator` object that enumerates the contents of the directory at a given path.

```
- (NSDirectoryEnumerator *)enumeratorAtPath:(NSString *)path
```

Parameters

path
The path of the directory to enumerate.

Return Value
An `NSDirectoryEnumerator` object that enumerates the contents of the directory at *path*. If *path* is a symbolic link, this method evaluates the link and returns an enumerator for the file or directory the link points to. If the link cannot be evaluated, the method returns `nil`.

If *path* is a filename, the method returns an enumerator object that enumerates no files—the first call to `nextObject` (page 558) will return `nil`.

Discussion
Because the enumeration is deep—that is, it lists the contents of all subdirectories—this enumerator object is useful for performing actions that involve large file-system subtrees. If the method is passed a directory on which another file system is mounted (a mount point), it traverses the mount point. This method does not resolve symbolic links encountered in the traversal process, nor does it recurse through them if they point to a directory.

This code fragment enumerates the subdirectories and files under a user's `Documents` directory and processes all files with an extension of `.doc`:

```
NSString *file;
NSString *docsDir = [NSHomeDirectory() stringByAppendingPathComponent:
@"Documents"];
NSDirectoryEnumerator *dirEnum =
```

```
    [[NSFileManager defaultManager] enumeratorAtPath:docsDir];

while (file = [dirEnum nextObject]) {
    if ([[file pathExtension] isEqualToString: @"doc"]) {
        [self scanDocument: [docsDir stringByAppendingPathComponent:file]];
    }
}
```

The `NSDirectoryEnumerator` class has methods for obtaining the attributes of the existing path and of the parent directory and for skipping descendants of the existing path.

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

See Also
- `currentDirectoryPath` (page 641)
- `fileAttributesAtPath:traverseLink:` (page 645)
- `directoryContentsAtPath:` (page 642)
- `subpathsAtPath:` (page 659)

Related Sample Code
BundleLoader
DeskPictAppDockMenu
NSOperationSample

Declared In
`NSFileManager.h`

fileAttributesAtPath:traverseLink:

Returns a dictionary that describes the POSIX attributes of the file specified at a given.

```
- (NSDictionary *)fileAttributesAtPath:(NSString *)path  traverseLink:(BOOL)flag
```

Parameters

path

A file path.

flag

If *path* is not a symbolic link, this parameter has no effect. If *path* is a symbolic link, then:

- If `YES` the attributes of the linked-to file are returned, or if the link points to a nonexistent file the method returns `nil`.
- If `NO`, the attributes of the symbolic link are returned.

Return Value
An `NSDictionary` object that describes the POSIX attributes of the file specified at *path*. The keys in the dictionary are described in "`File Attribute Keys`" (page 668). If there is no item at *path*, returns `nil`.

Discussion
This code example gets several attributes of a file and logs them.

```
NSFileManager *fileManager = [NSFileManager defaultManager];
NSString *path = @"/tmp/List";
```

```
NSDictionary *fileAttributes = [fileManager fileAttributesAtPath:path
traverseLink:YES];

if (fileAttributes != nil) {
    NSNumber *fileSize;
    NSString *fileOwner;
    NSDate *fileModDate;
    if (fileSize = [fileAttributes objectForKey:NSFileSize]) {
        NSLog(@"File size: %qi\n", [fileSize unsignedLongLongValue]);
    }
    if (fileOwner = [fileAttributes objectForKey:NSFileOwnerAccountName]) {
        NSLog(@"Owner: %@\n", fileOwner);
    }
    if (fileModDate = [fileAttributes objectForKey:NSFileModificationDate]) {
        NSLog(@"Modification date: %@\n", fileModDate);
    }
}
else {
    NSLog(@"Path (%@) is invalid.", path);
}
```

As a convenience, `NSDictionary` provides a set of methods (declared as a category in `NSFileManager.h`) for quickly and efficiently obtaining attribute information from the returned dictionary: `fileGroupOwnerAccountName` (page 509), `fileModificationDate` (page 511), `fileOwnerAccountName` (page 512), `filePosixPermissions` (page 512), `fileSize` (page 512), `fileSystemFileNumber` (page 513), `fileSystemNumber` (page 513), and `fileType` (page 514). For example, you could rewrite the file modification statement in the code example above as:

```
if (fileModDate = [fileAttributes fileModificationDate])
    NSLog(@"Modification date: %@\n", fileModDate);
```

Special Considerations

On Mac OS X v10.5 and later, use `attributesOfItemAtPath:error:` (page 631) instead.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `attributesOfItemAtPath:error:` (page 631)
- `changeFileAttributes:atPath:` (page 632)

Related Sample Code

AudioBurn
DeskPictAppDockMenu
Quartz Composer WWDC 2005 TextEdit
TextEditPlus
ThreadsImportMovie

Declared In

`NSFileManager.h`

fileExistsAtPath:

Returns a Boolean value that indicates whether a file or directory exists at a specified path.

```
- (BOOL)fileExistsAtPath:(NSString *)path
```

Parameters

path

The path of a file or directory. If *path* begins with a tilde (~), it must first be expanded with `stringByExpandingTildeInPath` (page 1602), or this method will return `NO`.

Return Value

`YES` if a file specified in *path* exists, otherwise `NO`. If the final element in *path* specifies a symbolic link, this method traverses the link and returns `YES` or `NO` based on the existence of the file at the link destination.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- fileExistsAtPath:isDirectory:` (page 647)

Related Sample Code

CoreRecipes

QTKitCreateMovie

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In

`NSFileManager.h`

fileExistsAtPath:isDirectory:

Returns a Boolean value that indicates whether a file or directory exists at a specified path.

```
- (BOOL)fileExistsAtPath:(NSString *)path   isDirectory:(BOOL *)isDirectory
```

Parameters

path

The path of a file or directory. If *path* begins with a tilde (~), it must first be expanded with `stringByExpandingTildeInPath` (page 1602), or this method will return `NO`.

isDirectory

Upon return, contains `YES` if *path* is a directory or if the final path element is a symbolic link that points to a directory, otherwise contains `NO`. If *path* doesn't exist, the return value is undefined. Pass `NULL` if you do not need this information.

Return Value

`YES` if there is a file or directory at *path*, otherwise `NO`. If the final element in *path* specifies a symbolic link, this method traverses the link and returns `YES` or `NO` based on the existence of the file or directory at the link destination.

Discussion

If you need to further determine if *path* is a package, use the `NSWorkspace` method `isFilePackageAtPath:`.

This example gets an array that identifies the fonts in the user's fonts directory:

```
NSArray *subpaths;
BOOL isDir;
```

```
NSArray *paths = NSSearchPathForDirectoriesInDomains
                    (NSLibraryDirectory, NSUserDomainMask, YES);

if ([paths count] == 1) {

    NSFileManager *fileManager = [NSFileManager defaultManager];
    NSString *fontPath = [[paths objectAtIndex:0]
stringByAppendingPathComponent:@"Fonts"];

    if ([fileManager fileExistsAtPath:fontPath isDirectory:&isDir] && isDir) {
        subpaths = [fileManager subpathsAtPath:fontPath];
// ...
```

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

See Also
`- fileExistsAtPath:` (page 646)

Related Sample Code
ImageBrowser
LSMSmartCategorizer
QTKitAdvancedDocument
QTKitImport
QTKitPlayer

Declared In
`NSFileManager.h`

fileSystemAttributesAtPath:

Returns a dictionary that describes the attributes of the mounted file system on which a given path resides.

```
- (NSDictionary *)fileSystemAttributesAtPath:(NSString *)path
```

Parameters
path
Any pathname within the mounted file system.

Return Value
An `NSDictionary` object that describes the attributes of the mounted file system on which *path* resides. See "File-System Attribute Keys" (page 672) for a description of the keys available in the dictionary.

Discussion
The following code example checks to see if there's sufficient space on the file system before adding a new file to it:

```
NSData *contents = [myImage TIFFRepresentation];
NSFileManager *fileManager = [NSFileManager defaultManager];
NSString *path = ...;
NSString *fileName = ...;
NSDictionary *fsAttributes =
        [fileManager fileSystemAttributesAtPath:path];
if ([[fsAttributes objectForKey:NSFileSystemFreeSize] unsignedLongLongValue]
>
```

```
    [contents length])
[fileManager createFileAtPath:[path stringByAppendingPathComponent:fileName]
        contents:contents attributes:nil];
```

Special Considerations

On Mac OS X v10.5 and later, use `attributesOfFileSystemForPath:error:` (page 630) instead.

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

See Also
- `attributesOfFileSystemForPath:error:` (page 630)
- `fileAttributesAtPath:traverseLink:` (page 645)
- `changeFileAttributes:atPath:` (page 632)

Declared In
`NSFileManager.h`

fileSystemRepresentationWithPath:

Returns a C-string representation of a given path that properly encodes Unicode strings for use by the file system.

```
- (const char *)fileSystemRepresentationWithPath:(NSString *)path
```

Parameters

path
A file path.

Return Value
A C-string representation of *path* that properly encodes Unicode strings for use by the file system.

Discussion
If you need the C string beyond the scope of your autorelease pool, you must copy it. This method raises an exception upon error. Use this method if your code calls system routines that expect C-string path arguments.

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

See Also
- `stringWithFileSystemRepresentation:length:` (page 659)

Declared In
`NSFileManager.h`

isDeletableFileAtPath:

Returns a Boolean value that indicates whether the invoking object appears able to delete a specified file.

```
- (BOOL)isDeletableFileAtPath:(NSString *)path
```

Parameters

path

A file path.

Return Value

`YES` if the invoking object appears able to delete the file specified in *path*, otherwise `NO`. If the file at *path* does not exist, this method returns `NO`.

Discussion

For a directory or file to be able to be deleted, either the parent directory of *path* must be writable or its owner must be the same as the owner of the application process. If *path* is a directory, every item contained in *path* must be able to be deleted.

This method does not traverse symbolic links.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSFileManager.h`

isExecutableFileAtPath:

Returns a Boolean value that indicates whether the operating system appears able to execute a specified file.

```
- (BOOL)isExecutableFileAtPath:(NSString *)path
```

Parameters

path

A file path.

Return Value

`YES` if the operating system appears able to execute the file specified in *path*, otherwise `NO`. If the file at *path* does not exist, this method returns `NO`.

Discussion

This method traverses symbolic links. This method uses the real user ID and group ID, as opposed to the effective user and group IDs, to determine if the file is executable.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSFileManager.h`

isReadableFileAtPath:

Returns a Boolean value that indicates whether the invoking object appears able to read a specified file.

```
- (BOOL)isReadableFileAtPath:(NSString *)path
```

Parameters

path

A file path.

Return Value

YES if the invoking object appears able to read the file specified in *path*, otherwise NO. If the file at *path* does not exist, this method returns NO.

Discussion

This method traverses symbolic links. This method uses the real user ID and group ID, as opposed to the effective user and group IDs, to determine if the file is readable.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTQuartzPlayer

Declared In

NSFileManager.h

isWritableFileAtPath:

Returns a Boolean value that indicates whether the invoking object appears able to write to a specified file.

```
- (BOOL)isWritableFileAtPath:(NSString *)path
```

Parameters

path

A file path.

Return Value

YES if the invoking object appears able to write to the file specified in *path*, otherwise NO. If the file at *path* does not exist, this method returns NO.

Discussion

This method traverses symbolic links. This method uses the real user ID and group ID, as opposed to the effective user and group IDs, to determine if the file is writable.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSFileManager.h

linkItemAtPath:toPath:error:

Creates a link from a source to a destination.

```
- (BOOL)linkItemAtPath:(NSString *)srcPath toPath:(NSString *)dstPath error:(NSError
    **)error
```

Parameters

srcPath

A path that identifies a source file.

The file or link specified by *srcPath* must exist. *srcPath* must not identify a directory.

dstPath

A path that identifies a destination file or directory on the same filesystem as *srcPath*.

The destination should not yet exist. The destination path must end in a filename; there is no implicit adoption of the source filename.

error

If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value

`YES` if the link operation is successful, otherwise `NO`.

Discussion

If pathname *srcPath* identifies a file, this method hard-links the file specified in *dstPath* to it. If *srcPath* is a symbolic link, this method copies it to *dstPath* instead of creating a hard link. Symbolic links in *srcPath* are not traversed.

Amongst other reasons (such as the disk being full, permissions problems, and so on), this method will fail if:

- *srcPath* doesn't point to any file in the file system;
- *srcPath* points to an existing symbolic link, but the symbolic link is “broken" (it doesn't in turn point to an existing regular file in the file system);
- *srcPath* points to a directory;
- The computer has more than one file system (such as extra partitions, mounted disk images, or network volumes), and *srcPath* and *dstPath* specify paths in different file systems.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `- fileManager:shouldLinkItemAtPath:toPath:` (page 662)
- `- fileManager:shouldProceedAfterError:linkingItemAtPath:toPath:` (page 665)
- `- createSymbolicLinkAtPath:withDestinationPath:error:` (page 640)
- `- copyItemAtPath:toPath:error:` (page 635)
- `- moveItemAtPath:toPath:error:` (page 654)
- `- linkPath:toPath:handler:` (page 652)

Declared In

`NSFileManager.h`

linkPath:toPath:handler:

Creates a link from a source to a destination.

```
- (BOOL)linkPath:(NSString *)source  toPath:(NSString *)destination
    handler:(id)handler
```

Parameters

source

A path that identifies a source file or directory.

The file, link, or directory specified by *source* must exist.

destination

A path that identifies a destination file or directory.

The destination should not yet exist. The destination path must end in a filename; there is no implicit adoption of the source filename.

handler

An object that responds to the callback messages `fileManager:willProcessPath:` (page 667) and `fileManager:shouldProceedAfterError:` (page 663). You can specify `nil` for *handler*; if you do so and an error occurs, the method automatically returns `NO`.

Return Value

`YES` if the link operation is successful. If the operation is not successful, but the handler method `fileManager:shouldProceedAfterError:` (page 663) returns `YES`, also returns `YES`. Otherwise returns `NO`.

Discussion

If pathname *source* identifies a file, this method hard-links the file specified in *destination* to it. If *source* is a directory or symbolic link, this method copies it to *destination* instead of creating a hard link. Symbolic links in *source* are not traversed.

The handler callback mechanism is similar to delegation. `NSFileManager` sends `fileManager:willProcessPath:` (page 667) when it begins a copy, move, remove, or link operation. It sends `fileManager:shouldProceedAfterError:` (page 663) when it encounters any error in processing

This code fragment verifies the pathname typed in a text field (`documentFileField`) and then links the file to the user's Documents directory:

```
NSString *source = [documentFileField stringValue];

NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
if ([paths count] > 0)
{
    NSString *documentsDirectory = [paths objectAtIndex:0];
    NSString *documentFileName = [source lastPathComponent];
    NSString *destination = [documentsDirectory
stringByAppendingPathComponent:documentFileName];
    NSFileManager *fileManager = [NSFileManager defaultManager];

    if ([fileManager fileExistsAtPath:source])
    {
        [fileManager linkPath:source toPath:destination handler:self];
    }
}
```

Availability

Available in Mac OS X v10.0 and later.

See Also
- `linkItemAtPath:toPath:error:` (page 651)
- `copyPath:toPath:handler:` (page 636)
- `createSymbolicLinkAtPath:pathContent:` (page 640)
- `movePath:toPath:handler:` (page 654)
- `fileManager:shouldProceedAfterError:` (page 663)
- `removeFileAtPath:handler:` (page 656)
- `fileManager:willProcessPath:` (page 667)

Declared In
`NSFileManager.h`

moveItemAtPath:toPath:error:

Moves the directory or file specified by a given path to a different location in the file system identified by another path.

```
- (BOOL)moveItemAtPath:(NSString *)srcPath toPath:(NSString *)dstPath error:(NSError
    **)error
```

Parameters

srcPath
The path of a file or directory to move. *srcPath* must exist.

dstPath
The path to which the file or directory at *srcPath* is moved. *destination* must not yet exist. The destination path must end in a filename; there is no implicit adoption of the source filename.

error
If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value
`YES` if the move operation is successful, otherwise `NO`.

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

See Also
- `fileManager:shouldMoveItemAtPath:toPath:` (page 662)
- `fileManager:shouldProceedAfterError:movingItemAtPath:toPath:` (page 665)

Declared In
`NSFileManager.h`

movePath:toPath:handler:

Moves the directory or file specified by a given path to a different location in the file system identified by another path.

```
- (BOOL)movePath:(NSString *)source  toPath:(NSString *)destination
    handler:(id)handler
```

Parameters

source

The path of a file or directory to move. *source* must exist.

destination

The path to which *source* is moved. *destination* must not yet exist. The destination path must end in a filename; there is no implicit adoption of the source filename.

handler

An object that responds to the callback messages `fileManager:willProcessPath:` (page 667) and `fileManager:shouldProceedAfterError:` (page 663). You can specify `nil` for *handler*; if you do so and an error occurs, the method automatically returns `NO`.

Return Value

`YES` if the move operation is successful. If the operation is not successful, but the handler method `fileManager:shouldProceedAfterError:` (page 663) returns `YES`, `movePath:toPath:handler:` (page 654) also returns `YES`; otherwise returns `NO`.

Discussion

If *source* is a file, the method creates a file at *destination* that holds the exact contents of the original file and then deletes the original file. If *source* is a directory, `movePath:toPath:handler:` creates a new directory at *destination* and recursively populates it with duplicates of the files and directories contained in *source*. It then deletes the old directory and its contents. Symbolic links are not traversed, however links are preserved. File or directory attributes—that is, metadata such as owner and group numbers, file permissions, and modification date—are also moved.

The handler callback mechanism is similar to delegation. `NSFileManager` sends `fileManager:willProcessPath:` (page 667) when it begins a copy, move, remove, or link operation. It sends `fileManager:shouldProceedAfterError:` (page 663) when it encounters any error in processing.

If a failure in a move operation occurs, either the preexisting path or the new path remains intact, but not both.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `copyPath:toPath:handler:` (page 636)
- `linkPath:toPath:handler:` (page 652)
- `removeFileAtPath:handler:` (page 656)
- `fileManager:shouldProceedAfterError:` (page 663)
- `fileManager:willProcessPath:` (page 667)

Related Sample Code

QTRecorder

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

WhackedTV

Declared In

`NSFileManager.h`

pathContentOfSymbolicLinkAtPath:

Returns the path of the directory or file that a symbolic link at a given path refers to.

```
- (NSString *)pathContentOfSymbolicLinkAtPath:(NSString *)path
```

Parameters

path

The path of a symbolic link.

Return Value

The path of the directory or file to which the symbolic link *path* refers, or `nil` upon failure. If the symbolic link is specified as a relative path, that relative path is returned.

Special Considerations

On Mac OS X v10.5 and later, use `destinationOfSymbolicLinkAtPath:error:` (page 642) instead.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `destinationOfSymbolicLinkAtPath:error:` (page 642)
- `createSymbolicLinkAtPath:pathContent:` (page 640)

Declared In

`NSFileManager.h`

removeFileAtPath:handler:

Deletes the file, link, or directory (including, recursively, all subdirectories, files, and links in the directory) identified by a given path.

```
- (BOOL)removeFileAtPath:(NSString *)path  handler:(id)handler
```

Parameters

path

The path of a file, link, or directory to delete. The value must not be "." or "..".

handler

An object that responds to the callback messages `fileManager:willProcessPath:` (page 667) and `fileManager:shouldProceedAfterError:` (page 663). You can specify `nil` for *handler*; if you do so and an error occurs, the deletion stops and the method automatically returns `NO`.

Return Value

`YES` if the removal operation is successful. If the operation is not successful, but the handler method `fileManager:shouldProceedAfterError:` (page 663) returns `YES`, also returns `YES`; otherwise returns `NO`.

Discussion

This callback mechanism provided by *handler* is similar to delegation. `NSFileManager` sends `fileManager:willProcessPath:` (page 667) when it begins a copy, move, remove, or link operation. It sends `fileManager:shouldProceedAfterError:` (page 663) when it encounters any error in processing.

Since the removal of directory contents is so thorough and final, be careful when using this method. If you specify "." or ".." for *path* an `NSInvalidArgumentException` exception is raised. This method does not traverse symbolic links.

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

See Also
- `removeItemAtPath:error:` (page 657)
- `copyPath:toPath:handler:` (page 636)
- `linkPath:toPath:handler:` (page 652)
- `movePath:toPath:handler:` (page 654)
- `fileManager:shouldProceedAfterError:` (page 663)
- `fileManager:willProcessPath:` (page 667)

Related Sample Code
AutoUpdater
CIVideoDemoGL
Core Data HTML Store
CustomAtomicStoreSubclass
SampleScannerApp

Declared In
`NSFileManager.h`

removeItemAtPath:error:

Deletes the file, link, or directory (including, recursively, all subdirectories, files, and links in the directory) identified by a given path.

```
- (BOOL)removeItemAtPath:(NSString *)path error:(NSError **)error
```

Parameters

path
The path of a file, link, or directory to delete. The value must not be "." or "..".

error
If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value
`YES` if the removal operation is successful, otherwise `NO`.

Discussion
Since the removal of directory contents is so thorough and final, be careful when using this method. If you specify "." or ".." for *path* an `NSInvalidArgumentException` exception is raised. This method does not traverse symbolic links.

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

See Also
- `copyItemAtPath:toPath:error:` (page 635)

- `linkItemAtPath:toPath:error:` (page 651)
- `moveItemAtPath:toPath:error:` (page 654)
- `fileManager:shouldRemoveItemAtPath:` (page 666)
- `fileManager:shouldProceedAfterError:removingItemAtPath:` (page 666)
- `removeFileAtPath:handler:` (page 656)

Related Sample Code
URL CacheInfo

Declared In
`NSFileManager.h`

setAttributes:ofItemAtPath:error:

Sets the attributes of a given file or directory.

```
- (BOOL)setAttributes:(NSDictionary *)attributes ofItemAtPath:(NSString *)path
    error:(NSError **)error
```

Parameters

attributes

A dictionary containing as keys the attributes to set for *path* and as values the corresponding value for the attribute. You can set following: `NSFileBusy`, `NSFileCreationDate`, `NSFileExtensionHidden`, `NSFileGroupOwnerAccountID`, `NSFileGroupOwnerAccountName`, `NSFileHFSCreatorCode`, `NSFileHFSTypeCode`, `NSFileImmutable`, `NSFileModificationDate`, `NSFileOwnerAccountID`, `NSFileOwnerAccountName`, `NSFilePosixPermissions`. You can change single attributes or any combination of attributes; you need not specify keys for all attributes.

path

The path of a file or directory.

error

If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value
`YES` if *all* changes succeed. If any change fails, returns `NO`, but it is undefined whether any changes actually occurred.

Discussion
As in the POSIX standard, the application either must own the file or directory or must be running as superuser for attribute changes to take effect. The method attempts to make all changes specified in attributes and ignores any rejection of an attempted modification.

The `NSFilePosixPermissions` value must be initialized with the code representing the POSIX file-permissions bit pattern. `NSFileHFSCreatorCode` and `NSFileHFSTypeCode` will only be heeded when *path* specifies a file.

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

Declared In
`NSFileManager.h`

setDelegate:

Sets the delegate for the receiver.

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

Parameters

delegate

The delegate for the receiver.

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

Declared In
NSFileManager.h

stringWithFileSystemRepresentation:length:

Returns an `NSString` object converted from the C-string representation of a pathname in the current file system.

```
- (NSString *)stringWithFileSystemRepresentation:(const char *)string
    length:(NSUInteger)len
```

Parameters

string

A C string representation of a pathname.

len

The number of characters in *string*.

Return Value
An `NSString` object converted from the C-string representation *string* with length *len* of a pathname in the current file system.

Discussion
Use this method if your code receives paths as C strings from system routines.

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

See Also
`- fileSystemRepresentationWithPath:` (page 649)

Declared In
NSFileManager.h

subpathsAtPath:

Returns an array that contains (as `NSString` objects) the contents of the directory identified by a given path.

```
- (NSArray *)subpathsAtPath:(NSString *)path
```

Parameters

path

The path of the directory to list.

Return Value

An array that contains (as `NSString` objects) the contents of the directory identified by *path*. If *path* is a symbolic link, `subpathsAtPath:` traverses the link. Returns `nil` if it cannot get the device of the linked-to file.

Discussion

This list of directory contents goes very deep and hence is very useful for large file-system subtrees. The method skips "." and "..".

This method reveals every element of the subtree at *path*, including the contents of file packages (such as applications, nib files, and RTFD files). This code fragment gets the contents of `/System/Library/Fonts` after verifying that the directory exists:

```
BOOL isDir=NO;
NSArray *subpaths;
NSString *fontPath = @"/System/Library/Fonts";
NSFileManager *fileManager = [NSFileManager defaultManager];
if ([fileManager fileExistsAtPath:fontPath isDirectory:&isDir] && isDir)
    subpaths = [fileManager subpathsAtPath:fontPath];
```

Special Considerations

On Mac OS X v10.5 and later, use `subpathsOfDirectoryAtPath:error:` (page 660) instead.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `subpathsOfDirectoryAtPath:error:` (page 660)
- `directoryContentsAtPath:` (page 642)
- `enumeratorAtPath:` (page 644)

Declared In

`NSFileManager.h`

subpathsOfDirectoryAtPath:error:

Returns an array that contains the filenames of the items in the directory specified by a given path and all its subdirectories recursively.

```
- (NSArray *)subpathsOfDirectoryAtPath:(NSString *)path error:(NSError **)error
```

Parameters

path

The path of the directory to list.

error

If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value
An array that contains `NSString` objects representing the filenames of the items in the directory specified by *`path`* and all its subdirectories recursively. If *`path`* is a symbolic link, `subpathsOfDirectoryAtPath:error:` traverses the link. Returns `nil` if it cannot get the device of the linked-to file.

Discussion
This list of directory contents goes very deep and hence is very useful for large file-system subtrees. The method skips "`.`" and "`..`".

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

See Also
- `subpathsAtPath:` (page 659)
- `directoryContentsAtPath:` (page 642)
- `enumeratorAtPath:` (page 644)

Declared In
`NSFileManager.h`

Delegate Methods

fileManager:shouldCopyItemAtPath:toPath:

An `NSFileManager` object sends this message immediately before attempting to copy to a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager shouldCopyItemAtPath:(NSString
    *)srcPath toPath:(NSString *)dstPath
```

Parameters

`fileManager`
The `NSFileManager` object that sent this message.

`srcPath`
The path or a file or directory that *`manager`* is about to attempt to copy.

`dstPath`
The path or a file or directory to which *`manager`* is about to attempt to copy.

Return Value
`YES` if the operation should proceed, otherwise `NO`.

Discussion
You can implement this method in your delegate to monitor file operations.

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

See Also
- `copyItemAtPath:toPath:error:` (page 635)
- `fileManager:shouldProceedAfterError:copyingItemAtPath:toPath:` (page 664)

Declared In
NSFileManager.h

fileManager:shouldLinkItemAtPath:toPath:

An NSFileManager object sends this message immediately before attempting to link to a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager
    shouldLinkItemAtPath:(NSString *)srcPath
    toPath:(NSString *)dstPath
```

Parameters

fileManager
The NSFileManager object that sent this message.

srcPath
The path or a file or directory that *manager* is about to attempt to link.

dstPath
The path or a file or directory to which *manager* is about to attempt to link.

Return Value
YES if the operation should proceed, otherwise NO.

Discussion
You can implement this method in your delegate to monitor file operations.

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

See Also
- linkItemAtPath:toPath:error: (page 651)
- fileManager:shouldProceedAfterError:linkingItemAtPath:toPath: (page 665)

Declared In
NSFileManager.h

fileManager:shouldMoveItemAtPath:toPath:

An NSFileManager object sends this message immediately before attempting to move to a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager  shouldMoveItemAtPath:(NSString
    *)srcPath  toPath:(NSString *)dstPath
```

Parameters

fileManager
The NSFileManager object that sent this message.

srcPath
The path or a file or directory that *manager* is about to attempt to move.

dstPath
The path or a file or directory to which *manager* is about to attempt to move.

Return Value
YES if the operation should proceed, otherwise NO.

Discussion
You can implement this method in your delegate to monitor file operations.

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

See Also
- `moveItemAtPath:toPath:error:` (page 654)
- `fileManager:shouldProceedAfterError:movingItemAtPath:toPath:` (page 665)

Declared In
`NSFileManager.h`

fileManager:shouldProceedAfterError:

An `NSFileManager` object sends this message to its handler for each error it encounters when copying, moving, removing, or linking files or directories.

```
- (BOOL)fileManager:(NSFileManager *)manager  shouldProceedAfterError:(NSDictionary
    *)errorInfo
```

Parameters

manager
The file manager that sent this message.

errorInfo
A dictionary that contains two or three pieces of information (all `NSString` objects) related to the error:

Key	Value
`@"Path"`	The path related to the error (usually the source path)
`@"Error"`	A description of the error
`@"ToPath"`	The destination path (not all errors)

Return Value
`YES` if the operation (which is often continuous within a loop) should proceed, otherwise `NO`.

Discussion
An `NSFileManager` object, *manager*, sends this message for each error it encounters when copying, moving, removing, or linking files or directories. The return value is passed back to the invoker of `copyPath:toPath:handler:` (page 636), `movePath:toPath:handler:` (page 654), `removeFileAtPath:handler:` (page 656), or `linkPath:toPath:handler:` (page 652). If an error occurs and your handler has not implemented this method, the invoking method automatically returns `NO`.

The following implementation of `fileManager:shouldProceedAfterError:` displays the error string in an alert dialog and leaves it to the user whether to proceed or stop:

```
-(BOOL)fileManager:(NSFileManager *)manager
        shouldProceedAfterError:(NSDictionary *)errorInfo
{
    int result;
```

```
    result = NSRunAlertPanel(@"Gumby App", @"File  operation error:
            %@ with file: %@", @"Proceed", @"Stop",  NULL,
            [errorInfo objectForKey:@"Error"],
            [errorInfo objectForKey:@"Path"]);

    if (result == NSAlertDefaultReturn)
        return YES;
    else
        return NO;
}
```

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

See Also
- `fileManager:willProcessPath:` (page 667)

Declared In
`NSFileManager.h`

fileManager:shouldProceedAfterError:copyingItemAtPath:toPath:

An `NSFileManager` object sends this message if an error occurs during an attempt to copy to a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager shouldProceedAfterError:(NSError
    *)error copyingItemAtPath:(NSString *)srcPath toPath:(NSString *)dstPath
```

Parameters

fileManager
The `NSFileManager` object that sent this message.

error
The error that occurred during the attempt to copy.

srcPath
The path or a file or directory that *manager* is attempting to copy.

dstPath
The path or a file or directory to which *manager* is attempting to copy.

Return Value
`YES` if the operation should proceed, otherwise `NO`.

Discussion
You can implement this method in your delegate to monitor file operations.

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

See Also
- `copyItemAtPath:toPath:error:` (page 635)
- `fileManager:shouldCopyItemAtPath:toPath:` (page 661)

Declared In
`NSFileManager.h`

fileManager:shouldProceedAfterError:linkingItemAtPath:toPath:

An NSFileManager object sends this message if an error occurs during an attempt to link to a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager
    shouldProceedAfterError:(NSError *)error
    linkingItemAtPath:(NSString *)srcPath
    toPath:(NSString *)dstPath
```

Parameters

fileManager

The NSFileManager object that sent this message.

error

The error that occurred during the attempt to link.

srcPath

The path or a file or directory that *manager* is attempting to link.

dstPath

The path or a file or directory to which *manager* is attempting to link.

Return Value

YES if the operation should proceed, otherwise NO.

Availability

Available in Mac OS X v10.5 and later.

See Also

- linkItemAtPath:toPath:error: (page 651)
- fileManager:shouldLinkItemAtPath:toPath: (page 662)

Declared In

NSFileManager.h

fileManager:shouldProceedAfterError:movingItemAtPath:toPath:

An NSFileManager object sends this message if an error occurs during an attempt to move to a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager
    shouldProceedAfterError:(NSError *)error
    movingItemAtPath:(NSString *)srcPath
    toPath:(NSString *)dstPath
```

Parameters

fileManager

The NSFileManager object that sent this message.

error

The error that occurred during the attempt to move.

srcPath

The path or a file or directory that *manager* is attempting to move.

dstPath

The path or a file or directory to which *manager* is attempting to move.

Return Value
YES if the operation should proceed, otherwise NO.

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

See Also
- moveItemAtPath:toPath:error: (page 654)
- fileManager:shouldMoveItemAtPath:toPath: (page 662)

Declared In
NSFileManager.h

fileManager:shouldProceedAfterError:removingItemAtPath:

An NSFileManager object sends this message if an error occurs during an attempt to delete a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager
    shouldProceedAfterError:(NSError *)error
    removingItemAtPath:(NSString *)path
```

Parameters

fileManager
The NSFileManager object that sent this message.

error
The error that occurred during the attempt to copy.

path
The path or a file or directory that *manager* is attempting to delete.

Return Value
YES if the operation should proceed, otherwise NO.

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

See Also
- removeItemAtPath:error: (page 657)
- fileManager:shouldRemoveItemAtPath: (page 666)

Declared In
NSFileManager.h

fileManager:shouldRemoveItemAtPath:

An NSFileManager object sends this message immediately before attempting to delete an item at a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager
    shouldRemoveItemAtPath:(NSString *)path
```

Parameters

fileManager

The NSFileManager object that sent this message.

path

The path or a file or directory that *manager* is about to attempt to delete.

Return Value

YES if the operation should proceed, otherwise NO.

Discussion

You can implement this method in your delegate to monitor file operations.

Availability

Available in Mac OS X v10.5 and later.

See Also

- removeItemAtPath:error: (page 657)
- fileManager:shouldProceedAfterError:removingItemAtPath: (page 666)

Declared In

NSFileManager.h

fileManager:willProcessPath:

An NSFileManager object sends this message to a handler immediately before attempting to move, copy, rename, or delete, or before attempting to link to a given path.

```
- (void)fileManager:(NSFileManager *)manager  willProcessPath:(NSString *)path
```

Parameters

manager

The NSFileManager object that sent this message.

path

The path or a file or directory that *manager* is about to attempt to move, copy, rename, delete, or link to.

Discussion

You can implement this method in your handler to monitor file operations.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSFileManager.h

Constants

File Attribute Keys

These keys access file attribute values contained in `NSDictionary` objects used by `changeFileAttributes:atPath:` (page 632), `fileAttributesAtPath:traverseLink:` (page 645), `createDirectoryAtPath:attributes:` (page 637), and `createFileAtPath:contents:attributes:` (page 639).

```
NSString *NSFileType;
NSString *NSFileTypeDirectory;
NSString *NSFileTypeRegular;
NSString *NSFileTypeSymbolicLink;
NSString *NSFileTypeSocket;
NSString *NSFileTypeCharacterSpecial;
NSString *NSFileTypeBlockSpecial;
NSString *NSFileTypeUnknown;
NSString *NSFileSize;
NSString *NSFileModificationDate;
NSString *NSFileReferenceCount;
NSString *NSFileDeviceIdentifier;
NSString *NSFileOwnerAccountName;
NSString *NSFileGroupOwnerAccountName;
NSString *NSFilePosixPermissions;
NSString *NSFileSystemNumber;
NSString *NSFileSystemFileNumber;
NSString *NSFileExtensionHidden;
NSString *NSFileHFSCreatorCode;
NSString *NSFileHFSTypeCode;
NSString *NSFileImmutable;
NSString *NSFileAppendOnly;
NSString *NSFileCreationDate;
NSString *NSFileOwnerAccountID;
NSString *NSFileGroupOwnerAccountID;
NSString *NSFileBusy;
```

Constants

`NSFileAppendOnly`

The key in a file attribute dictionary whose value indicates whether the file is read-only.

The corresponding value is an `NSNumber` object containing a Boolean value.

Available in Mac OS X v10.2 and later.

Declared in `NSFileManager.h`.

`NSFileBusy`

The key in a file attribute dictionary whose value indicates whether the file is busy.

The corresponding value is an `NSNumber` object containing a Boolean value.

Available in Mac OS X v10.4 and later.

Declared in `NSFileManager.h`.

`NSFileCreationDate`
: The key in a file attribute dictionary whose value indicates the file's creation date.

  The corresponding value is an `NSDate` object.

  Available in Mac OS X v10.2 and later.

  Declared in `NSFileManager.h`.

`NSFileOwnerAccountName`
: The key in a file attribute dictionary whose value indicates the name of the file's owner.

  The corresponding value is an `NSString` object.

  Available in Mac OS X v10.0 and later.

  Declared in `NSFileManager.h`.

`NSFileGroupOwnerAccountName`
: The key in a file attribute dictionary whose value indicates the group name of the file's owner.

  The corresponding value is an `NSString` object.

  Available in Mac OS X v10.0 and later.

  Declared in `NSFileManager.h`.

`NSFileDeviceIdentifier`
: The key in a file attribute dictionary whose value indicates the identifier for the device on which the file resides.

  The corresponding value is an `NSNumber` object containing an `unsigned long`.

  Available in Mac OS X v10.0 and later.

  Declared in `NSFileManager.h`.

`NSFileExtensionHidden`
: The key in a file attribute dictionary whose value indicates whether the file's extension is hidden.

  The corresponding value is an `NSNumber` object containing a Boolean value.

  Available in Mac OS X v10.1 and later.

  Declared in `NSFileManager.h`.

`NSFileGroupOwnerAccountID`
: The key in a file attribute dictionary whose value indicates the file's group ID.

  The corresponding value is an `NSNumber` object containing an `unsigned long`.

  Available in Mac OS X v10.2 and later.

  Declared in `NSFileManager.h`.

`NSFileHFSCreatorCode`
: The key in a file attribute dictionary whose value indicates the file's HFS creator code.

  The corresponding value is an `NSNumber` object containing an `unsigned long`. See HFS File Types for possible values.

  Available in Mac OS X v10.1 and later.

  Declared in `NSFileManager.h`.

`NSFileHFSTypeCode`
: The key in a file attribute dictionary whose value indicates the file's HFS type code.

  The corresponding value is an `NSNumber` object containing an `unsigned long`. See HFS File Types for possible values.

  Available in Mac OS X v10.1 and later.

  Declared in `NSFileManager.h`.

`NSFileImmutable`
The key in a file attribute dictionary whose value indicates whether the file is mutable.
The corresponding value is an `NSNumber` object containing a Boolean value.
Available in Mac OS X v10.2 and later.
Declared in `NSFileManager.h`.

`NSFileModificationDate`
The key in a file attribute dictionary whose value indicates the file's last modified date.
The corresponding value is an `NSDate` object.
Available in Mac OS X v10.0 and later.
Declared in `NSFileManager.h`.

`NSFileOwnerAccountID`
The key in a file attribute dictionary whose value indicates the file's owner's account ID.
The corresponding value is an `NSNumber` object containing an `unsigned long`.
Available in Mac OS X v10.2 and later.
Declared in `NSFileManager.h`.

`NSFilePosixPermissions`
The key in a file attribute dictionary whose value indicates the file's Posix permissions.
The corresponding value is an `NSNumber` object containing an `unsigned long`.
Available in Mac OS X v10.0 and later.
Declared in `NSFileManager.h`.

`NSFileReferenceCount`
The key in a file attribute dictionary whose value indicates the file's reference count.
The corresponding value is an `NSNumber` object containing an `unsigned long`.
The number specifies the number of hard links to a file.
Available in Mac OS X v10.0 and later.
Declared in `NSFileManager.h`.

`NSFileSize`
The key in a file attribute dictionary whose value indicates the file's size in bytes.
The corresponding value is an `NSNumber` object containing an `unsigned long long`.

Important: If the file has a resource fork, the returned value does *not* include the size of the resource fork.

Available in Mac OS X v10.0 and later.
Declared in `NSFileManager.h`.

`NSFileSystemFileNumber`
The key in a file attribute dictionary whose value indicates the file's filesystem file number.
The corresponding value is an `NSNumber` object containing an `unsigned long`. The value corresponds to the value of `st_ino`, as returned by `stat`(2).
Available in Mac OS X v10.0 and later.
Declared in `NSFileManager.h`.

`NSFileType`
The key in a file attribute dictionary whose value indicates the file's type.

The corresponding value is an `NSString` object (see below for possible values).

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

Discussion
`NSFileDeviceIdentifier` is used to access the identifier of a remote device.

Declared In
`NSFileManager.h`

File Type Attribute Keys

These strings the possible values for the `NSFileType` attribute key contained in the `NSDictionary` object returned from `NSFileManager`'s `fileAttributesAtPath:traverseLink:` (page 645).

```
extern NSString *NSFileTypeDirectory;
extern NSString *NSFileTypeRegular;
extern NSString *NSFileTypeSymbolicLink;
extern NSString *NSFileTypeSocket;
extern NSString *NSFileTypeCharacterSpecial;
extern NSString *NSFileTypeBlockSpecial;
extern NSString *NSFileTypeUnknown;
```

Constants
`NSFileTypeDirectory`
Directory

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileTypeRegular`
Regular file

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileTypeSymbolicLink`
Symbolic link

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileTypeSocket`
Socket

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileTypeCharacterSpecial`
Character special file

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileTypeBlockSpecial`
Block special file

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileTypeUnknown`
Unknown

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

Declared In
`NSFileManager.h`

File-System Attribute Keys

Keys to access the file attribute values contained in the `NSDictionary` object returned from `NSFileManager`'s `fileSystemAttributesAtPath:` (page 648) method.

```
extern NSString *NSFileSystemSize;
extern NSString *NSFileSystemFreeSize;
extern NSString *NSFileSystemNodes;
extern NSString *NSFileSystemFreeNodes;
extern NSString *NSFileSystemNumber;
```

Constants
`NSFileSystemSize`
The key in a file system attribute dictionary whose value indicates the size of the file system.

The corresponding value is an `NSNumber` object that specifies the size of the file system in bytes. The value is determined by `statfs()`.

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileSystemFreeSize`
The key in a file system attribute dictionary whose value indicates the amount of free space on the file system.

The corresponding value is an `NSNumber` object that specifies the amount of free space on the file system in bytes. The value is determined by `statfs()`.

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileSystemNodes`
The key in a file system attribute dictionary whose value indicates the number of nodes in the file system.

The corresponding value is an `NSNumber` object that specifies the number of nodes in the file system.

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileSystemFreeNodes`

The key in a file system attribute dictionary dictionary whose value indicates the number of free nodes in the file system.

The corresponding value is an `NSNumber` object that specifies the number of free nodes in the file system.

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

`NSFileSystemNumber`

The key in a file system attribute dictionary dictionary whose value indicates the filesystem number of the file system.

The corresponding value is an `NSNumber` object that specifies the filesystem number of the file system. The value corresponds to the value of `st_dev`, as returned by `stat`(2).

Available in Mac OS X v10.0 and later.

Declared in `NSFileManager.h`.

Declared In
`NSFileManager.h`

Resource Fork Support

Specifies the version of the Foundation framework in which `NSFileManager` first supported resource forks.

```
#define NSFoundationVersionWithFileManagerResourceForkSupport 412
```

Constants
`NSFoundationVersionWithFileManagerResourceForkSupport`

The version of the Foundation framework in which `NSFileManager` first supported resource forks.

Available in Mac OS X v10.1 and later.

Declared in `NSFileManager.h`.

Declared In
`NSFileManager.h`

NSFormatter Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSFormatter.h
Companion guide	Data Formatting Programming Guide for Cocoa
Related sample code	bMoviePalette bMoviePaletteCocoa QTMetadataEditor QTSSConnectionMonitor QTSSInspector

Overview

`NSFormatter` is an abstract class that declares an interface for objects that create, interpret, and validate the textual representation of cell contents. The Foundation framework provides two concrete subclasses of `NSFormatter` to generate these objects: `NSNumberFormatter` and `NSDateFormatter`.

Subclassing Notes

`NSFormatter` is intended for subclassing. A custom formatter can restrict the input and enhance the display of data in novel ways. For example, you could have a custom formatter that ensures that serial numbers entered by a user conform to predefined formats. Before you decide to create a custom formatter, make sure that you cannot configure the public subclasses `NSDateFormatter` and `NSNumberFormatter` to satisfy your requirements.

For instructions on how to create your own custom formatter, see Creating a Custom Formatter.

Tasks

Textual Representation of Cell Content

- `stringForObjectValue:` (page 680)
  The default implementation of this method raises an exception.
- `attributedStringForObjectValue:withDefaultAttributes:` (page 676)
  The default implementation returns `nil` to indicate that the formatter object does not provide an attributed string.
- `editingStringForObjectValue:` (page 677)
  The default implementation of this method invokes `stringForObjectValue:` (page 680).

Object Equivalent to Textual Representation

- `getObjectValue:forString:errorDescription:` (page 677)
  The default implementation of this method raises an exception.

Dynamic Cell Editing

- `isPartialStringValid:newEditingString:errorDescription:` (page 679)
  Returns a Boolean value that indicates whether a partial string is valid.
- `isPartialStringValid:proposedSelectedRange:originalString:originalSelectedRange:errorDescription:` (page 679)
  This method should be implemented in subclasses that want to validate user changes to a string in a field, where the user changes are not necessarily at the end of the string, and preserve the selection (or set a different one, such as selecting the erroneous part of the string the user has typed).

Instance Methods

attributedStringForObjectValue:withDefaultAttributes:

The default implementation returns `nil` to indicate that the formatter object does not provide an attributed string.

```
- (NSAttributedString *)attributedStringForObjectValue:(id)anObject
    withDefaultAttributes:(NSDictionary *)attributes
```

Parameters

anObject
  The object for which a textual representation is returned.

attributes
  The default attributes to use for the returned attributed string.

Return Value
An attributed string that represents *anObject*.

Discussion
When implementing a subclass, return an `NSAttributedString` object if the string for display should have some attributes. For instance, you might want negative values in a financial application to appear in red text. Invoke your implementation of `stringForObjectValue:` (page 680) to get the non-attributed string, then create an `NSAttributedString` object with it (see `initWithString:` (page 153)). Use the *attributes* default dictionary to reset the attributes of the string when a change in value warrants it (for example, a negative value becomes positive) For information on creating attributed strings, see *Attributed Strings Programming Guide*.

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

See Also
`- editingStringForObjectValue:` (page 677)

Declared In
`NSFormatter.h`

editingStringForObjectValue:

The default implementation of this method invokes `stringForObjectValue:` (page 680).

```
- (NSString *)editingStringForObjectValue:(id)anObject
```

Parameters
anObject
The object for which to return an editing string.

Return Value
An `NSString` object that is used for editing the textual representation of *anObject*.

Discussion
When implementing a subclass, override this method only when the string that users see and the string that they edit are different. In your implementation, return an `NSString` object that is used for editing, following the logic recommended for implementing `stringForObjectValue:` (page 680). As an example, you would implement this method if you want the dollar signs in displayed strings removed for editing.

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

See Also
`- attributedStringForObjectValue:withDefaultAttributes:` (page 676)

Declared In
`NSFormatter.h`

getObjectValue:forString:errorDescription:

The default implementation of this method raises an exception.

```
- (BOOL)getObjectValue:(id *)anObject forString:(NSString *)string
    errorDescription:(NSString **)error
```

Parameters

anObject

If conversion is successful, upon return contains the object created from *string*.

string

The string to parse.

error

If non-`nil`, if there is a error during the conversion, upon return contains an `NSString` object that describes the problem.

Return Value

`YES` if the conversion from string to cell content object was successful, otherwise `NO`.

Discussion

When implementing a subclass, return by reference the object *anObject* after creating it from *string*. Return `YES` if the conversion is successful. If you return `NO`, also return by indirection (in *error*) a localized user-presentable `NSString` object that explains the reason why the conversion failed; the delegate (if any) of the `NSControl` object managing the cell can then respond to the failure in `control:didFailToFormatString:errorDescription:`. However, if *error* is `nil`, the sender is not interested in the error description, and you should not attempt to assign one.

The following example (which is paired with the example given in `stringForObjectValue:` (page 680)) converts a string representation of a dollar amount that includes the dollar sign; it uses an `NSScanner` instance to convert this amount to a float after stripping out the initial dollar sign.

```
- (BOOL)getObjectValue:(id *)obj forString:(NSString *)string
errorDescription:(NSString  **)error
{
    float floatResult;
    NSScanner *scanner;
    BOOL returnValue = NO;

    scanner = [NSScanner scannerWithString: string];
    [scanner scanString: @"$" intoString: NULL];    //ignore  return value
    if ([scanner scanFloat:&floatResult] && ([scanner isAtEnd])) {
        returnValue = YES;
        if (obj)
            *obj = [NSNumber numberWithFloat:floatResult];
    } else {
        if (error)
            *error = NSLocalizedString(@"Couldn't convert  to float", @"Error
converting");
    }
    return returnValue;
}
```

Availability

Available in Mac OS X v10.0 and later.

See Also

`- stringForObjectValue:` (page 680)

Declared In

`NSFormatter.h`

isPartialStringValid:newEditingString:errorDescription:

Returns a Boolean value that indicates whether a partial string is valid.

```
- (BOOL)isPartialStringValid:(NSString *)partialString newEditingString:(NSString
    **)newString errorDescription:(NSString **)error
```

Parameters

partialString

The text currently in a cell.

newString

If *partialString* needs to be modified, upon return contains the replacement string.

error

If non-`nil`, if validation fails contains an `NSString` object that describes the problem.

Return Value

`YES` if *partialString* is an acceptable value, otherwise `NO`.

Discussion

This method is invoked each time the user presses a key while the cell has the keyboard focus—it lets you verify and edit the cell text as the user types it.

In a subclass implementation, evaluate *partialString* according to the context, edit the text if necessary, and return by reference any edited string in *newString*. Return `YES` if *partialString* is acceptable and `NO` if *partialString* is unacceptable. If you return `NO` and *newString* is `nil`, the cell displays *partialString* minus the last character typed. If you return `NO`, you can also return by indirection an `NSString` object (in *error*) that explains the reason why the validation failed; the delegate (if any) of the `NSControl` object managing the cell can then respond to the failure in `control:didFailToValidatePartialString:errorDescription:`. The selection range will always be set to the end of the text if replacement occurs.

This method is a compatibility method. If a subclass overrides this method and does not override `isPartialStringValid:proposedSelectedRange:originalString:originalSelectedRange:errorDescription:` (page 679), this method will be called as before (`isPartialStringValid:proposedSelectedRange:originalString:originalSelectedRange:errorDescription:` (page 679) just calls this one by default).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSFormatter.h`

isPartialStringValid:proposedSelectedRange:originalString:originalSelectedRange:errorDescription:

This method should be implemented in subclasses that want to validate user changes to a string in a field, where the user changes are not necessarily at the end of the string, and preserve the selection (or set a different one, such as selecting the erroneous part of the string the user has typed).

```
- (BOOL)isPartialStringValid:(NSString **)partialStringPtr
    proposedSelectedRange:(NSRangePointer)proposedSelRangePtr
    originalString:(NSString *)origString originalSelectedRange:(NSRange)origSelRange
     errorDescription:(NSString **)error
```

Parameters

partialStringPtr

The new string to validate.

proposedSelRangePtr

The selection range that will be used if the string is accepted or replaced.

origString

The original string, before the proposed change.

origSelRange

The selection range over which the change is to take place.

error

If non-`nil`, if validation fails contains an `NSString` object that describes the problem.

Return Value

`YES` if *partialStringPtr* is acceptable, otherwise `NO`.

Discussion

In a subclass implementation, evaluate *partialString* according to the context. Return `YES` if *partialStringPtr* is acceptable and `NO` if *partialStringPtr* is unacceptable. Assign a new string to *partialStringPtr* and a new range to *proposedSelRangePtr* and return `NO` if you want to replace the string and change the selection range. If you return `NO`, you can also return by indirection an `NSString` object (in *error*) that explains the reason why the validation failed; the delegate (if any) of the `NSControl` object managing the cell can then respond to the failure in `control:didFailToValidatePartialString:errorDescription:`.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- isPartialStringValid:newEditingString:errorDescription:` (page 679)

Declared In

`NSFormatter.h`

stringForObjectValue:

The default implementation of this method raises an exception.

```
- (NSString *)stringForObjectValue:(id)anObject
```

Parameters

anObject

The object for which a textual representation is returned.

Return Value

An `NSString` object that textually represents *object* for display. Returns `nil` if *object* is not of the correct class.

Discussion

When implementing a subclass, return the `NSString` object that textually represents the cell's object for display and—if `editingStringForObjectValue:` (page 677) is unimplemented—for editing. First test the passed-in object to see if it's of the correct class. If it isn't, return `nil`; but if it is of the right class, return a properly formatted and, if necessary, localized string. (See the specification of the `NSString` class for formatting and localizing details.)

The following implementation (which is paired with the `getObjectValue:forString:errorDescription:` (page 677) example above) prefixes a two-digit float representation with a dollar sign:

```
- (NSString *)stringForObjectValue:(id)anObject
{
    if (![anObject isKindOfClass:[NSNumber class]]) {
        return nil;
    }
    return [NSString stringWithFormat:@"$%.2f", [anObject  floatValue]];
}
```

Availability

Available in Mac OS X v10.0 and later.

See Also

- `attributedStringForObjectValue:withDefaultAttributes:` (page 676)
- `editingStringForObjectValue:` (page 677)
- `getObjectValue:forString:errorDescription:` (page 677)

Declared In

`NSFormatter.h`

NSGarbageCollector Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSGarbageCollector.h
Availability	Available in Mac OS X v10.5 and later.
Companion guide	Garbage Collection Programming Guide

Overview

`NSGarbageCollector` provides a convenient interface to the garbage collection system.

Cocoa's garbage collector is a conservative generational garbage collector. It uses "write-barriers" to detect cross generational stores of pointers so that "young" objects can be collected quickly.

You enable garbage collection (GC) by using the `-fobjc-gc compiler` option. This switch causes the generation of the write-barrier assignment primitives. You must use this option on your main application file *and all others used by the application*, including frameworks and bundles. Bundles are ignored if they are not GC-capable.

The collector determines what is garbage by recursively examining all nodes starting with globals, possible nodes referenced from the thread stacks, and all nodes marked as having "external" references. Nodes not reached by this search are deemed garbage. Weak references to garbage nodes are then cleared.

Garbage nodes that are objects are sent (in an arbitrary order) a `finalize` (page 1176) message, and after all `finalize` messages have been sent their memory is recovered. It is a runtime error (referred to as "resurrection") to store a object being finalized into one that is not. For more details, see Implementing a finalize Method in *Garbage Collection Programming Guide*.

You can request collection from any thread (see `collectIfNeeded` (page 685) and `collectExhaustively` (page 685)).

Tasks

Shared Instance

+ `defaultCollector` (page 685)
Returns the default garbage collector.

Collection State

- `disable` (page 686)
Temporarily disables collections.
- `enable` (page 687)
Enables collection after collection has been disabled.
- `isEnabled` (page 688)
Returns a Boolean value that indicates whether garbage collection is currently enabled for the current process.
- `isCollecting` (page 688)
Returns a Boolean value that indicates whether a collection is currently in progress.

Triggering Collection

- `collectExhaustively` (page 685)
Tells the receiver to collect iteratively.
- `collectIfNeeded` (page 685)
Tells the receiver to collect if memory consumption thresholds have been exceeded.

Manipulating External References

- `disableCollectorForPointer:` (page 686)
Specifies that a given pointer will not be collected.
- `enableCollectorForPointer:` (page 687)
Specifies that a given pointer may be collected.

Accessing an Unscanned Memory Zone

- `zone` (page 688)
Returns a zone of unscanned memory.

Class Methods

defaultCollector

Returns the default garbage collector.

```
+ (id)defaultCollector
```

Return Value
The default garbage collector for the current process. Returns `nil` if the current process is not running with garbage collection.

Discussion
There is at most one garbage collector for Cocoa within a single process.

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

Declared In
`NSGarbageCollector.h`

Instance Methods

collectExhaustively

Tells the receiver to collect iteratively.

```
- (void)collectExhaustively
```

Discussion
You use this method to indicate to the collector that it should perform an exhaustive collection. Collection is subject to interruption on user input.

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

Declared In
`NSGarbageCollector.h`

collectIfNeeded

Tells the receiver to collect if memory consumption thresholds have been exceeded.

```
- (void)collectIfNeeded
```

Discussion
You use this method to indicate to the collector that there is an opportunity to perform a collection. Collection is subject to interruption on user input.

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

Declared In
NSGarbageCollector.h

disable

Temporarily disables collections.

```
- (void)disable
```

Discussion
Invocations of this method can be nested. To reenable collection, you must send the collector an `enable` (page 687) message once for each invocation of this method.

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

See Also
- `enable` (page 687)

Declared In
NSGarbageCollector.h

disableCollectorForPointer:

Specifies that a given pointer will not be collected.

```
- (void)disableCollectorForPointer:(void *)ptr
```

Parameters

ptr

A pointer to the memory that should not be collected.

Discussion
You use this method to ensure that memory at a given address will not be collected. You can use this, for example, to create new root objects:

```
NSMutableDictionary *globalDictionary;
globalDictionary = [NSMutableDictionary dictionary];
[[NSGarbageCollector defaultCollector]
    disableCollectorForPointer:globalDictionary];
```

The new dictionary will not be collectable and will persist for the lifetime of the application unless it is subsequently passed as the argument to `enableCollectorForPointer:` (page 687). For more about root objects and scanned memory, see *Garbage Collection Programming Guide*.

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

See Also
- `enableCollectorForPointer:` (page 687)

Declared In
NSGarbageCollector.h

enable

Enables collection after collection has been disabled.

```
- (void)enable
```

Discussion
This method balances a single invocation of `disable` (page 686). To reenable collection, this method must be invoked as many times as was `disable` (page 686).

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

See Also
- `disable` (page 686)
- `isEnabled` (page 688)

Declared In
NSGarbageCollector.h

enableCollectorForPointer:

Specifies that a given pointer may be collected.

```
- (void)enableCollectorForPointer:(void *)ptr
```

Parameters
ptr
A pointer to the memory that may be collected.

Discussion
You use this method to make memory that was previously marked as uncollectable. For example, given the address of the global dictionary created in `disableCollectorForPointer:` (page 686), you could make the dictionary collectable as follows:

```
[[NSGarbageCollector defaultCollector]
    enableCollectorForPointer:globalDictionary];
```

For more about root objects and scanned memory, see *Garbage Collection Programming Guide*.

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

See Also
- `disableCollectorForPointer:` (page 686)

Declared In
NSGarbageCollector.h

isCollecting

Returns a Boolean value that indicates whether a collection is currently in progress.

```
- (BOOL)isCollecting
```

Return Value
`YES` if a collection is currently in progress, otherwise `NO`.

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

Declared In
`NSGarbageCollector.h`

isEnabled

Returns a Boolean value that indicates whether garbage collection is currently enabled for the current process.

```
- (BOOL)isEnabled
```

Return Value
`YES` if garbage collection is enabled for the current process, otherwise `NO`.

Discussion
This method returns `NO` if garbage collection is on, but has been temporarily suspended (using `disable` (page 686)).

To check whether the current process is using garbage collection check the result of `[NSGarbageCollector defaultCollector]`. If `defaultCollector` (page 685) is `nil`, then garbage collection is permanently off. If `defaultCollector` (page 685) is not `nil`, then the current process is using garbage collection—you can then use `isEnabled` to determine whether or not the collector is actually allowed to run right now.

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

See Also
- `disable` (page 686)
- `enable` (page 687)

Declared In
`NSGarbageCollector.h`

zone

Returns a zone of unscanned memory.

```
- (NSZone *)zone
```

Return Value
A memory zone of memory that is not scanned.

Discussion
The collector provides a `NSZoneMalloc` (page 2259)-style allocation interface, primarily for compatibility with existing code that maintains zone affinity. Such memory is unscanned and you must free it using `NSZoneFree` (page 2258). This is exactly equivalent to calling `NSAllocateCollectable` (page 2158) with the option `NSCollectorDisabledOption` (page ?).

You should typically allocate garbage-collected memory using `NSAllocateCollectable` (page 2158).

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

Declared In
`NSGarbageCollector.h`

NSGetCommand Class Reference

Inherits from	NSScriptCommand : NSObject
Conforms to	NSCoding (NSScriptCommand) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptStandardSuiteCommands.h
Companion guide	Cocoa Scripting Guide

Overview

An instance of `NSGetCommand` gets the specified value or object from the specified scriptable object: for example, the words from a paragraph or the name of a document.

When an instance of `NSGetCommand` is executed, it evaluates the specified receivers, gathers the specified data, if any, and packages it in a return Apple event.

`NSGetCommand` is part of Cocoa's built-in scripting support. It works automatically to support the `get` command through key-value coding. Most applications don't need to subclass `NSGetCommand` or call its methods.

For information on working with `get` commands, see Getting and Setting Properties and Elements in *Cocoa Scripting Guide*.

NSHashTable Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSFastEnumeration NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	Foundation/NSHashTable.h
Companion guide	Garbage Collection Programming Guide

Overview

`NSHashTable` is modeled after `NSSet` but provides different options, in particular to support weak relationships in a garbage-collected environment.

Tasks

Initialization

`- initWithOptions:capacity:` (page 697)
Returns a hash table initialized with the given attributes.

`- initWithPointerFunctions:capacity:` (page 698)
Returns a hash table initialized with the given functions and capacity.

Convenience Constructors

`+ hashTableWithOptions:` (page 695)
Returns a hash table with given pointer functions options.

`+ hashTableWithWeakObjects` (page 695)
Returns a new hash table for storing weak references to its contents.

Accessing Content

- `allObjects` (page 696)
  Returns an array that contains the receiver's members.
- `anyObject` (page 696)
  Returns one of the objects in the receiver.
- `containsObject:` (page 697)
  Returns a Boolean value that indicates whether the receiver contains a given object.
- `count` (page 697)
  Returns the number of elements in the receiver.
- `member:` (page 700)
  Determines whether a given object is an element in the receiver.
- `objectEnumerator` (page 700)
  Returns an enumerator object that lets you access each object in the receiver.
- `setRepresentation` (page 702)
  Returns a set that contains the receiver's members.

Manipulating Membership

- `addObject:` (page 696)
  Adds a given object to the receiver.
- `removeAllObjects` (page 701)
  Removes all objects from the receiver.
- `removeObject:` (page 701)
  Removes a given object from the receiver.

Comparing Hash Tables

- `intersectsHashTable:` (page 698)
  Returns a Boolean value that indicates whether a given hash table intersects with the receiver.
- `isEqualToHashTable:` (page 699)
  Returns a Boolean value that indicates whether a given hash table is equal to the receiver.
- `isSubsetOfHashTable:` (page 699)
  Returns a Boolean value that indicates whether every element in the receiver is also present in another given hash table.

Set Functions

- `intersectHashTable:` (page 698)
  Returns a Boolean value that indicates whether at least one element in the receiver is also present in another given hash table.
- `minusHashTable:` (page 700)
  Removes from the receiver each element contained in another given hash table that is present in the receiver.

- unionHashTable: (page 702)
  Adds to the receiver each element contained in another given hash table that is not already a member.

Accessing Pointer Functions

- pointerFunctions (page 701)
  Returns the pointer functions for the receiver.

Class Methods

hashTableWithOptions:

Returns a hash table with given pointer functions options.

```
+ (id)hashTableWithOptions:(NSPointerFunctionsOptions)options
```

Parameters

options
A bit field that specifies the options for the elements in the hash table. For possible values, see "Hash Table Options" (page 703).

Return Value
A hash table with given pointer functions options.

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

Declared In
NSHashTable.h

hashTableWithWeakObjects

Returns a new hash table for storing weak references to its contents.

```
+ (id)hashTableWithWeakObjects
```

Return Value
A new has table that uses the options NSHashTableZeroingWeakMemory (page 703) and NSPointerFunctionsObjectPersonality (page 1245) and has an initial capacity of 0.

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

Declared In
NSHashTable.h

Instance Methods

addObject:

Adds a given object to the receiver.

```
- (void)addObject:(id)object
```

Parameters

object

The object to add to the receiver.

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

Declared In
NSHashTable.h

allObjects

Returns an array that contains the receiver's members.

```
- (NSArray *)allObjects
```

Return Value
An array that contains the receiver's members.

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

Declared In
NSHashTable.h

anyObject

Returns one of the objects in the receiver.

```
- (id)anyObject
```

Return Value
One of the objects in the receiver, or `nil` if the receiver contains no objects.

Discussion
The object returned is chosen at the receiver's convenience—the selection is not guaranteed to be random.

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

Declared In
NSHashTable.h

containsObject:

Returns a Boolean value that indicates whether the receiver contains a given object.

```
- (BOOL)containsObject:(id)anObject
```

Parameters

anObject

The object to test for membership in the receiver.

Return Value

`YES` if the receiver contains *anObject*, otherwise `NO`.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSHashTable.h`

count

Returns the number of elements in the receiver.

```
- (NSUInteger)count
```

Return Value

The number of elements in the receiver.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSHashTable.h`

initWithOptions:capacity:

Returns a hash table initialized with the given attributes.

```
- (id)initWithOptions:(NSPointerFunctionsOptions)options
    capacity:(NSUInteger)capacity
```

Parameters

options

A bit field that specifies the options for the elements in the hash table. For possible values, see "Hash Table Options" (page 703).

capacity

The initial number of elements the receiver can hold.

Return Value

A hash table initialized with options specified by *options* and initial capacity of *capacity*.

Availability

Available in Mac OS X v10.5 and later.

Declared In
NSHashTable.h

initWithPointerFunctions:capacity:

Returns a hash table initialized with the given functions and capacity.

```
- (id)initWithPointerFunctions:(NSPointerFunctions *)functions
    capacity:(NSUInteger)initialCapacity
```

Parameters

functions
The pointer functions for the new hash table.

initialCapacity
The initial capacity of the hash table.

Return Value
A hash table initialized with the given functions and capacity.

Discussion
Hash tables allocate additional memory as needed, so *initialCapacity* simply establishes the object's initial capacity.

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

Declared In
NSHashTable.h

intersectHashTable:

Returns a Boolean value that indicates whether at least one element in the receiver is also present in another given hash table.

```
- (void)intersectHashTable:(NSHashTable *)other
```

Parameters

other
The hash table with which to compare the receiver.

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

Declared In
NSHashTable.h

intersectsHashTable:

Returns a Boolean value that indicates whether a given hash table intersects with the receiver.

```
- (BOOL)intersectsHashTable:(NSHashTable *)other
```

Parameters

other

The hash table with which to compare the receiver.

Return Value

YES if *other* intersects with the receiver, otherwise NO.

Availability

Available in Mac OS X v10.5 and later.

Declared In

NSHashTable.h

isEqualToHashTable:

Returns a Boolean value that indicates whether a given hash table is equal to the receiver.

```
- (BOOL)isEqualToHashTable:(NSHashTable *)other
```

Parameters

other

The hash table with which to compare the receiver.

Return Value

YES if the contents of *other* are equal to the contents of the receiver, otherwise NO.

Discussion

Two hash tables have equal contents if they each have the same number of members and if each member of one hash table is present in the other.

Availability

Available in Mac OS X v10.5 and later.

Declared In

NSHashTable.h

isSubsetOfHashTable:

Returns a Boolean value that indicates whether every element in the receiver is also present in another given hash table.

```
- (BOOL)isSubsetOfHashTable:(NSHashTable *)other
```

Parameters

other

The hash table with which to compare the receiver.

Return Value

YES if every element in the receiver is also present in *other*, otherwise NO.

Availability

Available in Mac OS X v10.5 and later.

Declared In
NSHashTable.h

member:

Determines whether a given object is an element in the receiver.

```
- (id)member:(id)object
```

Parameters

object
The object to test for membership in the receiver.

Return Value
If *object* is a member of the receiver, returns *object*, otherwise returns nil.

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

Declared In
NSHashTable.h

minusHashTable:

Removes from the receiver each element contained in another given hash table that is present in the receiver.

```
- (void)minusHashTable:(NSHashTable *)other
```

Parameters

other
The hash table of elements to remove from the receiver.

Discussion
If any element of *other* isn't present in the receiver, this method has no effect on either the receiver or *other*.

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

Declared In
NSHashTable.h

objectEnumerator

Returns an enumerator object that lets you access each object in the receiver.

```
- (NSEnumerator *)objectEnumerator
```

Return Value
An enumerator object that lets you access each object in the receiver.

Discussion
The following code fragment illustrates how you can use this method.

```
NSEnumerator *enumerator = [myHashTable objectEnumerator];
id value;

while ((value = [enumerator nextObject])) {
    /* code that acts on the hash table's values */
}
```

Note that `NSHashTable` also supports the `NSFastEnumeration` protocol.

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

Declared In
`NSHashTable.h`

pointerFunctions

Returns the pointer functions for the receiver.

```
- (NSPointerFunctions *)pointerFunctions
```

Return Value
The pointer functions for the receiver.

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

Declared In
`NSHashTable.h`

removeAllObjects

Removes all objects from the receiver.

```
- (void)removeAllObjects
```

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

Declared In
`NSHashTable.h`

removeObject:

Removes a given object from the receiver.

```
- (void)removeObject:(id)object
```

Parameters
object
The object to remove from the receiver.

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

Declared In
`NSHashTable.h`

setRepresentation

Returns a set that contains the receiver's members.

```
- (NSSet *)setRepresentation
```

Return Value
A set that contains the receiver's members.

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

Declared In
`NSHashTable.h`

unionHashTable:

Adds to the receiver each element contained in another given hash table that is not already a member.

```
- (void)unionHashTable:(NSHashTable *)other
```

Parameters
other
The hash table of elements to add to the receiver.

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

Declared In
`NSHashTable.h`

Constants

NSHashTableOptions

Type to specify a bit-field used to define the behavior of elements in an `NSHashTable` object.

```
typedef NSUInteger NSHashTableOptions;
```

Discussion
For possible values, see "Hash Table Options" (page 703).

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

Declared In
NSHashTable.h

Hash Table Options

Components in a bit-field to specify the behavior of elements in an `NSHashTable` object.

```
enum {
    NSHashTableStrongMemory             = 0,
    NSHashTableZeroingWeakMemory        = NSPointerFunctionsZeroingWeakMemory,
    NSHashTableCopyIn                   = NSPointerFunctionsCopyIn,
   NSHashTableObjectPointerPersonality = NSPointerFunctionsObjectPointerPersonality,
};
```

Constants

`NSHashTableStrongMemory`
Equal to `NSPointerFunctionsStrongMemory` (page 1244).
Available in Mac OS X v10.5 and later.
Declared in `NSHashTable.h`.

`NSHashTableZeroingWeakMemory`
Equal to `NSPointerFunctionsZeroingWeakMemory` (page 1244).
Available in Mac OS X v10.5 and later.
Declared in `NSHashTable.h`.

`NSHashTableCopyIn`
Equal to `NSPointerFunctionsCopyIn` (page 1246).
Available in Mac OS X v10.5 and later.
Declared in `NSHashTable.h`.

`NSHashTableObjectPointerPersonality`
Equal to `NSPointerFunctionsObjectPointerPersonality` (page 1245).
Available in Mac OS X v10.5 and later.
Declared in `NSHashTable.h`.

Declared In
NSHashTable.h

NSHost Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSHost.h
Companion guide	Interacting with the Operating System
Related sample code	Core Data HTML Store NameAndAddress

Overview

The `NSHost` class provides methods to access the network name and address information for a host. Instances of the `NSHost` class represent individual **hosts** on a network. Use `NSHost` objects to get the current host's name and address and to look up other hosts by name or by address.

To create an `NSHost` object, use the `currentHost` (page 707), `hostWithAddress:` (page 707), or `hostWithName:` (page 708) class methods (don't use `alloc` and `init`). These methods use available network administration services (such as NetInfo or the Domain Name Service) to discover all names and addresses for the host requested. They don't attempt to contact the host itself, however. This approach avoids untimely delays due to a host being unavailable, but it may result in incomplete information about the host.

An `NSHost` object contains all of the network addresses and names discovered for a given host by the network administration services. Each `NSHost` object typically contains one unique address, but it may have more than one name. If an `NSHost` object has more than one name, the additional names are variations on the same name, typically the basic host name plus the fully qualified domain name. For example, with a host name "`sales`" in the domain "`anycorp.com`", an `NSHost` object can hold both the names "`sales`" and "`sales.anycorp.com`".

The `NSHost` class maintains a cache of the `NSHost` objects it creates so that requests for an existing `NSHost` object return that object instead of creating a new one. Use the `setHostCacheEnabled:` (page 709) method to turn the cache off, forcing lookup of hosts as they're requested. You can also use the `flushHostCache` (page 707) method to clear the cache of its entries so that subsequent requests look up the host information and create new instances.

Tasks

Creating Hosts

`+ currentHost` (page 707)
Returns an `NSHost` object representing the host the process is running on.

`+ hostWithAddress:` (page 707)
Returns the `NSHost` with the Internet address *address*.

`+ hostWithName:` (page 708)
Returns a host with a specific name.

Getting Host Information

`- address` (page 709)
Returns one of the network addresses of the receiver.

`- addresses` (page 710)
Returns all the network addresses of the receiver.

`- name` (page 711)
Returns one of the hostnames of the receiver.

`- names` (page 711)
Returns all the hostnames of the receiver.

Comparing Hosts

`- isEqualToHost:` (page 710)
Indicates whether the receiver represents the same host as another `NSHost` object.

Managing the Host Cache

`+ isHostCacheEnabled` (page 709)
Indicates whether caching is turned on or off.

`+ setHostCacheEnabled:` (page 709)
Specifies whether the receiver is to cache instances as it creates them to avoid creating duplicate instances.

`+ flushHostCache` (page 707)
Releases the cache of existing `NSHost` objects so subsequent requests for `NSHost` objects create new ones.

Class Methods

currentHost

Returns an `NSHost` object representing the host the process is running on.

```
+ (NSHost *)currentHost
```

Return Value
`NSHost` object for the process's host.

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

See Also
`+ hostWithAddress:` (page 707)
`+ hostWithName:` (page 708)

Related Sample Code
Core Data HTML Store
NameAndAddress

Declared In
`NSHost.h`

flushHostCache

Releases the cache of existing `NSHost` objects so subsequent requests for `NSHost` objects create new ones.

```
+ (void)flushHostCache
```

Discussion
`NSHost` objects that were retained before this method was invoked remain valid.

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

See Also
`+ isHostCacheEnabled` (page 709)
`+ setHostCacheEnabled:` (page 709)

Declared In
`NSHost.h`

hostWithAddress:

Returns the `NSHost` with the Internet address *address*.

```
+ (NSHost *)hostWithAddress:(NSString *)address
```

Parameters

address

Network address to look up. For example, `@"127.0.0.1"` or `@"fe80::1"`.

Return Value

Host for *address*.

Discussion

If caching is turned on and the cache already contains an `NSHost` object with *address*, returns that object. Otherwise, this method creates an instance and returns it.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ hostWithName:` (page 708)

`+ setHostCacheEnabled:` (page 709)

Related Sample Code

NameAndAddress

Declared In

`NSHost.h`

hostWithName:

Returns a host with a specific name.

```
+ (NSHost *)hostWithName:(NSString *)hostname
```

Parameters

hostname

Name of the host to look up. Can be either a simple hostname, such as `@"sales"`, or a fully qualified domain name, such as `@"sales.anycorp.com"`.

Return Value

Host named *hostname*.

Discussion

If caching is turned on and the cache already contains an `NSHost` object with *name*, returns that object. Otherwise, this method creates a new instance and returns it.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ hostWithAddress:` (page 707)

`+ setHostCacheEnabled:` (page 709)

Related Sample Code

NameAndAddress

Declared In

`NSHost.h`

isHostCacheEnabled

Indicates whether caching is turned on or off.

```
+ (BOOL)isHostCacheEnabled
```

Return Value
`YES` when caching is turned on; `NO` otherwise.

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

See Also
`+ setHostCacheEnabled:` (page 709)
`+ flushHostCache` (page 707)

Declared In
`NSHost.h`

setHostCacheEnabled:

Specifies whether the receiver is to cache instances as it creates them to avoid creating duplicate instances.

```
+ (void)setHostCacheEnabled:(BOOL)cacheOn
```

Parameters
cacheOn
`YES` to turn on caching. `NO` to turn of caching.

Discussion
Caching is turned on by default.

This method doesn't flush the cache. If you turn caching off and then back on, new requests for hosts use what was in the cache at the time caching was turned off. However, `NSHost` objects created while caching is turned off aren't entered into the cache.

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

See Also
`+ isHostCacheEnabled` (page 709)
`+ flushHostCache` (page 707)

Declared In
`NSHost.h`

Instance Methods

address

Returns one of the network addresses of the receiver.

```
- (NSString *)address
```

Return Value
One of the network address for the receiver. For example, `@"192.42.172.1"` or `@"fe80::1"`.

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

See Also
- `addresses` (page 710)
- `name` (page 711)

Related Sample Code
NameAndAddress

Declared In
`NSHost.h`

addresses

Returns all the network addresses of the receiver.

```
- (NSArray *)addresses
```

Return Value
All the network addresses of the receiver.

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

See Also
- `address` (page 709)
- `names` (page 711)

Declared In
`NSHost.h`

isEqualToHost:

Indicates whether the receiver represents the same host as another `NSHost` object.

```
- (BOOL)isEqualToHost:(NSHost *)host
```

Parameters
host
Host to compare the receiver to.

Return Value
`YES` when the receiver and *host* share at least one network address; `NO` otherwise.

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

See Also
- `addresses` (page 710)

Declared In
`NSHost.h`

name

Returns one of the hostnames of the receiver.

```
- (NSString *)name
```

Return Value
One of the hostnames of the receiver. Can be either a simple hostname, such as `@"sales"`, or a fully qualified domain name, such as `@"sales.anycorp.com"`.

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

See Also
- `address` (page 709)
- `names` (page 711)

Related Sample Code
Core Data HTML Store
NameAndAddress

Declared In
`NSHost.h`

names

Returns all the hostnames of the receiver.

```
- (NSArray *)names
```

Return Value
All the hostnames of the receiver.

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

See Also
- `addresses` (page 710)
- `name` (page 711)

Declared In
`NSHost.h`

NSHTTPCookie Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSHTTPCookie.h
Availability	Available in Mac OS X v10.2 with Safari 1.0 installed. Available in Mac OS X v10.2.7 and later.
Companion guide	URL Loading System

Overview

An NSHTTPCookie object represents an HTTP cookie. It's an immutable object initialized from a dictionary containing the cookie attributes.

Two versions of cookies are supported:

- Version 0: This version refers to "traditional" or "old-style" cookies, the original cookie format defined by Netscape. The majority of cookies encountered are in this format.
- Version 1: This version refers to cookies as defined in RFC 2965, HTTP State Management Mechanism.

Adopted Protocols

NSCopying
- `copyWithZone:` (page 2042)

Tasks

Create Cookie Instances

+ `cookiesWithResponseHeaderFields:forURL:` (page 715)
  Returns an array of NSHTTPCookie objects corresponding to the provided response header fields for the provided URL.

+ `cookieWithProperties:` (page 715)
  Creates and initializes an NSHTTPCookie object using the provided properties.

- `initWithProperties:` (page 718)
  Returns an initialized `NSHTTPCookie` object using the provided properties.

Convert Cookies to Request Headers

+ `requestHeaderFieldsWithCookies:` (page 716)
  Returns a dictionary of header fields corresponding to a provided array of cookies.

Getting Cookie Properties

- `comment` (page 716)
  Returns the receiver's comment string.

- `commentURL` (page 717)
  Returns the receiver's comment URL.

- `domain` (page 717)
  Returns the domain of the receiver's cookie.

- `expiresDate` (page 717)
  Returns the receiver's expiration date.

- `isSecure` (page 718)
  Returns whether his cookie should only be sent over secure channels.

- `isSessionOnly` (page 719)
  Returns whether the receiver should be discarded at the end of the session (regardless of expiration date).

- `name` (page 719)
  Returns the receiver's name.

- `path` (page 719)
  Returns the receiver's path.

- `portList` (page 720)
  Returns the receiver's port list.

- `properties` (page 720)
  Returns the receiver's cookie properties.

- `value` (page 720)
  Returns the receiver's value.

- version (page 721)
  Returns the receiver's version.

Class Methods

cookiesWithResponseHeaderFields:forURL:

Returns an array of NSHTTPCookie objects corresponding to the provided response header fields for the provided URL.

```
+ (NSArray *)cookiesWithResponseHeaderFields:(NSDictionary *)headerFields
    forURL:(NSURL *)theURL
```

Parameters

headerFields
  The header fields used to create the NSHTTPCookie objects.

theURL
  The URL associated with the created cookies.

Return Value
The array of newly created cookies.

Discussion
This method will ignore irrelevant header fields in *headerFields*, allowing dictionaries to contain additional data.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSHTTPCookie.h

cookieWithProperties:

Creates and initializes an NSHTTPCookie object using the provided properties.

```
+ (id)cookieWithProperties:(NSDictionary *)properties
```

Parameters

properties
  The properties for the new cookie object, expressed as key value pairs.

Return Value
The newly created cookie object. Returns nil if the provided properties are invalid.

Discussion
See "Constants" (page 721) for more information on the available header field constants and the constraints imposed on the values in the dictionary.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
`- initWithProperties:` (page 718)

Declared In
`NSHTTPCookie.h`

requestHeaderFieldsWithCookies:

Returns a dictionary of header fields corresponding to a provided array of cookies.

```
+ (NSDictionary *)requestHeaderFieldsWithCookies:(NSArray *)cookies
```

Parameters
cookies
The cookies from which the header fields are created.

Return Value
The dictionary of header fields created from the provided cookies. This dictionary can be used to add cookies to a request.

Discussion
See "Constants" (page 721) for details on the header field keys and values in the returned dictionary.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSHTTPCookie.h`

Instance Methods

comment

Returns the receiver's comment string.

```
- (NSString *)comment
```

Return Value
The receiver's comment string or `nil` if the cookie has no comment. This string is suitable for presentation to the user, explaining the contents and purpose of this cookie.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSHTTPCookie.h

commentURL

Returns the receiver's comment URL.

```
- (NSURL *)commentURL
```

Return Value
The receiver's comment URL or `nil` if the cookie has none. This value specifies a URL which is suitable for presentation to the user as a link for further information about this cookie.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSHTTPCookie.h

domain

Returns the domain of the receiver's cookie.

```
- (NSString *)domain
```

Return Value
The domain of the receiver's cookie.

Discussion
If the domain does not start with a dot, then the cookie will only be sent to the exact host specified by the domain. If the domain does start with a dot, then the cookie will be sent to other hosts in that domain as well, subject to certain restrictions. See RFC 2965 for more detail.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSHTTPCookie.h

expiresDate

Returns the receiver's expiration date.

```
- (NSDate *)expiresDate
```

Return Value
The receiver's expiration date, or `nil` if there is no specific expiration date such as in the case of "session-only" cookies. The expiration date is the date when the cookie should be deleted.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSHTTPCookie.h

initWithProperties:

Returns an initialized NSHTTPCookie object using the provided properties.

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

Parameters

properties
The properties for the new cookie object, expressed as key value pairs.

Return Value
The initialized cookie object. Returns nil if the provided properties are invalid.

Discussion
See "Constants" (page 721) for more information on the available header field constants and the constraints imposed on the values in the dictionary.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
+ cookieWithProperties: (page 715)

Declared In
NSHTTPCookie.h

isSecure

Returns whether his cookie should only be sent over secure channels.

```
- (BOOL)isSecure
```

Return Value
YES if this cookie should only be sent over secure channels, otherwise NO.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSHTTPCookie.h

isSessionOnly

Returns whether the receiver should be discarded at the end of the session (regardless of expiration date).

```
- (BOOL)isSessionOnly
```

Return Value
`YES` if the receiver should be discarded at the end of the session (regardless of expiration date), otherwise `NO`.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSHTTPCookie.h`

name

Returns the receiver's name.

```
- (NSString *)name
```

Return Value
The receiver's name.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSHTTPCookie.h`

path

Returns the receiver's path.

```
- (NSString *)path
```

Return Value
The receiver's path.

Discussion
The cookie will be sent with requests for this path in the cookie's domain, and all paths that have this prefix. A path of "/" means the cookie will be sent for all URLs in the domain.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSHTTPCookie.h`

portList

Returns the receiver's port list.

```
- (NSArray *)portList
```

Return Value
The list of ports for the cookie, returned as an array of `NSNumber` objects containing integers. If the cookie has no port list this method returns `nil` and the cookie will be sent to any port. Otherwise, the cookie is only sent to ports specified in the port list.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSHTTPCookie.h`

properties

Returns the receiver's cookie properties.

```
- (NSDictionary *)properties
```

Return Value
A dictionary representation of the receiver's cookie properties.

Discussion
This dictionary can be used with `initWithProperties:` (page 718) or `cookieWithProperties:` (page 715) to create an equivalent `NSHTTPCookie` object.

See `initWithProperties:` (page 718) for more information on the constraints imposed on the *properties* dictionary.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSHTTPCookie.h`

value

Returns the receiver's value.

```
- (NSString *)value
```

Return Value
The receiver's value.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSHTTPCookie.h

version

Returns the receiver's version.

```
- (NSUInteger)version
```

Return Value
The receiver's version. Version 0 maps to "old-style" Netscape cookies. Version 1 maps to RFC 2965 cookies.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSHTTPCookie.h

Constants

HTTP Cookie Property Keys

These constants define the supported keys in a dictionary containing cookie attributes.

```
extern NSString *NSHTTPCookieComment;
extern NSString *NSHTTPCookieCommentURL;
extern NSString *NSHTTPCookieDiscard;
extern NSString *NSHTTPCookieDomain;
extern NSString *NSHTTPCookieExpires;
extern NSString *NSHTTPCookieMaximumAge;
extern NSString *NSHTTPCookieName;
extern NSString *NSHTTPCookieOriginURL;
extern NSString *NSHTTPCookiePath;
extern NSString *NSHTTPCookiePort;
extern NSString *NSHTTPCookieSecure;
extern NSString *NSHTTPCookieValue;
extern NSString *NSHTTPCookieVersion;
```

Constants
NSHTTPCookieComment
An NSString object containing the comment for the cookie.
Only valid for Version 1 cookies and later. This header field is optional.
Available in Mac OS X v10.2 and later.
Declared in NSHTTPCookie.h.

`NSHTTPCookieCommentURL`

An `NSURL` object or `NSString` object containing the comment URL for the cookie.

Only valid for Version 1 cookies or later. This header field is optional.

Available in Mac OS X v10.2 and later.

Declared in `NSHTTPCookie.h`.

`NSHTTPCookieDiscard`

An `NSString` object stating whether the cookie should be discarded at the end of the session.

String value must be either "TRUE" or "FALSE". This header field is optional. Default is "FALSE", unless this is cookie is version 1 or greater and a value for `NSHTTPCookieMaximumAge` is not specified, in which case it is assumed "TRUE".

Available in Mac OS X v10.2 and later.

Declared in `NSHTTPCookie.h`.

`NSHTTPCookieDomain`

An `NSString` object containing the domain for the cookie.

A value must be specified for either `NSHTTPCookieDomain` or `NSHTTPCookieOriginURL`. If this header field is missing the domain is inferred from the value for `NSHTTPCookieOriginURL`.

Available in Mac OS X v10.2 and later.

Declared in `NSHTTPCookie.h`.

`NSHTTPCookieExpires`

An `NSDate` object or `NSString` object specifying the expiration date for the cookie.

This header field is only used for Version 0 cookies. This header field is optional.

Available in Mac OS X v10.2 and later.

Declared in `NSHTTPCookie.h`.

`NSHTTPCookieMaximumAge`

An `NSString` object containing an integer value stating how long in seconds the cookie should be kept, at most.

Only valid for Version 1 cookies and later. Default is "0". This field is optional.

Available in Mac OS X v10.2 and later.

Declared in `NSHTTPCookie.h`.

`NSHTTPCookieName`

An `NSString` object containing the name of the cookie. This field is required.

Available in Mac OS X v10.2 and later.

Declared in `NSHTTPCookie.h`.

`NSHTTPCookieOriginURL`

An NSURL or `NSString` object containing the URL that set this cookie.

A value must be specified for either `NSHTTPCookieDomain` or `NSHTTPCookieOriginURL`.

Available in Mac OS X v10.2 and later.

Declared in `NSHTTPCookie.h`.

`NSHTTPCookiePath`

An `NSString` object containing the path for the cookie.

Inferred from the value for `NSHTTPCookieOriginURL` if not provided. Default is "/". This header field is optional.

Available in Mac OS X v10.2 and later.

Declared in `NSHTTPCookie.h`.

`NSHTTPCookiePort`

An `NSString` object containing comma-separated integer values specifying the ports for the cookie.

Only valid for Version 1 cookies or later. The default value is an empty string (""). This header field is optional.

Available in Mac OS X v10.2 and later.

Declared in `NSHTTPCookie.h`.

`NSHTTPCookieSecure`

An `NSString` object stating whether the cookie should be transmitted only over secure channels.

String value must be either "TRUE" or "FALSE". Default is "FALSE". This header field is optional.

Available in Mac OS X v10.2 and later.

Declared in `NSHTTPCookie.h`.

`NSHTTPCookieValue`

An `NSString` object containing the value of the cookie.

This header field is required.

Available in Mac OS X v10.2 and later.

Declared in `NSHTTPCookie.h`.

`NSHTTPCookieVersion`

An `NSString` object that specifies the version of the cookie.

Must be either "0" or "1". The default is "0". This header field is optional.

Available in Mac OS X v10.2 and later.

Declared in `NSHTTPCookie.h`.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

`NSHTTPCookie.h`

NSHTTPCookieStorage Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSHTTPCookieStorage.h
Availability	Available in Mac OS X v10.2 with Safari 1.0 installed. Available in Mac OS X v10.2.7 and later.
Companion guide	URL Loading System

Overview

`NSHTTPCookieStorage` implements a singleton object (shared instance) that manages the shared cookie storage. These cookies are shared among all applications and are kept in sync cross-process.

Note: Changes made to the cookie accept policy will affect all currently running applications using the cookie storage.

Tasks

Getting the Shared Cookie Storage Object

`+ sharedHTTPCookieStorage` (page 726)
Returns the shared cookie storage instance.

Getting and Setting the Cookie Accept Policy

`- cookieAcceptPolicy` (page 726)
Returns the receiver's cookie accept policy.

`- setCookieAcceptPolicy:` (page 728)
Sets the cookie accept policy of the receiver

Adding and Removing Cookies

- `cookies` (page 727)
  Returns the receiver's cookies.
- `cookiesForURL:` (page 727)
  Returns all the receiver's cookies that will be sent to a specified URL.
- `deleteCookie:` (page 728)
  Deletes the specified cookie from the receiver.
- `setCookie:` (page 728)
  Stores a specified cookie in the receiver if the receiver's cookie accept policy permits.
- `setCookies:forURL:mainDocumentURL:` (page 729)
  Adds an array of cookies to the receiver if the receiver's cookie acceptance policy permits.

Class Methods

sharedHTTPCookieStorage

Returns the shared cookie storage instance.

```
+ (NSHTTPCookieStorage *)sharedHTTPCookieStorage
```

Return Value
The shared cookie storage instance.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSHTTPCookieStorage.h`

Instance Methods

cookieAcceptPolicy

Returns the receiver's cookie accept policy.

```
- (NSHTTPCookieAcceptPolicy)cookieAcceptPolicy
```

Return Value
The receiver's cookie accept policy. The default cookie accept policy is `NSHTTPCookieAcceptPolicyAlways`.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- `setCookieAcceptPolicy:` (page 728)

Declared In
`NSHTTPCookieStorage.h`

cookies

Returns the receiver's cookies.

```
- (NSArray *)cookies
```

Return Value
An array containing all of the receiver's cookies.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- `cookiesForURL:` (page 727)

Declared In
`NSHTTPCookieStorage.h`

cookiesForURL:

Returns all the receiver's cookies that will be sent to a specified URL.

```
- (NSArray *)cookiesForURL:(NSURL *)theURL
```

Parameters
theURL
The URL to filter on.

Return Value
An array of cookies whose URL matches the provided URL.

Discussion
An application can use `NSHTTPCookie`'s `requestHeaderFieldsWithCookies:` (page 716) method to turn this array into a set of header fields to add to an `NSMutableURLRequest` object.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- `cookies` (page 727)

Declared In
`NSHTTPCookieStorage.h`

deleteCookie:

Deletes the specified cookie from the receiver.

```
- (void)deleteCookie:(NSHTTPCookie *)aCookie
```

Parameters

aCookie

The cookie to delete.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

NSHTTPCookieStorage.h

setCookie:

Stores a specified cookie in the receiver if the receiver's cookie accept policy permits.

```
- (void)setCookie:(NSHTTPCookie *)aCookie
```

Parameters

aCookie

The cookie to store.

Discussion

The cookie will replace an existing cookie with the same name, domain and path, if one exists in the cookie storage. This method will accept the cookie only if the receiver's cookie accept policy is `NSHTTPCookieAcceptPolicyAlways` or `NSHTTPCookieAcceptPolicyOnlyFromMainDocumentDomain`. The cookie will be ignored if the receiver's cookie accept policy is `NSHTTPCookieAcceptPolicyNever`.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

NSHTTPCookieStorage.h

setCookieAcceptPolicy:

Sets the cookie accept policy of the receiver

```
- (void)setCookieAcceptPolicy:(NSHTTPCookieAcceptPolicy)aPolicy
```

Parameters

aPolicy

The new cookie accept policy.

Discussion

The default cookie accept policy is `NSHTTPCookieAcceptPolicyAlways`. Changing the cookie policy will affect all currently running applications using the cookie storage.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- `cookieAcceptPolicy` (page 726)

Declared In
`NSHTTPCookieStorage.h`

setCookies:forURL:mainDocumentURL:

Adds an array of cookies to the receiver if the receiver's cookie acceptance policy permits.

```
- (void)setCookies:(NSArray *)cookies forURL:(NSURL *)theURL mainDocumentURL:(NSURL
    *)mainDocumentURL
```

Parameters

cookies
The cookies to add.

theURL
The URL associated with the added cookies.

mainDocumentURL
The URL of the main HTML document for the top-level frame, if known. Can be `nil`. This URL is used to determine if the cookie should be accepted if the cookie accept policy is `NSHTTPCookieAcceptPolicyOnlyFromMainDocumentDomain`.

Discussion
The cookies will replace existing cookies with the same name, domain, and path, if one exists in the cookie storage. The cookie will be ignored if the receiver's cookie accept policy is `NSHTTPCookieAcceptPolicyNever`.

To store cookies from a set of response headers, an application can use `cookiesWithResponseHeaderFields:forURL:` (page 715) passing a header field dictionary and then use this method to store the resulting cookies in accordance with the receiver's cookie acceptance policy.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSHTTPCookieStorage.h`

Constants

NSHTTPCookieAcceptPolicy

`NSHTTPCookieAcceptPolicy` specifies the cookie acceptance policies implemented by the `NSHTTPCookieStorage` class.

```
typedef enum {
    NSHTTPCookieAcceptPolicyAlways,
    NSHTTPCookieAcceptPolicyNever,
    NSHTTPCookieAcceptPolicyOnlyFromMainDocumentDomain
} NSHTTPCookieAcceptPolicy;
```

Constants

`NSHTTPCookieAcceptPolicyAlways`

Accept all cookies. This is the default cookie accept policy.

Available in Mac OS X v10.2 and later.

Declared in `NSHTTPCookieStorage.h`.

`NSHTTPCookieAcceptPolicyNever`

Reject all cookies.

Available in Mac OS X v10.2 and later.

Declared in `NSHTTPCookieStorage.h`.

`NSHTTPCookieAcceptPolicyOnlyFromMainDocumentDomain`

Accept cookies only from the main document domain.

Available in Mac OS X v10.2 and later.

Declared in `NSHTTPCookieStorage.h`.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

`NSHTTPCookieStorage.h`

Notifications

NSHTTPCookieManagerCookiesChangedNotification

This notification is posted when the cookies stored in the `NSHTTPCookieStorage` instance have changed. Since cookies are shared among applications, this notification can be sent in response to another application's actions.

The notification object is the `NSHTTPCookieStorage` instance. This notification does not contain a `userInfo` dictionary.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

`NSHTTPCookieStorage.h`

NSHTTPCookieManagerAcceptPolicyChangedNotification

This notification is posted when the acceptance policy of the `NSHTTPCookieStorage` instance has changed. Since cookies are shared among applications, this notification can be sent in response to another application's actions.

The notification object is the `NSHTTPCookieStorage` instance. This notification does not contain a `userInfo` dictionary.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSHTTPCookieStorage.h`

NSHTTPURLResponse Class Reference

Inherits from	NSURLResponse : NSObject
Conforms to	NSCoding (NSURLResponse) NSCopying (NSURLResponse) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSURLResponse.h
Availability	Available in Mac OS X v10.2 with Safari 1.0 installed. Available in Mac OS X v10.2.7 and later.
Companion guide	URL Loading System

Overview

An NSHTTPURLResponse object represents a response to an HTTP URL load request. It's a subclass of NSURLResponse that provides methods for accessing information specific to HTTP protocol responses.

Adopted Protocols

NSCoding
- `- encodeWithCoder:` (page 2034)
- `- initWithCoder:` (page 2034)

NSCopying
- `- copyWithZone:` (page 2042)

Tasks

Getting HTTP Response Headers

`- allHeaderFields` (page 734)
Returns all the HTTP header fields of the receiver.

Getting Response Status Code

`+ localizedStringForStatusCode:` (page 734)
Returns a localized string corresponding to a specified HTTP status code.

`- statusCode` (page 735)
Returns the receiver's HTTP status code.

Class Methods

localizedStringForStatusCode:

Returns a localized string corresponding to a specified HTTP status code.

```
+ (NSString *)localizedStringForStatusCode:(NSInteger)statusCode
```

Parameters

statusCode
The HTTP status code.

Return Value
A localized string suitable for displaying to users that describes the specified status code.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
`- statusCode` (page 735)

Declared In
`NSURLResponse.h`

Instance Methods

allHeaderFields

Returns all the HTTP header fields of the receiver.

```
- (NSDictionary *)allHeaderFields
```

Return Value
A dictionary containing all the HTTP header fields of the receiver. By examining this dictionary clients can see the "raw" header information returned by the HTTP server.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSURLResponse.h

statusCode

Returns the receiver's HTTP status code.

```
- (NSInteger)statusCode
```

Return Value
The receiver's HTTP status code.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
+ localizedStringForStatusCode: (page 734)

Declared In
NSURLResponse.h

NSIndexPath Class Reference

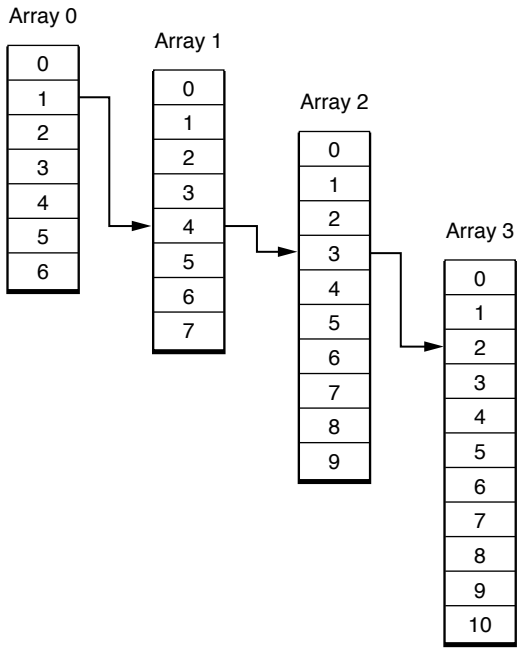
Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.4 and later.
Declared in	Foundation/NSIndexPath.h

Overview

The `NSIndexPath` class represents the path to a specific node in a tree of nested array collections. This path is known as an **index path**.

Each index in an index path represents the index into an array of children from one node in the tree to another, deeper, node. For example, the index path `1.4.3.2` specifies the path shown in Figure 56-1.

Figure 56-1 Index path `1.4.3.2`



`NSIndexPath` objects are uniqued and shared. If an index path containing the specified index or indexes already exists, that object is returned instead of a new instance.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 2034)
- `initWithCoder:` (page 2034)

NSCopying
- `copyWithZone:` (page 2042)

Tasks

Creating Index Paths

`+ indexPathWithIndex:` (page 739)
Creates an one-node index path.

`+ indexPathWithIndexes:length:` (page 739)
Creates an index path with one or more nodes.

`- initWithIndex:` (page 742)
Initializes an allocated `NSIndexPath` (page 737) object with a one-node index path.

`- initWithIndexes:length:` (page 742)
Initializes an allocated `NSIndexPath` (page 737) object with an index path of a specific length.

Querying Index Paths

`- getIndexes:` (page 740)
Provides a reference to the receiver's indexes.

`- indexAtPosition:` (page 741)
Provides the index at a particular node in the receiver.

`- indexPathByAddingIndex:` (page 741)
Provides an index path containing the indexes in the receiver and another index.

`- indexPathByRemovingLastIndex` (page 741)
Provides an index path with the indexes in the receiver, excluding the last one.

`- length` (page 743)
Provides the number of indexes in the receiver.

Comparing Index Paths

`- compare:` (page 740)
Indicates the depth-first traversal order of the receiver and another index path.

Class Methods

indexPathWithIndex:

Creates an one-node index path.

```
+ (id)indexPathWithIndex:(NSUInteger)index
```

Parameters
index
Index of the item in node 0 to point to.

Return Value
One-node index path with *index*.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- initWithIndex:` (page 742)

Declared In
`NSIndexPath.h`

indexPathWithIndexes:length:

Creates an index path with one or more nodes.

```
+ (id)indexPathWithIndexes:(NSUInteger *)indexes length:(NSUInteger)length
```

Parameters
indexes
Array of indexes to make up the index path.
length
Number of nodes to include in the index path.

Return Value
Index path with *indexes* up to *length*.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- initWithIndexes:length:` (page 742)

Declared In
NSIndexPath.h

Instance Methods

compare:

Indicates the depth-first traversal order of the receiver and another index path.

```
- (NSComparisonResult)compare:(NSIndexPath *)indexPath
```

Parameters

indexPath

Index path to compare.

This value must not be `nil`. If the value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

Return Value
The depth-first traversal ordering of the receiver and *indexPath*.

- `NSOrderedAscending`: The receiver comes before *indexPath*.
- `NSOrderedDescending`: The receiver comes after *indexPath*.
- `NSOrderedSame`: The receiver and *indexPath* are the same index path.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSIndexPath.h

getIndexes:

Provides a reference to the receiver's indexes.

```
- (void)getIndexes:(NSUInteger *)indexes
```

Parameters

indexes

Pointer to an unsigned integer array. On return, the receiver indexes.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSIndexPath.h

indexAtPosition:

Provides the index at a particular node in the receiver.

```
- (NSUInteger)indexAtPosition:(NSUInteger)node
```

Parameters

node

Index value of the desired node. Node numbering starts at zero.

Return Value
Index value at *node*.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSIndexPath.h

indexPathByAddingIndex:

Provides an index path containing the indexes in the receiver and another index.

```
- (NSIndexPath *)indexPathByAddingIndex:(NSUInteger)index
```

Parameters

index

Index to append to the receiver's indexes.

Return Value
New NSIndexPath (page 737) object containing the receiver's indexes and *index*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- indexPathByRemovingLastIndex (page 741)

Declared In
NSIndexPath.h

indexPathByRemovingLastIndex

Provides an index path with the indexes in the receiver, excluding the last one.

```
- (NSIndexPath *)indexPathByRemovingLastIndex
```

Return Value
New index path with the receiver's indexes, excluding the last one.

Discussion
Returns an empty NSIndexPath instance if the receiver's length is 1 or less.

Special Considerations

On Mac OS X 10.4 and earlier this method returns `nil` when the length of the receiver is 1 or less. On Mac OS X 10.5 and later this method will never return `nil`.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- indexPathByAddingIndex:` (page 741)

Declared In

`NSIndexPath.h`

initWithIndex:

Initializes an allocated `NSIndexPath` (page 737) object with a one-node index path.

```
- (id)initWithIndex:(NSUInteger)index
```

Parameters

index

Index of the item in node 0 to point to.

Return Value

Initialized `NSIndexPath` (page 737) object representing a one-node index path with *index*.

Availability

Available in Mac OS X v10.4 and later.

See Also

`+ indexPathWithIndex:` (page 739)

Declared In

`NSIndexPath.h`

initWithIndexes:length:

Initializes an allocated `NSIndexPath` (page 737) object with an index path of a specific length.

```
- (id)initWithIndexes:(NSUInteger *)indexes length:(NSUInteger)length
```

Parameters

indexes

Array of indexes to make up the index path.

length

Number of nodes to include in the index path.

Return Value

Initialized `NSIndexPath` (page 737) object with *indexes* up to *length*.

Availability

Available in Mac OS X v10.4 and later.

See Also
`+ indexPathWithIndexes:length:` (page 739)

Declared In
`NSIndexPath.h`

length

Provides the number of indexes in the receiver.

```
- (NSUInteger)length
```

Return Value
Number of indexes in the receiver.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSIndexPath.h`

NSIndexSet Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSMutableCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.3 and later.
Declared in	Foundation/NSIndexSet.h
Companion guide	Collections Programming Topics for Cocoa
Related sample code	Core Data HTML Store IdentitySample ImageBrowser iSpend QTKitMovieShuffler

Overview

The `NSIndexSet` class represents an immutable collection of unique unsigned integers, known as **indexes** because of the way they are used. This collection is referred to as a **index set**.

You use index sets in your code to store indexes into some other data structure. For example, given an `NSArray` object, you could use an index set to identify a subset of objects in that array.

Each index value can appear only once in the index set. This is an important concept to understand and is why you would not use index sets to store an arbitrary collection of integer values. To illustrate how this works, if you created an `NSIndexSet` object with the values `4`, `5`, `2`, and `5`, the resulting set would only have the values `4`, `5`, and `2` in it. Because index values are always maintained in sorted order, the actual order of the values when you created the set would be `2`, `4`, and then `5`.

In most cases, using an index set is more efficient than storing a collection of individual integers. Internally, the `NSIndexSet` class represents indexes using ranges. For maximum performance and efficiency, overlapping ranges in an index set are automatically coalesced—that is, ranges merge rather than overlap. Thus, the more contiguous the indexes in the set, the fewer ranges are required to specify those indexes.

The designated initializers of the NSIndexSet class are: `initWithIndexesInRange:` (page 755) and `initWithIndexSet:` (page 756).

You must not subclass the `NSIndexSet` class.

The mutable subclass of `NSIndexSet` is NSMutableIndexSet.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 2034)
- `initWithCoder:` (page 2034)

NSCopying
- `copyWithZone:` (page 2042)

NSMutableCopying
- `mutableCopyWithZone:` (page 2094)

Tasks

Creating Index Sets

`+ indexSet` (page 747)
Creates an empty index set.

`+ indexSetWithIndex:` (page 748)
Creates an index set with an index.

`+ indexSetWithIndexesInRange:` (page 748)
Creates an index set with an index range.

`- init` (page 754)
Initializes an allocated `NSIndexSet` (page 745) object.

`- initWithIndex:` (page 755)
Initializes an allocated `NSIndexSet` (page 745) object with an index.

`- initWithIndexesInRange:` (page 755)
Initializes an allocated `NSIndexSet` (page 745) object with an index range.

`- initWithIndexSet:` (page 756)
Initializes an allocated `NSIndexSet` (page 745) object with an index set.

Querying Index Sets

`- containsIndex:` (page 749)
Indicates whether the receiver contains a specific index.

`- containsIndexes:` (page 749)
Indicates whether the receiver contains a superset of the indexes in another index set.

- `containsIndexesInRange:` (page 750)
  Indicates whether the receiver contains the indexes represented by an index range.
- `intersectsIndexesInRange:` (page 756)
  Indicates whether the receiver contains any of the indexes in a range.
- `count` (page 750)
  Returns the number of indexes in the receiver.
- `countOfIndexesInRange:` (page 751)
  Returns the number of indexes in the receiver that are members of a given range.

Comparing Index Sets

- `isEqualToIndexSet:` (page 756)
  Indicates whether the indexes in the receiver are the same indeces contained in another index set.

Getting Indexes

- `firstIndex` (page 751)
  Returns either the first index in the receiver or the not-found indicator.
- `lastIndex` (page 757)
  Returns either the last index in the receiver or the not-found indicator.
- `indexLessThanIndex:` (page 753)
  Returns either the closest index in the receiver that is less than a specific index or the not-found indicator.
- `indexLessThanOrEqualToIndex:` (page 754)
  Returns either the closest index in the receiver that is less than or equal to a specific index or the not-found indicator.
- `indexGreaterThanOrEqualToIndex:` (page 753)
  Returns either the closest index in the receiver that is greater than or equal to a specific index or the not-found indicator.
- `indexGreaterThanIndex:` (page 752)
  Returns either the closest index in the receiver that is greater than a specific index or the not-found indicator.
- `getIndexes:maxCount:inIndexRange:` (page 751)
  The receiver fills an index buffer with the indexes contained both in the receiver and in an index range, returning the number of indexes copied.

Class Methods

indexSet

Creates an empty index set.

```
+ (id)indexSet
```

Return Value
`NSIndexSet` (page 745) object with no members.

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

See Also
`- init` (page 754)

Related Sample Code
Core Data HTML Store

Declared In
`NSIndexSet.h`

indexSetWithIndex:

Creates an index set with an index.

```
+ (id)indexSetWithIndex:(NSUInteger)index
```

Parameters

index
An index.

Return Value
`NSIndexSet` (page 745) object containing *index*.

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

See Also
`- initWithIndex:` (page 755)

Related Sample Code
AutoSample
IdentitySample
PDFKitLinker2

Declared In
`NSIndexSet.h`

indexSetWithIndexesInRange:

Creates an index set with an index range.

```
+ (id)indexSetWithIndexesInRange:(NSRange)indexRange
```

Parameters

indexRange
An index range.

Return Value
`NSIndexSet` (page 745) object containing *indexRange*.

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

See Also
`- initWithIndexesInRange:` (page 755)

Related Sample Code
iSpend

Declared In
`NSIndexSet.h`

Instance Methods

containsIndex:

Indicates whether the receiver contains a specific index.

```
- (BOOL)containsIndex:(NSUInteger)index
```

Parameters

index
Index being inquired about.

Return Value
`YES` when the receiver contains *index*, `NO` otherwise.

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

See Also
`- containsIndexes:` (page 749)
`- containsIndexesInRange:` (page 750)

Declared In
`NSIndexSet.h`

containsIndexes:

Indicates whether the receiver contains a superset of the indexes in another index set.

```
- (BOOL)containsIndexes:(NSIndexSet *)indexSet
```

Parameters

indexSet
Index set being inquired about.

Return Value
YES when the receiver contains a superset of the indexes in *indexSet*, NO otherwise.

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

See Also
- containsIndex: (page 749)
- containsIndexesInRange: (page 750)

Declared In
NSIndexSet.h

containsIndexesInRange:

Indicates whether the receiver contains the indexes represented by an index range.

```
- (BOOL)containsIndexesInRange:(NSRange)indexRange
```

Parameters
indexRange
The index range being inquired about.

Return Value
YES when the receiver contains the indexes in *indexRange*, NO otherwise.

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

See Also
- containsIndex: (page 749)
- containsIndexes: (page 749)
- intersectsIndexesInRange: (page 756)

Declared In
NSIndexSet.h

count

Returns the number of indexes in the receiver.

```
- (NSUInteger)count
```

Return Value
Number of indexes in the receiver.

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

See Also
- countOfIndexesInRange: (page 751)

Declared In
NSIndexSet.h

countOfIndexesInRange:

Returns the number of indexes in the receiver that are members of a given range.

```
- (NSUInteger)countOfIndexesInRange:(NSRange)indexRange
```

Parameters

indexRange
Index range being inquired about.

Return Value
Number of indexes in the receiver that are members of *indexRange*.

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

See Also
- count (page 750)

Declared In
NSIndexSet.h

firstIndex

Returns either the first index in the receiver or the not-found indicator.

```
- (NSUInteger)firstIndex
```

Return Value
First index in the receiver or NSNotFound (page 2287) when the receiver is empty.

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

See Also
- lastIndex (page 757)

Related Sample Code
AutomatorHandsOn
iSpend

Declared In
NSIndexSet.h

getIndexes:maxCount:inIndexRange:

The receiver fills an index buffer with the indexes contained both in the receiver and in an index range, returning the number of indexes copied.

```
- (NSUInteger)getIndexes:(NSUInteger *)indexBuffer maxCount:(NSUInteger)bufferSize
    inIndexRange:(NSRangePointer)indexRangePointer
```

Parameters

indexBuffer

Index buffer to fill.

bufferSize

Maximum size of *indexBuffer*.

indexRange

Index range to compare with indexes in the receiver; `nil` represents all the indexes in the receiver. Indexes in the index range and in the receiver are copied to *indexBuffer*. On output, the range of indexes not copied to *indexBuffer*.

Return Value
Number of indexes placed in *indexBuffer*.

Discussion
You are responsible for allocating the memory required for *indexBuffer* and for releasing it later.

Suppose you have an index set with contiguous indexes from 1 to 100. If you use this method to request a range of `(1, 100)`—which represents the set of indexes 1 through 100—and specify a buffer size of `20`, this method returns 20 indexes—1 through 20—in *indexBuffer* and sets *indexRange* to `(21, 80)`—which represents the indexes 21 through 100.

Use this method to retrieve entries quickly and efficiently from an index set. You can call this method repeatedly to retrieve blocks of index values and then process them. When doing so, use the return value and *indexRange* to determine when you have finished processing the desired indexes. When the return value is less than *bufferSize*, you have reached the end of the range.

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

Declared In
`NSIndexSet.h`

indexGreaterThanIndex:

Returns either the closest index in the receiver that is greater than a specific index or the not-found indicator.

```
- (NSUInteger)indexGreaterThanIndex:(NSUInteger)index
```

Parameters

index

Index being inquired about.

Return Value
Closest index in the receiver greater than *index*; `NSNotFound` (page 2287) when the receiver contains no qualifying index.

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

See Also
`- indexLessThanIndex:` (page 753)

- indexGreaterThanOrEqualToIndex: (page 753)
- indexLessThanOrEqualToIndex: (page 754)

Related Sample Code
AutomatorHandsOn
Core Data HTML Store
QTKitMovieShuffler

Declared In
NSIndexSet.h

indexGreaterThanOrEqualToIndex:

Returns either the closest index in the receiver that is greater than or equal to a specific index or the not-found indicator.

```
- (NSUInteger)indexGreaterThanOrEqualToIndex:(NSUInteger)index
```

Parameters

index
Index being inquired about.

Return Value
Closest index in the receiver greater than or equal to *index*; NSNotFound (page 2287) when the receiver contains no qualifying index.

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

See Also
- indexGreaterThanIndex: (page 752)
- indexLessThanIndex: (page 753)
- indexLessThanOrEqualToIndex: (page 754)

Declared In
NSIndexSet.h

indexLessThanIndex:

Returns either the closest index in the receiver that is less than a specific index or the not-found indicator.

```
- (NSUInteger)indexLessThanIndex:(NSUInteger)index
```

Parameters

index
Index being inquired about.

Return Value
Closest index in the receiver less than *index*; NSNotFound (page 2287) when the receiver contains no qualifying index.

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

See Also
- `indexGreaterThanIndex:` (page 752)
- `indexGreaterThanOrEqualToIndex:` (page 753)
- `indexLessThanOrEqualToIndex:` (page 754)

Related Sample Code
ImageBrowser

Declared In
`NSIndexSet.h`

indexLessThanOrEqualToIndex:

Returns either the closest index in the receiver that is less than or equal to a specific index or the not-found indicator.

```
- (NSUInteger)indexLessThanOrEqualToIndex:(NSUInteger)index
```

Parameters

index
Index being inquired about.

Return Value
Closest index in the receiver less than or equal to *index*; `NSNotFound` (page 2287) when the receiver contains no qualifying index.

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

See Also
- `indexGreaterThanIndex:` (page 752)
- `indexLessThanIndex:` (page 753)
- `indexGreaterThanOrEqualToIndex:` (page 753)

Declared In
`NSIndexSet.h`

init

Initializes an allocated `NSIndexSet` (page 745) object.

```
- (id)init
```

Return Value
Initialized, empty `NSIndexSet` (page 745) object.

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

See Also
- `indexSet` (page 747)

Declared In
NSIndexSet.h

initWithIndex:

Initializes an allocated NSIndexSet (page 745) object with an index.

```
- (id)initWithIndex:(NSUInteger)index
```

Parameters

index
An index.

Return Value
Initialized NSIndexSet (page 745) object with *index*.

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

See Also
+ indexSetWithIndex: (page 748)

Declared In
NSIndexSet.h

initWithIndexesInRange:

Initializes an allocated NSIndexSet (page 745) object with an index range.

```
- (id)initWithIndexesInRange:(NSRange)indexRange
```

Parameters

indexRange
An index range. Must include only indexes representable as unsigned integers.

Return Value
Initialized NSIndexSet (page 745) object with *indexRange*.

Discussion
This method raises an NSRangeException (page 2306) when *indexRange* would add an index that exceeds the maximum allowed value for unsigned integers.

This method is a designated initializer for NSIndexSet (page 745).

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

See Also
+ indexSetWithIndexesInRange: (page 748)

Declared In
NSIndexSet.h

initWithIndexSet:

Initializes an allocated `NSIndexSet` (page 745) object with an index set.

```
- (id)initWithIndexSet:(NSIndexSet *)indexSet
```

Parameters

indexSet

An index set.

Return Value

Initialized `NSIndexSet` (page 745) object with *indexSet*.

Discussion

This method is a designated initializer for `NSIndexSet` (page 745).

Availability

Available in Mac OS X v10.3 and later.

Declared In

`NSIndexSet.h`

intersectsIndexesInRange:

Indicates whether the receiver contains any of the indexes in a range.

```
- (BOOL)intersectsIndexesInRange:(NSRange)indexRange
```

Parameters

indexRange

Index range being inquired about.

Return Value

`YES` when the receiver contains one or more of the indexes in *indexRange*, `NO` otherwise.

Availability

Available in Mac OS X v10.3 and later.

See Also

`- containsIndexesInRange:` (page 750)

Declared In

`NSIndexSet.h`

isEqualToIndexSet:

Indicates whether the indexes in the receiver are the same indeces contained in another index set.

```
- (BOOL)isEqualToIndexSet:(NSIndexSet *)indexSet
```

Parameters

indexSet

Index set being inquired about.

Return Value
`YES` when the indexes in the receiver are the same indexes *`indexSet`* contains, `NO` otherwise.

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

Declared In
`NSIndexSet.h`

lastIndex

Returns either the last index in the receiver or the not-found indicator.

```
- (NSUInteger)lastIndex
```

Return Value
Last index in the receiver or `NSNotFound` (page 2287) when the receiver is empty.

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

See Also
`- firstIndex` (page 751)

Related Sample Code
IdentitySample
iSpend

Declared In
`NSIndexSet.h`

NSIndexSpecifier Class Reference

Inherits from	NSScriptObjectSpecifier : NSObject
Conforms to	NSCoding (NSScriptObjectSpecifier) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptObjectSpecifiers.h
Companion guide	Cocoa Scripting Guide
Related sample code	Quartz Composer WWDC 2005 TextEdit Sketch-112 TextEditPlus

Overview

The `NSIndexSpecifier` class represents an object in a collection (or container) with an index number. The script terms `first` and `front` specify the object with index `0`, while `last` specifies the object with index of `count-1`. A negative index indicates a location by counting backward from the last object in the collection.

You don't normally subclass `NSIndexSpecifier`.

Tasks

Creating Index Specifiers

`- initWithContainerClassDescription:containerSpecifier:key:index:` (page 760)
Initializes an allocated `NSIndexSpecifier` (page 759) object with a class description, container specifier, collection key, and object index.

Accessing the Index

`- index` (page 760)
Returns the value receiver's `index` property.

`- setIndex:` (page 761)
Sets the value of the receiver's `index` property.

Instance Methods

index

Returns the value receiver's `index` property.

```
- (NSInteger)index
```

Return Value
Value of the receiver's `index` property.

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

Declared In
`NSScriptObjectSpecifiers.h`

initWithContainerClassDescription:containerSpecifier:key:index:

Initializes an allocated `NSIndexSpecifier` (page 759) object with a class description, container specifier, collection key, and object index.

```
- (id)initWithContainerClassDescription:(NSScriptClassDescription *)classDescription
    containerSpecifier:(NSScriptObjectSpecifier *)containerSpecifier key:(NSString
    *)collectionKey index:(NSInteger)objectIndex
```

Parameters
classDescription
Description for the container of the collection.
containerSpecifier
Container of the collection.
collectionKey
Name of the collection.
objectIndex
The object within the *key* collection the index specifier is to identify.

Return Value
Initialized `NSIndexSpecifier` (page 759) object with its `index` property set to *objectIndex*.

Discussion
Invokes the super class's `initWithContainerClassDescription:containerSpecifier:key:` (page 1418) method and sets the `index` property of the index specifier to *objectIndex*.

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

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus

Declared In
`NSScriptObjectSpecifiers.h`

setIndex:

Sets the value of the receiver's `index` property.

```
- (void)setIndex:(NSInteger)index
```

Parameters

index

Value for the receiver's `index` property.

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

Declared In
`NSScriptObjectSpecifiers.h`

NSInputStream Class Reference

Inherits from	NSStream : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.3 and later.
Declared in	Foundation/NSStream.h
Companion guide	Stream Programming Guide for Cocoa
Related sample code	CocoaEcho CocoaHTTPServer CocoaSOAP

Overview

`NSInputStream` is a subclass of `NSStream` that provides read-only stream functionality.

Subclassing Notes

`NSInputStream` is a concrete subclass of `NSStream` that gives you standard read-only access to stream data. Although `NSInputStream` is probably sufficient for most situations requiring access to stream data, you can create a subclass of `NSInputStream` if you want more specialized behavior (for example, you want to record statistics on the data in a stream).

Methods to Override

To create a subclass of `NSInputStream` you may have to implement initializers for the type of stream data supported and suitably reimplement existing initializers. You must also provide complete implementations of the following methods:

- `read:maxLength:` (page 767)

  From the current read index, take up to the number of bytes specified in the second parameter from the stream and place them in the client-supplied buffer (first parameter). The buffer must be of the size specified by the second parameter. Return the actual number of bytes placed in the buffer; if there is nothing left in the stream, return `0`. Reset the index into the stream for the next read operation.

- `getBuffer:length:` (page 765)

Return in 0(1) a pointer to the subclass-allocated buffer (first parameter). Return by reference in the second parameter the number of bytes actually put into the buffer. The buffer's contents are valid only until the next stream operation. Return `NO` if you cannot access data in the buffer; otherwise, return `YES`. If this method is not appropriate for your type of stream, you may return `NO`.

- `hasBytesAvailable` (page 766)

  Return `YES` if there is more data to read in the stream, `NO` if there is not. If you want to be semantically compatible with `NSInputStream`, return `YES` if a read must be attempted to determine if bytes are available.

Tasks

Creating Streams

`+ inputStreamWithData:` (page 764)
Creates and returns an initialized `NSInputStream` object for reading from a given `NSData` object.

`+ inputStreamWithFileAtPath:` (page 765)
Creates and returns an initialized `NSInputStream` object that reads data from the file at a given path.

`- initWithData:` (page 766)
Initializes and returns an `NSInputStream` object for reading from a given `NSData` object.

`- initWithFileAtPath:` (page 767)
Initializes and returns an `NSInputStream` object that reads data from the file at a given path.

Using Streams

`- read:maxLength:` (page 767)
Reads up to a given number of bytes into a given buffer, and returns the actual number of bytes read.

`- getBuffer:length:` (page 765)
Returns by reference a pointer to a read buffer and, by reference, the number of bytes available, and returns a Boolean value that indicates whether the buffer is available.

`- hasBytesAvailable` (page 766)
Returns a Boolean value that indicates whether the receiver has bytes available to read.

Class Methods

inputStreamWithData:

Creates and returns an initialized `NSInputStream` object for reading from a given `NSData` object.

```
+ (id)inputStreamWithData:(NSData *)data
```

Parameters

data

The data object from which to read. The contents of *data* are copied.

Return Value

An initialized `NSInputStream` object for reading from *data*. If *data* is not an NSData object, this method returns `nil`.

Availability

Available in Mac OS X v10.3 and later.

See Also

`+ inputStreamWithFileAtPath:` (page 765)

`- initWithData:` (page 766)

Declared In

`NSStream.h`

inputStreamWithFileAtPath:

Creates and returns an initialized `NSInputStream` object that reads data from the file at a given path.

```
+ (id)inputStreamWithFileAtPath:(NSString *)path
```

Parameters

path

The path to the file.

Return Value

An initialized `NSInputStream` object that reads data from the file at *path*. If the file specified by *path* doesn't exist or is unreadable, returns `nil`.

Availability

Available in Mac OS X v10.3 and later.

See Also

`+ inputStreamWithData:` (page 764)

`- initWithFileAtPath:` (page 767)

Declared In

`NSStream.h`

Instance Methods

getBuffer:length:

Returns by reference a pointer to a read buffer and, by reference, the number of bytes available, and returns a Boolean value that indicates whether the buffer is available.

```
- (BOOL)getBuffer:(uint8_t **)buffer length:(NSUInteger *)len
```

Parameters

buffer

Upon return, contains a pointer to a read buffer. The buffer is only valid until the next stream operation is performed.

len

Upon return, contains the number of bytes available.

Return Value
YES if the buffer is available, otherwise NO.

Subclasses of NSInputStream may return NO if this operation is not appropriate for the stream type.

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

Declared In
NSStream.h

hasBytesAvailable

Returns a Boolean value that indicates whether the receiver has bytes available to read.

```
- (BOOL)hasBytesAvailable
```

Return Value
YES if the receiver has bytes available to read, otherwise NO. May also return YES if a read must be attempted in order to determine the availability of bytes.

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

Declared In
NSStream.h

initWithData:

Initializes and returns an NSInputStream object for reading from a given NSData object.

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

Parameters

data

The data object from which to read. The contents of *data* are copied.

Return Value
An initialized NSInputStream object for reading from *data*.

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

See Also
- initWithFileAtPath: (page 767)
+ inputStreamWithData: (page 764)

Declared In
NSStream.h

initWithFileAtPath:

Initializes and returns an NSInputStream object that reads data from the file at a given path.

```
- (id)initWithFileAtPath:(NSString *)path
```

Parameters

path

The path to the file.

Return Value
An initialized NSInputStream object that reads data from the file at *path*. If the file specified by *path* doesn't exist or is unreadable, returns nil.

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

See Also
- initWithData: (page 766)
+ inputStreamWithFileAtPath: (page 765)

Declared In
NSStream.h

read:maxLength:

Reads up to a given number of bytes into a given buffer, and returns the actual number of bytes read.

```
- (NSInteger)read:(uint8_t *)buffer maxLength:(NSUInteger)len
```

Parameters

buffer

A data buffer. The buffer must be large enough to contain the number of bytes specified by *len*.

len

The maximum number of bytes to read.

Return Value
The actual number of bytes read.

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

Related Sample Code
CocoaEcho
CocoaHTTPServer
CocoaSOAP

Declared In
NSStream.h

NSInvocation Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSInvocation.h
Companion guide	Distributed Objects Programming Topics
Related sample code	CubePuzzle DeskPictAppDockMenu

Overview

An `NSInvocation` is an Objective-C message rendered static, that is, it is an action turned into an object. `NSInvocation` objects are used to store and forward messages between objects and between applications, primarily by `NSTimer` objects and the distributed objects system.

An `NSInvocation` object contains all the elements of an Objective-C message: a target, a selector, arguments, and the return value. Each of these elements can be set directly, and the return value is set automatically when the `NSInvocation` object is dispatched.

An `NSInvocation` object can be repeatedly dispatched to different targets; its arguments can be modified between dispatch for varying results; even its selector can be changed to another with the same method signature (argument and return types). This flexibility makes `NSInvocation` useful for repeating messages with many arguments and variations; rather than retyping a slightly different expression for each message, you modify the `NSInvocation` object as needed each time before dispatching it to a new target.

`NSInvocation` does not support invocations of methods with either variable numbers of arguments or `union` arguments. You should use the `invocationWithMethodSignature:` (page 771) class method to create `NSInvocation` objects; you should not create these objects using `alloc` (page 1152) and `init` (page 1178).

This class does not retain the arguments for the contained invocation by default. If those objects might disappear between the time you create your instance of `NSInvocation` and the time you use it, you should explicitly retain the objects yourself or invoke the `retainArguments` method to have the invocation object retain them itself.

Note: `NSInvocation` conforms to the `NSCoding` protocol, but only supports coding by an `NSPortCoder`. `NSInvocation` does not support archiving.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 2034)
- `initWithCoder:` (page 2034)

Tasks

Creating NSInvocation Objects

`+ invocationWithMethodSignature:` (page 771)
Returns an `NSInvocation` object able to construct messages using a given method signature.

Configuring an Invocation Object

`- setSelector:` (page 777)
Sets the receiver's selector.

`- selector` (page 775)
Returns the receiver's selector, or 0 if it hasn't been set.

`- setTarget:` (page 777)
Sets the receiver's targe.

`- target` (page 777)
Returns the receiver's target, or `nil` if the receiver has no target.

`- setArgument:atIndex:` (page 775)
Sets an argument of the receiver.

`- getArgument:atIndex:` (page 772)
Returns by indirection the receiver's argument at a specified index.

`- argumentsRetained` (page 771)
Returns `YES` if the receiver has retained its arguments, `NO` otherwise.

`- retainArguments` (page 775)
If the receiver hasn't already done so, retains the target and all object arguments of the receiver and copies all of its C-string arguments.

`- setReturnValue:` (page 776)
Sets the receiver's return value.

`- getReturnValue:` (page 773)
Gets the receiver's return value.

Dispatching an Invocation

- `invoke` (page 773)
  Sends the receiver's message (with arguments) to its target and sets the return value.
- `invokeWithTarget:` (page 774)
  Sets the receiver's target, sends the receiver's message (with arguments) to that target, and sets the return value.

Getting the Method Signature

- `methodSignature` (page 774)
  Returns the receiver's method signature.

Class Methods

invocationWithMethodSignature:

Returns an `NSInvocation` object able to construct messages using a given method signature.

```
+ (NSInvocation *)invocationWithMethodSignature:(NSMethodSignature *)signature
```

Parameters

signature
  An object encapsulating a method signature.

Discussion
The new object must have its selector set with `setSelector:` (page 777) and its arguments set with `setArgument:atIndex:` (page 775) before it can be invoked. Do not use the `alloc` (page 1152)/`init` (page 1178) approach to create `NSInvocation` objects.

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

Related Sample Code
CubePuzzle
DeskPictAppDockMenu

Declared In
`NSInvocation.h`

Instance Methods

argumentsRetained

Returns `YES` if the receiver has retained its arguments, `NO` otherwise.

```
- (BOOL)argumentsRetained
```

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

See Also
- `retainArguments` (page 775)

Declared In
`NSInvocation.h`

getArgument:atIndex:

Returns by indirection the receiver's argument at a specified index.

```
- (void)getArgument:(void *)buffer atIndex:(NSInteger)index
```

Parameters

buffer

An untyped buffer to hold the returned argument. See the discussion below relating to argument values that are objects.

index

An integer specifying the index of the argument to get.

Indices 0 and 1 indicate the hidden arguments *self* and *_cmd*, respectively; these values can be retrieved directly with the `target` and `selector` methods. Use indices 2 and greater for the arguments normally passed in a message.

Discussion
This method copies the argument stored at *index* into the storage pointed to by *buffer*. The size of *buffer* must be large enough to accommodate the argument value.

When the argument value is an object, pass a pointer to the variable (or memory) into which the object should be placed:

```
NSArray *anArray;
[invocation getArgument:&anArray atIndex:3];
```

This method raises `NSInvalidArgumentException` if *index* is greater than the actual number of arguments for the selector.

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

See Also
- `setArgument:atIndex:` (page 775)
- `numberOfArguments` (page 901) (NSMethodSignature)

Declared In
`NSInvocation.h`

getReturnValue:

Gets the receiver's return value.

```
- (void)getReturnValue:(void *)buffer
```

Parameters

buffer

An untyped buffer into which the receiver copies its return value. It should be large enough to accommodate the value. See the discussion below for more information about *buffer*.

Discussion

Use the `NSMethodSignature` method `methodReturnLength` (page 900) to determine the size needed for *buffer*:

```
NSUInteger length = [[myInvocation methodSignature] methodReturnLength];
buffer = (void *)malloc(length);
[invocation getReturnValue:buffer];
```

When the return value is an object, pass a pointer to the variable (or memory) into which the object should be placed:

```
id anObject;
NSArray *anArray;
[invocation1 getReturnValue:&anObject];
[invocation2 getReturnValue:&anArray];
```

If the `NSInvocation` object has never been invoked, the result of this method is undefined.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `setReturnValue:` (page 776)
- `methodReturnType` (page 901) (NSMethodSignature)

Related Sample Code

CubePuzzle

Declared In

`NSInvocation.h`

invoke

Sends the receiver's message (with arguments) to its target and sets the return value.

```
- (void)invoke
```

Discussion

You must set the receiver's target, selector, and argument values before calling this method.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `getReturnValue:` (page 773)

- setSelector: (page 777)
- setTarget: (page 777)
- setArgument:atIndex: (page 775)

Related Sample Code
CubePuzzle

Declared In
NSInvocation.h

invokeWithTarget:

Sets the receiver's target, sends the receiver's message (with arguments) to that target, and sets the return value.

```
- (void)invokeWithTarget:(id)anObject
```

Parameters

anObject
The object to set as the receiver's target.

Discussion
You must set the receiver's selector and argument values before calling this method.

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

See Also
- getReturnValue: (page 773)
- invoke (page 773)
- setSelector: (page 777)
- setTarget: (page 777)
- setArgument:atIndex: (page 775)

Declared In
NSInvocation.h

methodSignature

Returns the receiver's method signature.

```
- (NSMethodSignature *)methodSignature
```

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

Declared In
NSInvocation.h

retainArguments

If the receiver hasn't already done so, retains the target and all object arguments of the receiver and copies all of its C-string arguments.

```
- (void)retainArguments
```

Discussion
Before this method is invoked, `argumentsRetained` (page 771) returns `NO`; after, it returns `YES`.

For efficiency, newly created NSInvocations don't retain or copy their arguments, nor do they retain their targets or copy C strings. You should instruct an NSInvocation to retain its arguments if you intend to cache it, since the arguments may otherwise be released before the NSInvocation is invoked. NSTimers always instruct their NSInvocations to retain their arguments, for example, because there's usually a delay before an NSTimer fires.

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

Declared In
`NSInvocation.h`

selector

Returns the receiver's selector, or 0 if it hasn't been set.

```
- (SEL)selector
```

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

See Also
`- setSelector:` (page 777)

Declared In
`NSInvocation.h`

setArgument:atIndex:

Sets an argument of the receiver.

```
- (void)setArgument:(void *)buffer atIndex:(NSInteger)index
```

Parameters

buffer

An untyped buffer containing an argument to be assigned to the receiver. See the discussion below relating to argument values that are objects.

index

An integer specifying the index of the argument.

Indices 0 and 1 indicate the hidden arguments *self* and *_cmd*, respectively; you should set these values directly with the `setTarget:` (page 777) and `setSelector:` (page 777) methods. Use indices 2 and greater for the arguments normally passed in a message.

Discussion
This method copies the contents of *buffer* as the argument at *index*. The number of bytes copied is determined by the argument size.

When the argument value is an object, pass a pointer to the variable (or memory) from which the object should be copied:

```
NSArray *anArray;
[invocation setArgument:&anArray atIndex:3];
```

This method raises `NSInvalidArgumentException` if the value of *index* is greater than the actual number of arguments for the selector.

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

See Also
- `getArgument:atIndex:` (page 772)
- `numberOfArguments` (page 901) (NSMethodSignature)

Related Sample Code
CubePuzzle
DeskPictAppDockMenu

Declared In
`NSInvocation.h`

setReturnValue:

Sets the receiver's return value.

```
- (void)setReturnValue:(void *)buffer
```

Parameters
buffer
An untyped buffer whose contents are copied as the receiver's return value.

Discussion
This value is normally set when you send an `invoke` (page 773) or `invokeWithTarget:` (page 774) message.

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

See Also
- `getReturnValue:` (page 773)
- `methodReturnLength` (page 900) (NSMethodSignature)
- `methodReturnType` (page 901) (NSMethodSignature)

Declared In
`NSInvocation.h`

setSelector:

Sets the receiver's selector.

```
- (void)setSelector:(SEL)selector
```

Parameters

selector

The selector to assign to the receiver.

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

See Also
- selector (page 775)

Related Sample Code
CubePuzzle
DeskPictAppDockMenu

Declared In
NSInvocation.h

setTarget:

Sets the receiver's targe.

```
- (void)setTarget:(id)anObject
```

Parameters

anObject

The object to assign to the receiver as target. The target is the receiver of the message sent by invoke (page 773).

Discussion

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

See Also
- target (page 777)
- invokeWithTarget: (page 774)

Related Sample Code
CubePuzzle
DeskPictAppDockMenu

Declared In
NSInvocation.h

target

Returns the receiver's target, or nil if the receiver has no target.

```
- (id)target
```

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

See Also
`- setTarget:` (page 777)

Declared In
`NSInvocation.h`

Constants

Parameter Type Constants

Method argument types. (**Deprecated.** These constants are used internally by `NSInvocation`—you should not use them directly.)

```
enum _NSObjCValueType {
    NSObjCNoType = 0,
    NSObjCVoidType = 'v',
    NSObjCCharType = 'c',
    NSObjCShortType = 's',
    NSObjCLongType = 'l',
    NSObjCLonglongType = 'q',
    NSObjCFloatType = 'f',
    NSObjCDoubleType = 'd',

    NSObjCBoolType = 'B',

    NSObjCSelectorType = ':',
    NSObjCObjectType = '@',
    NSObjCStructType = '{',
    NSObjCPointerType = '^',
    NSObjCStringType = '*',
    NSObjCArrayType = '[',
    NSObjCUnionType = '(',
    NSObjCBitfield = 'b'
};
```

Constants

`NSObjCNoType`

No type information. (**Deprecated.** Used internally by `NSInvocation`—do not use it directly)

Available in Mac OS X v10.0 and later.

Declared in `NSInvocation.h`.

`NSObjCVoidType`

The `void` type. (**Deprecated.** Used internally by `NSInvocation`—do not use it directly)

Available in Mac OS X v10.0 and later.

Declared in `NSInvocation.h`.

`NSObjCCharType`

The `char` type. (**Deprecated.** Used internally by `NSInvocation`—do not use it directly)

Available in Mac OS X v10.0 and later.

Declared in `NSInvocation.h`.

`NSObjCShortType`

The `short` integer type. (**Deprecated.** Used internally by `NSInvocation`—do not use it directly)

Available in Mac OS X v10.0 and later.

Declared in `NSInvocation.h`.

`NSObjCLongType`

The `long` integer type. (**Deprecated.** Used internally by `NSInvocation`—do not use it directly)

Available in Mac OS X v10.0 and later.

Declared in `NSInvocation.h`.

`NSObjCLonglongType`

The `long long` integer type. (**Deprecated.** Used internally by `NSInvocation`—do not use it directly)

Available in Mac OS X v10.0 and later.

Declared in `NSInvocation.h`.

`NSObjCFloatType`

The `float` type. (**Deprecated.** Used internally by `NSInvocation`—do not use it directly)

Available in Mac OS X v10.0 and later.

Declared in `NSInvocation.h`.

`NSObjCDoubleType`

The `double` type. (**Deprecated.** Used internally by `NSInvocation`—do not use it directly)

Available in Mac OS X v10.0 and later.

Declared in `NSInvocation.h`.

`NSObjCBoolType`

The `BOOL` type. (**Deprecated.** Used internally by `NSInvocation`—do not use it directly)

Available in Mac OS X v10.2 and later.

Declared in `NSInvocation.h`.

`NSObjCSelectorType`

The `SEL` type. (**Deprecated.** Used internally by `NSInvocation`—do not use it directly)

Available in Mac OS X v10.0 and later.

Declared in `NSInvocation.h`.

`NSObjCObjectType`

The `id` type. (**Deprecated.** Used internally by `NSInvocation`—do not use it directly)

Available in Mac OS X v10.0 and later.

Declared in `NSInvocation.h`.

`NSObjCStructType`

The `struct` type. (**Deprecated.** Used internally by `NSInvocation`—do not use it directly)

Available in Mac OS X v10.0 and later.

Declared in `NSInvocation.h`.

`NSObjCPointerType`

The `void*` type. (**Deprecated.** Used internally by `NSInvocation`—do not use it directly)

Available in Mac OS X v10.0 and later.

Declared in `NSInvocation.h`.

`NSObjCStringType`

The `char*` type. (**Deprecated.** Used internally by `NSInvocation`—do not use it directly)

Available in Mac OS X v10.0 and later.

Declared in `NSInvocation.h`.

`NSObjCArrayType`

A C-style array of items. (**Deprecated.** Used internally by `NSInvocation`—do not use it directly)

Available in Mac OS X v10.0 and later.

Declared in `NSInvocation.h`.

`NSObjCUnionType`

A union `union` type. (**Deprecated.** Used internally by `NSInvocation`—do not use it directly)

Available in Mac OS X v10.0 and later.

Declared in `NSInvocation.h`.

`NSObjCBitfield`

A bit field. (**Deprecated.** Used internally by `NSInvocation`—do not use it directly)

Available in Mac OS X v10.0 and later.

Declared in `NSInvocation.h`.

Declared In
`NSInvocation.h`

NSInvocationOperation Class Reference

Inherits from	NSOperation : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	Foundation/NSOperation.h
Companion guide	Threading Programming Guide

Overview

The `NSInvocationOperation` class is a concrete subclass of `NSOperation` that manages the execution of a single encapsulated task specified as an invocation. You can use this class to initiate an operation that consists of invoking a selector on a specified object. This class implements a non-concurrent operation.

For more information on concurrent versus non-concurrent operations, see *NSOperation Class Reference*.

Tasks

Initialization

- `initWithTarget:selector:object:` (page 782)
  Returns an `NSInvocationOperation` object initialized with the specified target and selector.
- `initWithInvocation:` (page 782)
  Returns an `NSInvocationOperation` object initialized with the specified invocation object.

Getting Attributes

- `invocation` (page 783)
  Returns the receiver's invocation object.
- `result` (page 783)
  Returns the result of the invocation or method.

Instance Methods

initWithInvocation:

Returns an `NSInvocationOperation` object initialized with the specified invocation object.

```
- (id)initWithInvocation:(NSInvocation *)inv
```

Parameters

inv

The invocation object identifying the target object, selector, and parameter objects.

Return Value

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

Discussion

This method is the designated initializer. The receiver tells the invocation object to retain its arguments.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSOperation.h`

initWithTarget:selector:object:

Returns an `NSInvocationOperation` object initialized with the specified target and selector.

```
- (id)initWithTarget:(id)target selector:(SEL)sel object:(id)arg
```

Parameters

target

The object defining the specified selector.

sel

The selector to invoke when running the operation. The selector may take 0 or 1 parameters. If it accepts a parameter, the type of that parameter should be `id`.

arg

The parameter object to pass to the selector. If the selector does not take an argument, specify `nil`.

Return Value

An initialized `NSInvocationOperation` object or `nil` if the target object does not implement the specified selector.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSOperation.h`

invocation

Returns the receiver's invocation object.

```
- (NSInvocation *)invocation
```

Return Value
The invocation object identifying the target object, selector, and parameters to use to execute the operation's task.

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

See Also
- `initWithTarget:selector:object:` (page 782)
- `initWithInvocation:` (page 782)

Declared In
`NSOperation.h`

result

Returns the result of the invocation or method.

```
- (id)result
```

Return Value
The object returned by the method or an `NSValue` object containing the return value if it is not an object. If the method or invocation is not finished executing, this method returns `nil`.

Discussion
If an exception was raised during the execution of the method or invocation, this method raises that exception again. If the operation was cancelled or the invocation or method has a `void` return type, calling this method raises an exception; see "`Result Exceptions`" (page 783).

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

Declared In
`NSOperation.h`

Constants

Result Exceptions

Names of exceptions raised by `NSInvocationOperation` if there is an error when calling the `result` (page 783) method.

```
extern NSString * const NSInvocationOperationVoidResultException;
extern NSString * const NSInvocationOperationCancelledException;
```

Constants

`NSInvocationOperationVoidResultException`

The name of the exception raised if the `result` method is called for an invocation method with a `void` return type.

Available in Mac OS X v10.5 and later.

Declared in `NSOperation.h`.

`NSInvocationOperationCancelledException`

The name of the exception raised if the `result` method is called after the operation was cancelled.

Available in Mac OS X v10.5 and later.

Declared in `NSOperation.h`.

Declared In

`NSOperation.h`

NSKeyedArchiver Class Reference

Inherits from	NSCoder : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.2 and later.
Declared in	Foundation/NSKeyedArchiver.h
Companion guide	Archives and Serializations Programming Guide for Cocoa
Related sample code	CoreRecipes CustomAtomicStoreSubclass iSpend QTQuartzPlayer Squiggles

Overview

`NSKeyedArchiver`, a concrete subclass of `NSCoder`, provides a way to encode objects (and scalar values) into an architecture-independent format that can be stored in a file. When you archive a set of objects, the class information and instance variables for each object are written to the archive. `NSKeyedArchiver`'s companion class, `NSKeyedUnarchiver`, decodes the data in an archive and creates a set of objects equivalent to the original set.

A keyed archive differs from a non-keyed archive in that all the objects and values encoded into the archive are given names, or keys. When decoding a non-keyed archive, values have to be decoded in the same order in which they were encoded. When decoding a keyed archive, because values are requested by name, values can be decoded out of sequence or not at all. Keyed archives, therefore, provide better support for forward and backward compatibility.

The keys given to encoded values must be unique only within the scope of the current object being encoded. A keyed archive is hierarchical, so the keys used by object A to encode its instance variables do not conflict with the keys used by object B, even if A and B are instances of the same class. Within a single object, however, the keys used by a subclass can conflict with keys used in its superclasses.

An `NSArchiver` object can write the archive data to a file or to a mutable-data object (an instance of `NSMutableData`) that you provide.

Tasks

Initializing an NSKeyedArchiver Object

- `initForWritingWithMutableData:` (page 794)
  Returns the receiver, initialized for encoding an archive into a given a mutable-data object.

Archiving Data

+ `archivedDataWithRootObject:` (page 787)
  Returns an `NSData` object containing the encoded form of the object graph whose root object is given.

+ `archiveRootObject:toFile:` (page 788)
  Archives an object graph rooted at a given object by encoding it into a data object then atomically writes the resulting data object to a file at a given path, and returns a Boolean value that indicates whether the operation was successful.

- `finishEncoding` (page 794)
  Instructs the receiver to construct the final data stream.

- `outputFormat` (page 795)
  Returns the format in which the receiver encodes its data.

- `setOutputFormat:` (page 796)
  Sets the format in which the receiver encodes its data.

Encoding Data and Objects

- `archiver:didEncodeObject:` (page 797) *delegate method*
  Informs the delegate that a given object has been encoded.

- `archiverDidFinish:` (page 798) *delegate method*
  Notifies the delegate that encoding has finished.

- `archiver:willEncodeObject:` (page 797) *delegate method*
  Informs the delegate that *`object`* is about to be encoded.

- `archiverWillFinish:` (page 798) *delegate method*
  Notifies the delegate that encoding is about to finish.

- `archiver:willReplaceObject:withObject:` (page 798) *delegate method*
  Informs the delegate that one given object is being substituted for another given object.

- `encodeBool:forKey:` (page 790)
  Encodes a given Boolean value and associates it with a given key.

- `encodeBytes:length:forKey:` (page 791)
  Encodes a given number of bytes from a given C array of bytes and associates them with the a given key.

- `encodeConditionalObject:forKey:` (page 791)
  Encodes a reference to a given object and associates it with a given key only if it has been unconditionally encoded elsewhere in the archive with `encodeObject:forKey:` (page 794).

- `encodeDouble:forKey:` (page 792)
  Encodes a given `double` value and associates it with a given key.
- `encodeFloat:forKey:` (page 792)
  Encodes a given `float` value and associates it with a given key.
- `encodeInt:forKey:` (page 793)
  Encodes a given `int` value and associates it with a given key.
- `encodeInt32:forKey:` (page 792)
  Encodes a given 32-bit integer value and associates it with a given key.
- `encodeInt64:forKey:` (page 793)
  Encodes a given 64-bit integer value and associates it with a given key.
- `encodeObject:forKey:` (page 794)
  Encodes a given object and associates it with a given key.

Managing Delegates

- `delegate` (page 790)
  Returns the receiver's delegate.
- `setDelegate:` (page 796)
  Sets the delegate for the receiver.

Managing Classes and Class Names

+ `setClassName:forClass:` (page 789)
  Adds a class translation mapping to `NSKeyedArchiver` whereby instances of of a given class are encoded with a given class name instead of their real class names.
+ `classNameForClass:` (page 788)
  Returns the class name with which `NSKeyedArchiver` encodes instances of a given class.
- `setClassName:forClass:` (page 795)
  Adds a class translation mapping to the receiver whereby instances of of a given class are encoded with a given class name instead of their real class names.
- `classNameForClass:` (page 789)
  Returns the class name with which the receiver encodes instances of a given class.

Class Methods

archivedDataWithRootObject:

Returns an `NSData` object containing the encoded form of the object graph whose root object is given.

```
+ (NSData *)archivedDataWithRootObject:(id)rootObject
```

Parameters

rootObject

The root of the object graph to archive.

Return Value

An `NSData` object containing the encoded form of the object graph whose root object is *rootObject*. The format of the archive is `NSPropertyListBinaryFormat_v1_0`.

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

CoreRecipes

CustomAtomicStoreSubclass

iSpend

QTQuartzPlayer

Squiggles

Declared In

`NSKeyedArchiver.h`

archiveRootObject:toFile:

Archives an object graph rooted at a given object by encoding it into a data object then atomically writes the resulting data object to a file at a given path, and returns a Boolean value that indicates whether the operation was successful.

```
+ (BOOL)archiveRootObject:(id)rootObject  toFile:(NSString *)path
```

Parameters

rootObject

The root of the object graph to archive.

path

The path of the file in which to write the archive.

Return Value

`YES` if the operation was successful, otherwise `NO`.

Discussion

The format of the archive is `NSPropertyListBinaryFormat_v1_0`.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`NSKeyedArchiver.h`

classNameForClass:

Returns the class name with which `NSKeyedArchiver` encodes instances of a given class.

```
+ (NSString *)classNameForClass:(Class)cls
```

Parameters

cls

The class for which to determine the translation mapping.

Return Value

The class name with which `NSKeyedArchiver` encodes instances of *cls*. Returns `nil` if `NSKeyedArchiver` does not have a translation mapping for *cls*.

Availability

Available in Mac OS X v10.2 and later.

See Also

`+ setClassName:forClass:` (page 789)

`- classNameForClass:` (page 789)

Declared In

`NSKeyedArchiver.h`

setClassName:forClass:

Adds a class translation mapping to `NSKeyedArchiver` whereby instances of of a given class are encoded with a given class name instead of their real class names.

```
+ (void)setClassName:(NSString *)codedName  forClass:(Class)cls
```

Parameters

codedName

The name of the class that `NSKeyedArchiver` uses in place of `cls`.

cls

The class for which to set up a translation mapping.

Discussion

When encoding, the class's translation mapping is used only if no translation is found first in an instance's separate translation map.

Availability

Available in Mac OS X v10.2 and later.

See Also

`+ classNameForClass:` (page 788)

`- setClassName:forClass:` (page 795)

Declared In

`NSKeyedArchiver.h`

Instance Methods

classNameForClass:

Returns the class name with which the receiver encodes instances of a given class.

```
- (NSString *)classNameForClass:(Class)cls
```

Parameters

cls

The class for which to determine the translation mapping.

Return Value

The class name with which the receiver encodes instances of *cls*. Returns `nil` if the receiver does not have a translation mapping for *cls*. The class's separate translation map is not searched.

Availability

Available in Mac OS X v10.2 and later.

See Also

`- setClassName:forClass:` (page 795)

`+ classNameForClass:` (page 788)

Declared In

`NSKeyedArchiver.h`

delegate

Returns the receiver's delegate.

```
- (id)delegate
```

Return Value

The receiver's delegate.

Availability

Available in Mac OS X v10.2 and later.

See Also

`- setDelegate:` (page 796)

Declared In

`NSKeyedArchiver.h`

encodeBool:forKey:

Encodes a given Boolean value and associates it with a given key.

```
- (void)encodeBool:(BOOL)boolv  forKey:(NSString *)key
```

Parameters

boolv

The value to encode.

key

The key with which to associate *boolv*. This value must not be `nil`.

Availability

Available in Mac OS X v10.2 and later.

See Also
`decodeBoolForKey:` (page 807) (`NSKeyedUnarchiver`)

Declared In
`NSKeyedArchiver.h`

encodeBytes:length:forKey:

Encodes a given number of bytes from a given C array of bytes and associates them with the a given key.

```
- (void)encodeBytes:(const uint8_t *)bytesp  length:(NSUInteger)lenv
    forKey:(NSString *)key
```

Parameters

bytesp
A C array of bytes to encode.

lenv
The number of bytes from *bytesp* to encode.

key
The key with which to associate the encoded value. This value must not be `nil`.

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

See Also
`decodeBytesForKey:returnedLength:` (page 807) (`NSKeyedUnarchiver`)

Declared In
`NSKeyedArchiver.h`

encodeConditionalObject:forKey:

Encodes a reference to a given object and associates it with a given key only if it has been unconditionally encoded elsewhere in the archive with `encodeObject:forKey:` (page 794).

```
- (void)encodeConditionalObject:(id)objv  forKey:(NSString *)key
```

Parameters

objv
The object to encode.

key
The key with which to associate the encoded value. This value must not be `nil`.

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

Declared In
`NSKeyedArchiver.h`

encodeDouble:forKey:

Encodes a given `double` value and associates it with a given key.

```
- (void)encodeDouble:(double)realv  forKey:(NSString *)key
```

Parameters

realv

The value to encode.

key

The key with which to associate *realv*. This value must not be `nil`.

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

See Also
`decodeDoubleForKey:` (page 808) (`NSKeyedUnarchiver`)
`decodeFloatForKey:` (page 808) (`NSKeyedUnarchiver`)

Declared In
`NSKeyedArchiver.h`

encodeFloat:forKey:

Encodes a given `float` value and associates it with a given key.

```
- (void)encodeFloat:(float)realv  forKey:(NSString *)key
```

Parameters

realv

The value to encode.

key

The key with which to associate *realv*. This value must not be `nil`.

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

See Also
`decodeFloatForKey:` (page 808) (`NSKeyedUnarchiver`)
`decodeDoubleForKey:` (page 808) (`NSKeyedUnarchiver`)

Declared In
`NSKeyedArchiver.h`

encodeInt32:forKey:

Encodes a given 32-bit integer value and associates it with a given key.

```
- (void)encodeInt32:(int32_t)intv  forKey:(NSString *)key
```

Parameters

intv

The value to encode.

key

The key with which to associate *intv*. This value must not be `nil`.

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

See Also
`decodeInt32ForKey:` (page 809) (`NSKeyedUnarchiver`)

Declared In
`NSKeyedArchiver.h`

encodeInt64:forKey:

Encodes a given 64-bit integer value and associates it with a given key.

```
- (void)encodeInt64:(int64_t)intv  forKey:(NSString *)key
```

Parameters

intv

The value to encode.

key

The key with which to associate *intv*. This value must not be `nil`.

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

See Also
`decodeInt64ForKey:` (page 809) (`NSKeyedUnarchiver`)

Declared In
`NSKeyedArchiver.h`

encodeInt:forKey:

Encodes a given `int` value and associates it with a given key.

```
- (void)encodeInt:(int)intv  forKey:(NSString *)key
```

Parameters

intv

The value to encode.

key

The key with which to associate *intv*. This value must not be `nil`.

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

See Also
`decodeIntForKey:` (page 809) (`NSKeyedUnarchiver`)

Declared In
`NSKeyedArchiver.h`

encodeObject:forKey:

Encodes a given object and associates it with a given key.

```
- (void)encodeObject:(id)objv  forKey:(NSString *)key
```

Parameters

objv
The value to encode. This value may be `nil`.

key
The key with which to associate *objv*. This value must not be `nil`.

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

See Also
`decodeObjectForKey:` (page 810) (`NSKeyedUnarchiver`)

Declared In
`NSKeyedArchiver.h`

finishEncoding

Instructs the receiver to construct the final data stream.

```
- (void)finishEncoding
```

Discussion
No more values can be encoded after this method is called. You must call this method when finished.

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

See Also
`- initForWritingWithMutableData:` (page 794)

Declared In
`NSKeyedArchiver.h`

initForWritingWithMutableData:

Returns the receiver, initialized for encoding an archive into a given a mutable-data object.

```
- (id)initForWritingWithMutableData:(NSMutableData *)data
```

Parameters

data

The mutable-data object into which the archive is written.

Return Value

The receiver, initialized for encoding an archive into *data*.

Discussion

When you finish encoding data, you must invoke `finishEncoding` (page 794) at which point *data* is filled. The format of the receiver is `NSPropertyListBinaryFormat_v1_0`.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`NSKeyedArchiver.h`

outputFormat

Returns the format in which the receiver encodes its data.

```
- (NSPropertyListFormat)outputFormat
```

Return Value

The format in which the receiver encodes its data. The available formats are `NSPropertyListXMLFormat_v1_0` and `NSPropertyListBinaryFormat_v1_0`.

Availability

Available in Mac OS X v10.2 and later.

See Also

`- setOutputFormat:` (page 796)

Declared In

`NSKeyedArchiver.h`

setClassName:forClass:

Adds a class translation mapping to the receiver whereby instances of of a given class are encoded with a given class name instead of their real class names.

```
- (void)setClassName:(NSString *)codedName  forClass:(Class)cls
```

Parameters

codedName

The name of the class that the receiver uses uses in place of `cls`.

cls

The class for which to set up a translation mapping.

Discussion

When encoding, the receiver's translation map overrides any translation that may also be present in the class's map.

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

See Also
- `classNameForClass:` (page 789)
+ `setClassName:forClass:` (page 789)

Declared In
`NSKeyedArchiver.h`

setDelegate:

Sets the delegate for the receiver.

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

Parameters

delegate
The delegate for the receiver.

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

See Also
- `delegate` (page 790)

Declared In
`NSKeyedArchiver.h`

setOutputFormat:

Sets the format in which the receiver encodes its data.

```
- (void)setOutputFormat:(NSPropertyListFormat)format
```

Parameters

format
The format in which the receiver encodes its data. *format* can be `NSPropertyListXMLFormat_v1_0` or `NSPropertyListBinaryFormat_v1_0`.

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

See Also
- `outputFormat` (page 795)

Declared In
`NSKeyedArchiver.h`

Delegate Methods

archiver:didEncodeObject:

Informs the delegate that a given object has been encoded.

```
- (void)archiver:(NSKeyedArchiver *)archiver  didEncodeObject:(id)object
```

Parameters

archiver

The archiver that sent the message.

object

The object that has been encoded. *object* may be `nil`.

Discussion

The delegate might restore some state it had modified previously, or use this opportunity to keep track of the objects that are encoded.

This method is not called for conditional objects until they are actually encoded (if ever).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`NSKeyedArchiver.h`

archiver:willEncodeObject:

Informs the delegate that *object* is about to be encoded.

```
- (id)archiver:(NSKeyedArchiver *)archiver  willEncodeObject:(id)object
```

Parameters

archiver

The archiver that sent the message.

object

The object that is about to be encoded. This value is never `nil`.

Return Value

Either *object* or a different object to be encoded in its stead. The delegate can also modify the coder state. If the delegate returns `nil`, `nil` is encoded.

Discussion

This method is called after the original object may have replaced itself with `replacementObjectForKeyedArchiver:` (page 1191).

This method is called whether or not the object is being encoded conditionally.

This method is not called for an object once a replacement mapping has been set up for that object (either explicitly, or because the object has previously been encoded). This method is also not called when `nil` is about to be encoded.

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

Declared In
NSKeyedArchiver.h

archiver:willReplaceObject:withObject:

Informs the delegate that one given object is being substituted for another given object.

```
- (void)archiver:(NSKeyedArchiver *)archiver  willReplaceObject:(id)object
    withObject:(id)newObject
```

Parameters

archiver
The archiver that sent the message.

object
The object being replaced in the archive.

newObject
The object replacing *object* in the archive.

Discussion
This method is called even when the delegate itself is doing, or has done, the substitution. The delegate may use this method if it is keeping track of the encoded or decoded objects.

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

Declared In
NSKeyedArchiver.h

archiverDidFinish:

Notifies the delegate that encoding has finished.

```
- (void)archiverDidFinish:(NSKeyedArchiver *)archiver
```

Parameters

archiver
The archiver that sent the message.

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

Declared In
NSKeyedArchiver.h

archiverWillFinish:

Notifies the delegate that encoding is about to finish.

```
- (void)archiverWillFinish:(NSKeyedArchiver *)archiver
```

Parameters

archiver

The archiver that sent the message.

Availability

Available in Mac OS X v10.2 and later.

Declared In

NSKeyedArchiver.h

Constants

Keyed Archiving Exception Names

Names of exceptions that are raised by `NSKeyedArchiver` if there is a problem creating an archive.

```
extern NSString *NSInvalidArchiveOperationException;
```

Constants

`NSInvalidArchiveOperationException`

The name of the exception raised by `NSKeyedArchiver` if there is a problem creating an archive.

Available in Mac OS X v10.2 and later.

Declared in `NSKeyedArchiver.h`.

Declared In

NSKeyedArchiver.h

NSKeyedUnarchiver Class Reference

Inherits from	NSCoder : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.2 and later.
Declared in	Foundation/NSKeyedArchiver.h
Companion guide	Archives and Serializations Programming Guide for Cocoa
Related sample code	CoreRecipes CustomAtomicStoreSubclass iSpend QTQuartzPlayer Squiggles

Overview

`NSKeyedUnarchiver`, a concrete subclass of `NSCoder`, defines methods for decoding a set of named objects (and scalar values) from a keyed archive. Such archives are produced by instances of the `NSKeyedArchiver` class.

A keyed archive is encoded as a hierarchy of objects. Each object in the hierarchy serves as a namespace into which other objects are encoded. The objects available for decoding are restricted to those that were encoded within the immediate scope of a particular object. Objects encoded elsewhere in the hierarchy, whether higher than, lower than, or parallel to this particular object, are not accessible. In this way, the keys used by a particular object to encode its instance variables need to be unique only within the scope of that object.

If you invoke one of the `decode...` methods of this class using a key that does not exist in the archive, a non-positive value is returned. This value varies by decoded type. For example, if a key does not exist in an archive, `decodeBoolForKey:` (page 807) returns `NO`, `decodeIntForKey:` (page 809) returns `0`, and `decodeObjectForKey:` (page 810) returns nil.

`NSKeyedUnarchiver` supports limited type coercion. A value encoded as any type of integer, whether a standard `int` or an explicit 32-bit or 64-bit integer, can be decoded using any of the integer decode methods. Likewise, a value encoded as a `float` or `double` can be decoded as either a `float` or a `double` value. If an encoded value is too large to fit within the coerced type, the decoding method raises an `NSRangeException`. Further, when trying to coerce a value to an incompatible type, for example decoding an `int` as a `float`, the decoding method raises an `NSInvalidUnarchiveOperationException`.

Tasks

Initializing a Keyed Unarchiver

- `initForReadingWithData:` (page 811)
  Initializes the receiver for decoding an archive previously encoded by `NSKeyedArchiver`.

Unarchiving Data

+ `unarchiveObjectWithData:` (page 804)
  Decodes and returns the object graph previously encoded by `NSKeyedArchiver` and stored in a given `NSData` object.

+ `unarchiveObjectWithFile:` (page 805)
  Decodes and returns the object graph previously encoded by `NSKeyedArchiver` written to the file at a given path.

Decoding Data

- `containsValueForKey:` (page 806)
  Returns a Boolean value that indicates whether the archive contains a value for a given key within the current decoding scope.

- `decodeBoolForKey:` (page 807)
  Decodes a Boolean value associated with a given key.

- `decodeBytesForKey:returnedLength:` (page 807)
  Decodes a stream of bytes associated with a given key.

- `decodeDoubleForKey:` (page 808)
  Decodes a double-precision floating-point value associated with a given key.

- `decodeFloatForKey:` (page 808)
  Decodes a single-precision floating-point value associated with a given key.

- `decodeIntForKey:` (page 809)
  Decodes an integer value associated with a given key.

- `decodeInt32ForKey:` (page 809)
  Decodes a 32-bit integer value associated with a given key.

- `decodeInt64ForKey:` (page 809)
  Decodes a 64-bit integer value associated with a given key.

- `decodeObjectForKey:` (page 810)
  Decodes and returns an object associated with a given key.

- `finishDecoding` (page 811)
  Tells the receiver that you are finished decoding objects.

Managing the Delegate

- `delegate` (page 810)
  Returns the receiver's delegate.
- `setDelegate:` (page 812)
  Sets the receiver's delegate.

Managing Class Names

+ `setClass:forClassName:` (page 804)
  Adds a class translation mapping to `NSKeyedUnarchiver` whereby objects encoded with a given class name are decoded as instances of a given class instead.
+ `classForClassName:` (page 803)
  Returns the class from which `NSKeyedUnarchiver` instantiates an encoded object with a given class name.
- `setClass:forClassName:` (page 812)
  Adds a class translation mapping to the receiver whereby objects encoded with a given class name are decoded as instances of a given class instead.
- `classForClassName:` (page 806)
  Returns the class from which the receiver instantiates an encoded object with a given class name.

Decoding Objects

- `unarchiver:cannotDecodeObjectOfClassName:originalClasses:` (page 812) *delegate method*
  Informs the delegate that the class with a given name is not available during decoding.
- `unarchiver:didDecodeObject:` (page 813) *delegate method*
  Informs the delegate that a given object has been decoded.
- `unarchiver:willReplaceObject:withObject:` (page 814) *delegate method*
  Informs the delegate that one object is being substituted for another.

Finishing Decoding

- `unarchiverDidFinish:` (page 814) *delegate method*
  Notifies the delegate that decoding has finished.
- `unarchiverWillFinish:` (page 814) *delegate method*
  Notifies the delegate that decoding is about to finish.

Class Methods

classForClassName:

Returns the class from which `NSKeyedUnarchiver` instantiates an encoded object with a given class name.

```
+ (Class)classForClassName:(NSString *)codedName
```

Parameters

codedName

The ostensible name of a class in an archive.

Return Value

The class from which `NSKeyedUnarchiver` instantiates an object encoded with the class name *codedName*. Returns `nil` if `NSKeyedUnarchiver` does not have a translation mapping for *codedName*.

Availability

Available in Mac OS X v10.2 and later.

See Also

`+ setClass:forClassName:` (page 804)

`- classForClassName:` (page 806)

Declared In

`NSKeyedArchiver.h`

setClass:forClassName:

Adds a class translation mapping to `NSKeyedUnarchiver` whereby objects encoded with a given class name are decoded as instances of a given class instead.

```
+ (void)setClass:(Class)cls forClassName:(NSString *)codedName
```

Parameters

cls

The class with which to replace instances of the class named *codedName*.

codedName

The ostensible name of a class in an archive.

Discussion

When decoding, the class's translation mapping is used only if no translation is found first in an instance's separate translation map.

Availability

Available in Mac OS X v10.2 and later.

See Also

`+ classForClassName:` (page 803)

`- setClass:forClassName:` (page 812)

Declared In

`NSKeyedArchiver.h`

unarchiveObjectWithData:

Decodes and returns the object graph previously encoded by `NSKeyedArchiver` and stored in a given `NSData` object.

```
+ (id)unarchiveObjectWithData:(NSData *)data
```

Parameters

data

An object graph previously encoded by `NSKeyedArchiver`.

Return Value

The object graph previously encoded by `NSKeyedArchiver` and stored in *data*.

Discussion

This method raises an NSInvalidArchiveOperationException (page 799) if *data* is not a valid archive.

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

CoreRecipes

CustomAtomicStoreSubclass

iSpend

QTQuartzPlayer

Squiggles

Declared In

`NSKeyedArchiver.h`

unarchiveObjectWithFile:

Decodes and returns the object graph previously encoded by `NSKeyedArchiver` written to the file at a given path.

```
+ (id)unarchiveObjectWithFile:(NSString *)path
```

Parameters

path

A path to a file that contains an object graph previously encoded by `NSKeyedArchiver`.

Return Value

The object graph previously encoded by `NSKeyedArchiver` written to the file *path*. Returns `nil` if there is no file at *path*.

Discussion

This method raises an `NSInvalidArgumentException` (page 2307) if the file at *path* does not contain a valid archive.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`NSKeyedArchiver.h`

Instance Methods

classForClassName:

Returns the class from which the receiver instantiates an encoded object with a given class name.

```
- (Class)classForClassName:(NSString *)codedName
```

Parameters

codedName

The name of a class.

Return Value

The class from which the receiver instantiates an encoded object with the class name *codedName*. Returns `nil` if the receiver does not have a translation mapping for *codedName*.

Discussion

The class's separate translation map is not searched.

Availability

Available in Mac OS X v10.2 and later.

See Also

`- setClass:forClassName:` (page 812)

`+ classForClassName:` (page 803)

Declared In

`NSKeyedArchiver.h`

containsValueForKey:

Returns a Boolean value that indicates whether the archive contains a value for a given key within the current decoding scope.

```
- (BOOL)containsValueForKey:(NSString *)key
```

Parameters

key

A key in the archive within the current decoding scope. *key* must not be `nil`.

Return Value

`YES` if the archive contains a value for *key* within the current decoding scope, otherwise `NO`.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`NSKeyedArchiver.h`

decodeBoolForKey:

Decodes a Boolean value associated with a given key.

```
- (BOOL)decodeBoolForKey:(NSString *)key
```

Parameters

key

A key in the archive within the current decoding scope. *key* must not be `nil`.

Return Value

The Boolean value associated with the key *key*. Returns `NO` if *key* does not exist.

Availability

Available in Mac OS X v10.2 and later.

See Also

`- encodeBool:forKey:` (page 790) (`NSKeyedArchiver`)

Declared In

`NSKeyedArchiver.h`

decodeBytesForKey:returnedLength:

Decodes a stream of bytes associated with a given key.

```
- (const uint8_t *)decodeBytesForKey:(NSString *)key returnedLength:(NSUInteger
    *)lengthp
```

Parameters

key

A key in the archive within the current decoding scope. *key* must not be `nil`.

lengthp

Upon return, contains the number of bytes returned.

Return Value

The stream of bytes associated with the key *key*. Returns `NULL` if *key* does not exist.

Discussion

The returned value is a pointer to a temporary buffer owned by the receiver. The buffer goes away with the unarchiver, not the containing autorelease pool. You must copy the bytes into your own buffer if you need the data to persist beyond the life of the receiver.

Availability

Available in Mac OS X v10.2 and later.

See Also

`- encodeBytes:length:forKey:` (page 791) (`NSKeyedArchiver`)

Declared In

`NSKeyedArchiver.h`

decodeDoubleForKey:

Decodes a double-precision floating-point value associated with a given key.

```
- (double)decodeDoubleForKey:(NSString *)key
```

Parameters

key

A key in the archive within the current decoding scope. *key* must not be `nil`.

Return Value

The double-precision floating-point value associated with the key *key*. Returns `0.0` if *key* does not exist.

Discussion

If the archived value was encoded as single-precision, the type is coerced.

Availability

Available in Mac OS X v10.2 and later.

See Also

- `encodeDouble:forKey:` (page 792) (`NSKeyedArchiver`)
- `encodeFloat:forKey:` (page 792) (`NSKeyedArchiver`)

Declared In

`NSKeyedArchiver.h`

decodeFloatForKey:

Decodes a single-precision floating-point value associated with a given key.

```
- (float)decodeFloatForKey:(NSString *)key
```

Parameters

key

A key in the archive within the current decoding scope. *key* must not be `nil`.

Return Value

The single-precision floating-point value associated with the key *key*. Returns `0.0` if *key* does not exist.

Discussion

If the archived value was encoded as double precision, the type is coerced, loosing precision. If the archived value is too large for single precision, the method raises an `NSRangeException`.

Availability

Available in Mac OS X v10.2 and later.

See Also

- `encodeFloat:forKey:` (page 792) (`NSKeyedArchiver`)
- `encodeDouble:forKey:` (page 792) (`NSKeyedArchiver`)

Declared In

`NSKeyedArchiver.h`

decodeInt32ForKey:

Decodes a 32-bit integer value associated with a given key.

```
- (int32_t)decodeInt32ForKey:(NSString *)key
```

Parameters

key

A key in the archive within the current decoding scope. *key* must not be `nil`.

Return Value

The 32-bit integer value associated with the key *key*. Returns `0` if *key* does not exist.

Discussion

If the archived value was encoded with a different size but is still an integer, the type is coerced. If the archived value is too large to fit into a 32-bit integer, the method raises an `NSRangeException`.

Availability

Available in Mac OS X v10.2 and later.

See Also

`- encodeInt32:forKey:` (page 792) (`NSKeyedArchiver`)

Declared In

`NSKeyedArchiver.h`

decodeInt64ForKey:

Decodes a 64-bit integer value associated with a given key.

```
- (int64_t)decodeInt64ForKey:(NSString *)key
```

Parameters

key

A key in the archive within the current decoding scope. *key* must not be `nil`.

Return Value

The 64-bit integer value associated with the key *key*. Returns `0` if *key* does not exist.

Discussion

If the archived value was encoded with a different size but is still an integer, the type is coerced.

Availability

Available in Mac OS X v10.2 and later.

See Also

`- encodeInt64:forKey:` (page 793) (`NSKeyedArchiver`)

Declared In

`NSKeyedArchiver.h`

decodeIntForKey:

Decodes an integer value associated with a given key.

```
- (int)decodeIntForKey:(NSString *)key
```

Parameters

key

A key in the archive within the current decoding scope. *key* must not be `nil`.

Return Value

The integer value associated with the key `key`. Returns `0` if *key* does not exist.

Discussion

If the archived value was encoded with a different size but is still an integer, the type is coerced. If the archived value is too large to fit into the default size for an integer, the method raises an `NSRangeException`.

Availability

Available in Mac OS X v10.2 and later.

See Also

`- encodeInt:forKey:` (page 793) (`NSKeyedArchiver`)

Declared In

`NSKeyedArchiver.h`

decodeObjectForKey:

Decodes and returns an object associated with a given key.

```
- (id)decodeObjectForKey:(NSString *)key
```

Parameters

key

A key in the archive within the current decoding scope. *key* must not be `nil`.

Return Value

The object associated with the key *key*. Returns `nil` if *key* does not exist, or if the value for *key* is `nil`.

Availability

Available in Mac OS X v10.2 and later.

See Also

`- encodeObject:forKey:` (page 794) (`NSKeyedArchiver`)

Declared In

`NSKeyedArchiver.h`

delegate

Returns the receiver's delegate.

```
- (id)delegate
```

Return Value

The receiver's delegate.

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

See Also
`- setDelegate:` (page 812)

Declared In
`NSKeyedArchiver.h`

finishDecoding

Tells the receiver that you are finished decoding objects.

```
- (void)finishDecoding
```

Discussion
Invoking this method allows the receiver to notify its delegate and to perform any final operations on the archive. Once this method is invoked, the receiver cannot decode any further values.

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

Declared In
`NSKeyedArchiver.h`

initForReadingWithData:

Initializes the receiver for decoding an archive previously encoded by `NSKeyedArchiver`.

```
- (id)initForReadingWithData:(NSData *)data
```

Parameters
data
An archive previously encoded by `NSKeyedArchiver`.

Return Value
An `NSKeyedUnarchiver` object initialized for for decoding *data*.

Discussion
When you finish decoding data, you should invoke `finishDecoding` (page 811).

This method raises an NSInvalidArchiveOperationException (page 799) if *data* is not a valid archive.

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

Declared In
`NSKeyedArchiver.h`

setClass:forClassName:

Adds a class translation mapping to the receiver whereby objects encoded with a given class name are decoded as instances of a given class instead.

```
- (void)setClass:(Class)cls forClassName:(NSString *)codedName
```

Parameters

cls

The class with which to replace instances of the class named *codedName*.

codedName

The ostensible name of a class in an archive.

Discussion

When decoding, the receiver's translation map overrides any translation that may also be present in the class's map (see setClass:forClassName: (page 804)).

Availability

Available in Mac OS X v10.2 and later.

See Also

- classForClassName: (page 806)
+ setClass:forClassName: (page 804)

Declared In

NSKeyedArchiver.h

setDelegate:

Sets the receiver's delegate.

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

Parameters

delegate

The delegate for the receiver.

Availability

Available in Mac OS X v10.2 and later.

See Also

- delegate (page 810)

Declared In

NSKeyedArchiver.h

Delegate Methods

unarchiver:cannotDecodeObjectOfClassName:originalClasses:

Informs the delegate that the class with a given name is not available during decoding.

```
- (Class)unarchiver:(NSKeyedUnarchiver *)unarchiver
    cannotDecodeObjectOfClassName:(NSString *)name originalClasses:(NSArray
    *)classNames
```

Parameters

unarchiver

An unarchiver for which the receiver is the delegate.

name

The name of the class of an object `unarchiver` is trying to decode.

classNames

An array describing the class hierarchy of the encoded object, where the first element is the class name string of the encoded object, the second element is the class name of its immediate superclass, and so on.

Return Value

The class unarchiver should use in place of the class named *name*.

Discussion

The delegate may, for example, load some code to introduce the class to the runtime and return the class, or substitute a different class object. If the delegate returns `nil`, unarchiving aborts and the method raises an `NSInvalidUnarchiveOperationException`.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`NSKeyedArchiver.h`

unarchiver:didDecodeObject:

Informs the delegate that a given object has been decoded.

```
- (id)unarchiver:(NSKeyedUnarchiver *)unarchiver didDecodeObject:(id)object
```

Parameters

unarchiver

An unarchiver for which the receiver is the delegate.

object

The object that has been decoded. *object* may be `nil`.

Return Value

The object to use in place of *object*. The delegate can either return *object* or return a different object to replace the decoded one. If the delegate returns `nil`, `nil` is the result of decoding *object*.

Discussion

This method is called after *object* has been sent `initWithCoder:` (page 2034) and `awakeAfterUsingCoder:` (page 1169).

The delegate may use this method to keep track of the decoded objects.

Availability

Available in Mac OS X v10.2 and later.

Declared In
NSKeyedArchiver.h

unarchiver:willReplaceObject:withObject:

Informs the delegate that one object is being substituted for another.

```
- (void)unarchiver:(NSKeyedUnarchiver *)unarchiver willReplaceObject:(id)object
    withObject:(id)newObject
```

Parameters

unarchiver
An unarchiver for which the receiver is the delegate.

object
An object in the archive.

newObject
The object with which *unarchiver* will replace *object*.

Discussion
This method is called even when the delegate itself is doing, or has done, the substitution with `unarchiver:didDecodeObject:` (page 813).

The delegate may use this method if it is keeping track of the encoded or decoded objects.

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

Declared In
NSKeyedArchiver.h

unarchiverDidFinish:

Notifies the delegate that decoding has finished.

```
- (void)unarchiverDidFinish:(NSKeyedUnarchiver *)unarchiver
```

Parameters

unarchiver
An unarchiver for which the receiver is the delegate.

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

Declared In
NSKeyedArchiver.h

unarchiverWillFinish:

Notifies the delegate that decoding is about to finish.

```
- (void)unarchiverWillFinish:(NSKeyedUnarchiver *)unarchiver
```

Parameters

unarchiver

An unarchiver for which the receiver is the delegate.

Availability

Available in Mac OS X v10.2 and later.

Declared In

NSKeyedArchiver.h

Constants

Keyed Unarchiving Exception Names

Names of exceptions that are raised by `NSKeyedUnarchiver` if there is a problem extracting an archive.

```
extern NSString *NSInvalidUnarchiveOperationException;
```

Constants

`NSInvalidUnarchiveOperationException`

The name of the exception raised by `NSKeyedArchiver` if there is a problem extracting an archive.

Available in Mac OS X v10.2 and later.

Declared in `NSKeyedArchiver.h`.

Declared In

NSKeyedUnarchiver.h

NSLocale Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.4 and later.
Declared in	Foundation/NSLocale.h
Companion guides	Locales Programming Guide Data Formatting Programming Guide for Cocoa
Related sample code	Mountains

Overview

Locales encapsulate information about linguistic, cultural, and technological conventions and standards. Examples of information encapsulated by a locale include the symbol used for the decimal separator in numbers and the way dates are formatted.

Locales are typically used to provide, format, and interpret information about and according to the user's customs and preferences. They are frequently used in conjunction with formatters (see *Data Formatting Programming Guide for Cocoa*). Although you can use many locales, you usually use the one associated with the current user.

`NSLocale` is "toll-free bridged" with its Core Foundation counterpart, `CFLocale`. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSLocale *` parameter, you can pass a `CFLocaleRef`, and in a function where you see a `CFLocaleRef` parameter, you can pass an `NSLocale` instance (you cast one type to the other to suppress compiler warnings). See Interchangeable Data Types for more information on toll-free bridging.

Tasks

Getting and Initializing Locales

- `initWithLocaleIdentifier:` (page 826)
  Initializes the receiver using a given locale identifier.
+ `systemLocale` (page 824)
  Returns the "root", canonical locale, that contains fixed "backstop" settings that provide values for otherwise undefined keys.
+ `currentLocale` (page 821)
  Returns the logical locale for the current user.
+ `autoupdatingCurrentLocale` (page 819)
  Returns the current logical locale for the current user.

Getting Information About a Locale

- `displayNameForKey:value:` (page 825)
  Returns the display name for the given value.
- `localeIdentifier` (page 826)
  Returns the identifier for the receiver.
- `objectForKey:` (page 827)
  Returns the object corresponding to the specified key.

Getting System Locale Information

+ `availableLocaleIdentifiers` (page 819)
  Returns an array of `NSString` objects, each of which identifies a locale available on the system.
+ `ISOCountryCodes` (page 822)
  Returns an array of `NSString` objects that represents all known legal country codes.
+ `ISOCurrencyCodes` (page 822)
  Returns an array of `NSString` objects that represents all known legal ISO currency codes.
+ `ISOLanguageCodes` (page 823)
  Returns an array of `NSString` objects that represents all known legal ISO language codes.
+ `commonISOCurrencyCodes` (page 820)
  Returns an array of common ISO currency codes

Converting Between Identifiers

+ `canonicalLocaleIdentifierFromString:` (page 820)
  Returns the canonical identifier for a given locale identification string.
+ `componentsFromLocaleIdentifier:` (page 821)
  Returns a dictionary that is the result of parsing a locale ID.

`+ localeIdentifierFromComponents:` (page 823)
Returns a locale identifier from the components specified in a given dictionary.

Getting Preferred Languages

`+ preferredLanguages` (page 824)
Returns the user's language preference order as an array of strings.

Class Methods

autoupdatingCurrentLocale

Returns the current logical locale for the current user.

```
+ (id)autoupdatingCurrentLocale
```

Return Value
The current logical locale for the current user. The locale is formed from the settings for the current user's chosen system locale overlaid with any custom settings the user has specified in System Preferences.

The object always reflects the current state of the current user's locale settings.

Discussion
Settings you get from this locale do change as the user's settings change (contrast with `currentLocale` (page 821)).

Note that if you cache values based on the locale or related information, those caches will of course not be automatically updated by the updating of the locale object. You can recompute caches upon receipt of the notification (`NSCurrentLocaleDidChangeNotification`) that gets sent out for locale changes (see *Notification Programming Topics for Cocoa* to learn how to register for and receive notifications).

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

See Also
`+ systemLocale` (page 824)
`+ currentLocale` (page 821)

Related Sample Code
Mountains

Declared In
`NSLocale.h`

availableLocaleIdentifiers

Returns an array of `NSString` objects, each of which identifies a locale available on the system.

```
+ (NSArray *)availableLocaleIdentifiers
```

Return Value
An array of `NSString` objects, each of which identifies a locale available on the system.

Availability
Available in Mac OS X v10.4 and later.

See Also
`+ ISOLanguageCodes` (page 823)
`+ ISOCountryCodes` (page 822)
`+ ISOCurrencyCodes` (page 822)
`+ commonISOCurrencyCodes` (page 820)

Declared In
`NSLocale.h`

canonicalLocaleIdentifierFromString:

Returns the canonical identifier for a given locale identification string.

```
+ (NSString *)canonicalLocaleIdentifierFromString:(NSString *)string
```

Parameters
string
A locale identification string.

Return Value
The canonical identifier for an the locale identified by *string*.

Availability
Available in Mac OS X v10.4 and later.

See Also
`+ componentsFromLocaleIdentifier:` (page 821)
`+ localeIdentifierFromComponents:` (page 823)

Related Sample Code
Mountains

Declared In
`NSLocale.h`

commonISOCurrencyCodes

Returns an array of common ISO currency codes

```
+ (NSArray *)commonISOCurrencyCodes
```

Return Value
An array of `NSString` objects that represents common ISO currency codes.

Discussion
Common codes may include, for example, AED, AUD, BZD, DKK, EUR, GBP, JPY, KES, MXN, OMR, STD, USD, XCD, and ZWD.

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

See Also
+ availableLocaleIdentifiers (page 819)
+ ISOCountryCodes (page 822)
+ ISOCurrencyCodes (page 822)

Declared In
NSLocale.h

componentsFromLocaleIdentifier:

Returns a dictionary that is the result of parsing a locale ID.

```
+ (NSDictionary *)componentsFromLocaleIdentifier:(NSString *)string
```

Parameters

string
A locale ID, consisting of language, script, country, variant, and keyword/value pairs, for example, "en_US@calendar=japanese".

Return Value
A dictionary that is the result of parsing *string* as a locale ID. The keys are the constant NSString constants corresponding to the locale ID components, and the values correspond to constants where available. For the complete set of dictionary keys, see "Constants" (page 827).

Discussion
For example: the locale ID "en_US@calendar=japanese" yields a dictionary with three entries: NSLocaleLanguageCode=en, NSLocaleCountryCode=US, and NSLocaleCalendar=NSJapaneseCalendar.

Availability
Available in Mac OS X v10.4 and later.

See Also
+ localeIdentifierFromComponents: (page 823)
+ canonicalLocaleIdentifierFromString: (page 820)

Declared In
NSLocale.h

currentLocale

Returns the logical locale for the current user.

```
+ (id)currentLocale
```

Return Value
The logical locale for the current user. The locale is formed from the settings for the current user's chosen system locale overlaid with any custom settings the user has specified in System Preferences.

This method may return a retained cached object.

Discussion
Settings you get from this locale do not change as System Preferences are changed so that your operations are consistent. Typically you perform some operations on the returned object and then allow it to be disposed of. Moreover, since the returned object may be cached, you do not need to hold on to it indefinitely. Contrast with `autoupdatingCurrentLocale` (page 819).

Availability
Available in Mac OS X v10.4 and later.

See Also
`+ systemLocale` (page 824)
`+ autoupdatingCurrentLocale` (page 819)

Declared In
`NSLocale.h`

ISOCountryCodes

Returns an array of `NSString` objects that represents all known legal country codes.

```
+ (NSArray *)ISOCountryCodes
```

Return Value
An array of `NSString` objects that represents all known legal country codes.

Discussion
Note that many of country codes do not have any supporting locale data in Mac OS X.

Availability
Available in Mac OS X v10.4 and later.

See Also
`+ availableLocaleIdentifiers` (page 819)
`+ ISOLanguageCodes` (page 823)
`+ ISOCurrencyCodes` (page 822)
`+ commonISOCurrencyCodes` (page 820)

Declared In
`NSLocale.h`

ISOCurrencyCodes

Returns an array of `NSString` objects that represents all known legal ISO currency codes.

```
+ (NSArray *)ISOCurrencyCodes
```

Return Value
An array of `NSString` objects that represents all known legal ISO currency codes.

Discussion
Note that some of the currency codes may not have any supporting locale data in Mac OS X.

Availability
Available in Mac OS X v10.4 and later.

See Also
+ availableLocaleIdentifiers (page 819)
+ ISOCountryCodes (page 822)
+ ISOLanguageCodes (page 823)
+ commonISOCurrencyCodes (page 820)

Declared In
NSLocale.h

ISOLanguageCodes

Returns an array of NSString objects that represents all known legal ISO language codes.

```
+ (NSArray *)ISOLanguageCodes
```

Return Value
An array of NSString objects that represents all known legal ISO language codes.

Discussion
Note that many of the language codes will not have any supporting locale data in Mac OS X.

Availability
Available in Mac OS X v10.4 and later.

See Also
+ availableLocaleIdentifiers (page 819)
+ ISOCountryCodes (page 822)
+ ISOCurrencyCodes (page 822)
+ commonISOCurrencyCodes (page 820)

Declared In
NSLocale.h

localeIdentifierFromComponents:

Returns a locale identifier from the components specified in a given dictionary.

```
+ (NSString *)localeIdentifierFromComponents:(NSDictionary *)dict
```

Parameters

dict

A dictionary containing components that specify a locale. For valid dictionary keys, see "Constants" (page 827).

Return Value
A locale identifier created from the components specified in *dict*.

Discussion
This reverses the actions of `componentsFromLocaleIdentifier:` (page 821), so for example the dictionary `{NSLocaleLanguageCode="en", NSLocaleCountryCode="US", NSLocaleCalendar=NSJapaneseCalendar}` becomes `"en_US@calendar=japanese"`.

Availability
Available in Mac OS X v10.4 and later.

See Also
`+ componentsFromLocaleIdentifier:` (page 821)
`+ canonicalLocaleIdentifierFromString:` (page 820)
`+ ISOLanguageCodes` (page 823)

Declared In
`NSLocale.h`

preferredLanguages

Returns the user's language preference order as an array of strings.

```
+ (NSArray *)preferredLanguages
```

Return Value
The user's language preference order as an array of `NSString` objects, each of which is a canonicalized IETF BCP 47 language identifier.

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

Related Sample Code
Mountains

Declared In
`NSLocale.h`

systemLocale

Returns the “root”, canonical locale, that contains fixed “backstop” settings that provide values for otherwise undefined keys.

```
+ (id)systemLocale
```

Return Value
The “root”, canonical locale, that contains fixed “backstop” settings that provide values for otherwise undefined keys.

Availability
Available in Mac OS X v10.4 and later.

See Also
`+ autoupdatingCurrentLocale` (page 819)
`+ autoupdatingCurrentLocale` (page 819)

Declared In
NSLocale.h

Instance Methods

displayNameForKey:value:

Returns the display name for the given value.

```
- (NSString *)displayNameForKey:(id)key value:(id)value
```

Parameters

key
Specifies which of the locale property keys *value* is (see "Constants" (page 827)),

value
A value for *key*.

Return Value
The display name for *value*.

Discussion
Not all locale property keys have values with display name values.

You can use the NSLocaleIdentifier key to get the name of a locale in the language of another locale, as illustrated in the following examples. The first uses the fr_FR locale.

```
NSLocale *frLocale = [[[NSLocale alloc] initWithLocaleIdentifier:@"fr_FR"]
autorelease];
NSString *displayNameString = [frLocale displayNameForKey:NSLocaleIdentifier
value:@"fr_FR"];
NSLog(@"displayNameString fr_FR: %@", displayNameString);
displayNameString = [frLocale displayNameForKey:NSLocaleIdentifier
value:@"en_US"];
NSLog(@"displayNameString en_US: %@", displayNameString);
```

returns

```
displayNameString fr_FR: français (France)
displayNameString en_US: anglais (États-Unis)
```

The following example uses the en_GB locale.

```
NSLocale *gbLocale = [[[NSLocale alloc] initWithLocaleIdentifier:@"en_GB"]
autorelease];
displayNameString = [gbLocale displayNameForKey:NSLocaleIdentifier
value:@"fr_FR"];
NSLog(@"displayNameString fr_FR: %@", displayNameString);
displayNameString = [gbLocale displayNameForKey:NSLocaleIdentifier
value:@"en_US"];
NSLog(@"displayNameString en_US: %@", displayNameString);
```

returns

```
displayNameString fr_FR: French (France)
```

```
displayNameString en_US: English (United States)
```

Availability
Available in Mac OS X v10.4 and later.

See Also
- `localeIdentifier` (page 826)

Declared In
`NSLocale.h`

initWithLocaleIdentifier:

Initializes the receiver using a given locale identifier.

```
- (id)initWithLocaleIdentifier:(NSString *)string
```

Parameters
string
The identifier for the new locale.

Return Value
The initialized locale.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
Mountains

Declared In
`NSLocale.h`

localeIdentifier

Returns the identifier for the receiver.

```
- (NSString *)localeIdentifier
```

Return Value
The identifier for the receiver. This may not be the same string that the locale was created with, since `NSLocale` may canonicalize it.

Discussion
Equivalent to sending `objectForKey:` with key `NSLocaleIdentifier`.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `displayNameForKey:value:` (page 825)

Related Sample Code
Mountains

Declared In
NSLocale.h

objectForKey:

Returns the object corresponding to the specified key.

```
- (id)objectForKey:(id)key
```

Parameters

key

The key for which to return the corresponding value. For valid values of *key*, see "Constants" (page 827).

Return Value
The object corresponding to *key*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- displayNameForKey:value: (page 825)

Declared In
NSLocale.h

Constants

NSLocale Component Keys

The following constants specify keys used to retrieve components of a locale with objectForKey: (page 827).

```
extern NSString * const NSLocaleIdentifier;
extern NSString * const NSLocaleLanguageCode;
extern NSString * const NSLocaleCountryCode;
extern NSString * const NSLocaleScriptCode;
extern NSString * const NSLocaleVariantCode;
extern NSString * const NSLocaleExemplarCharacterSet;
extern NSString * const NSLocaleCalendar;
extern NSString * const NSLocaleCollationIdentifier;
extern NSString * const NSLocaleUsesMetricSystem;
extern NSString * const NSLocaleMeasurementSystem;
extern NSString * const NSLocaleDecimalSeparator;
extern NSString * const NSLocaleGroupingSeparator;
extern NSString * const NSLocaleCurrencySymbol;
extern NSString * const NSLocaleCurrencyCode;
```

Constants

`NSLocaleIdentifier`

The key for the locale identifier.

The corresponding value is an `NSString` object. An example value might be `"es_ES_PREEURO"`.

Available in Mac OS X v10.4 and later.

Declared in `NSLocale.h`.

`NSLocaleLanguageCode`

The key for the locale language code.

The corresponding value is an `NSString` object. An example value might be `"es"`.

Available in Mac OS X v10.4 and later.

Declared in `NSLocale.h`.

`NSLocaleCountryCode`

The key for the locale country code.

The corresponding value is an `NSString` object. An example value might be `"ES"`.

Available in Mac OS X v10.4 and later.

Declared in `NSLocale.h`.

`NSLocaleScriptCode`

The key for the locale script code.

The corresponding value is an `NSString` object.

Available in Mac OS X v10.4 and later.

Declared in `NSLocale.h`.

`NSLocaleVariantCode`

The key for the locale variant code.

The corresponding value is an `NSString` object. An example value might be `"PREEURO"`.

Available in Mac OS X v10.4 and later.

Declared in `NSLocale.h`.

`NSLocaleExemplarCharacterSet`

The key for the exemplar character set for the locale.

The corresponding value is an `NSCharacterSet` object.

Available in Mac OS X v10.4 and later.

Declared in `NSLocale.h`.

`NSLocaleCalendar`
: The key for the calendar associated with the locale.

  The corresponding value is an `NSCalendar` object.

  Available in Mac OS X v10.4 and later.

  Declared in `NSLocale.h`.

`NSLocaleCollationIdentifier`
: The key for the collation associated with the locale.

  The corresponding value is an `NSString` object.

  Available in Mac OS X v10.4 and later.

  Declared in `NSLocale.h`.

`NSLocaleUsesMetricSystem`
: The key for the flag that indicates whether the locale uses the metric system.

  The corresponding value is a Boolean `NSNumber` object. If the value is `NO`, you can typically assume American measurement units (for example, the statute mile).

  Available in Mac OS X v10.4 and later.

  Declared in `NSLocale.h`.

`NSLocaleMeasurementSystem`
: The key for the measurement system associated with the locale.

  The corresponding value is an `NSString` object containing a description of the measurement system used by the locale, for example “Metric” or “U.S.”.

  Available in Mac OS X v10.4 and later.

  Declared in `NSLocale.h`.

`NSLocaleDecimalSeparator`
: The key for the decimal separator associated with the locale.

  The corresponding value is an `NSString` object.

  Available in Mac OS X v10.4 and later.

  Declared in `NSLocale.h`.

`NSLocaleGroupingSeparator`
: The key for the numeric grouping separator associated with the locale.

  The corresponding value is an `NSString` object.

  Available in Mac OS X v10.4 and later.

  Declared in `NSLocale.h`.

`NSLocaleCurrencySymbol`
: The key for the currency symbol associated with the locale.

  The corresponding value is an `NSString` object.

  Available in Mac OS X v10.4 and later.

  Declared in `NSLocale.h`.

`NSLocaleCurrencyCode`
: The key for the currency code associated with the locale.

  The corresponding value is an `NSString` object.

  Available in Mac OS X v10.4 and later.

  Declared in `NSLocale.h`.

Declared In
NSLocale.h

NSLocale Calendar Keys

These constants identify NSCalendar instances.

```
extern NSString * const NSGregorianCalendar;
extern NSString * const NSBuddhistCalendar;
extern NSString * const NSChineseCalendar;
extern NSString * const NSHebrewCalendar;
extern NSString * const NSIslamicCalendar;
extern NSString * const NSIslamicCivilCalendar;
extern NSString * const NSJapaneseCalendar;
```

Constants

NSGregorianCalendar
Identifier for the Gregorian calendar.
Available in Mac OS X v10.4 and later.
Declared in NSLocale.h.

NSBuddhistCalendar
Identifier for the Buddhist calendar.
Available in Mac OS X v10.4 and later.
Declared in NSLocale.h.

NSChineseCalendar
Identifier for the Chinese calendar (unsupported).
Note that the Chinese calendar is not supported in Mac OS X v10.4-10.5. Although you can create a calendar using this constant, the object will not function correctly.
Available in Mac OS X v10.4 and later.
Declared in NSLocale.h.

NSHebrewCalendar
Identifier for the Hebrew calendar.
Available in Mac OS X v10.4 and later.
Declared in NSLocale.h.

NSIslamicCalendar
Identifier for the Islamic calendar.
Available in Mac OS X v10.4 and later.
Declared in NSLocale.h.

NSIslamicCivilCalendar
Identifier for the Islamic civil calendar.
Available in Mac OS X v10.4 and later.
Declared in NSLocale.h.

NSJapaneseCalendar
Identifier for the Japanese calendar.
Available in Mac OS X v10.4 and later.
Declared in NSLocale.h.

Discussion
You use these identifiers to initialize a new `NSCalendar` object, using `initWithCalendarIdentifier:` (page 207). You get one of these identifiers as the return value from `calendarIdentifier` (page 202).

Declared In
`NSLocale.h`

Notifications

NSCurrentLocaleDidChangeNotification

Notification that indicates that the user's locale changed.

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

Declared In
`NSLocale.h`

NSLock Class Reference

Inherits from	NSObject
Conforms to	NSLocking NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSLock.h
Companion guide	Threading Programming Guide
Related sample code	Aperture Image Resizer ExtractMovieAudioToAIFF QTExtractAndConvertToAIFF QTQuartzPlayer SimpleThreads

Overview

An `NSLock` object is used to coordinate the operation of multiple threads of execution within the same application. An `NSLock` object can be used to mediate access to an application's global data or to protect a critical section of code, allowing it to run atomically.

Warning: The `NSLock` class uses POSIX threads to implement its locking behavior. When sending an unlock message to an `NSLock` object, you must be sure that message is sent from the same thread that sent the initial lock message. Unlocking a lock from a different thread can result in undefined behavior.

You should not use this class to implement a recursive lock. Calling the `lock` method twice on the same thread will lock up your thread permanently. Use the `NSRecursiveLock` class to implement recursive locks instead.

Unlocking a lock that is not locked is considered a programmer error and should be fixed in your code. The `NSLock` class reports such errors by printing an error message to the console when they occur.

Adopted Protocols

NSLocking
- `- lock` (page 2091)
- `- unlock` (page 2092)

Tasks

Acquiring a Lock

`- lockBeforeDate:` (page 834)
Attempts to acquire a lock before a given time and returns a Boolean value indicating whether the attempt was successful.

`- tryLock` (page 835)
Attempts to acquire a lock and immediately returns a Boolean value that indicates whether the attempt was successful.

Naming the Lock

`- setName:` (page 835)
Assigns a name to the receiver.

`- name` (page 835)
Returns the name associated with the receiver.

Instance Methods

lockBeforeDate:

Attempts to acquire a lock before a given time and returns a Boolean value indicating whether the attempt was successful.

```
- (BOOL)lockBeforeDate:(NSDate *)limit
```

Parameters

limit
The time limit for attempting to acquire a lock.

Return Value
`YES` if the lock is acquired before *limit*, otherwise `NO`.

Discussion
The thread is blocked until the receiver acquires the lock or *limit* is reached.

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

Declared In
NSLock.h

name

Returns the name associated with the receiver.

```
- (NSString *)name
```

Return Value
The name of the receiver.

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

See Also
- setName: (page 835)

Declared In
NSLock.h

setName:

Assigns a name to the receiver.

```
- (void)setName:(NSString *)newName
```

Parameters

newName
The new name for the receiver. This method makes a copy of the specified string.

Discussion
You can use a name string to identify a lock within your code. Cocoa also uses this name as part of any error descriptions involving the receiver.

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

See Also
- name (page 835)

Declared In
NSLock.h

tryLock

Attempts to acquire a lock and immediately returns a Boolean value that indicates whether the attempt was successful.

```
- (BOOL)tryLock
```

Return Value
`YES` if the lock was acquired, otherwise `NO`.

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

Declared In
`NSLock.h`

NSLogicalTest Class Reference

Inherits from	NSScriptWhoseTest : NSObject
Conforms to	NSCoding (NSScriptWhoseTest) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptWhoseTests.h
Companion guide	Cocoa Scripting Guide

Overview

Instances of this class perform logical operations of `AND`, `OR`, and `NOT` on Boolean expressions represented by `NSSpecifierTest` objects. These operators are equivalent to "`&&`", "`||`", and "`!`" in the C language.

For `AND` and `OR` operations, an `NSLogicalTest` object is typically initialized with an array containing two or more `NSSpecifierTest` objects. `isTrue` (page 1438)—inherited from `NSScriptWhoseTest`—evaluates the array in a manner appropriate to the logical operation. For `NOT` operations, an `NSLogicalTest` object is initialized with only one `NSSpecifierTest` object; it simply reverses the Boolean outcome of the `isTrue` (page 1438) method.

You don't normally subclass `NSLogicalTest`.

Tasks

Initializing a Logical Test

`- initAndTestWithTests:` (page 838)
Returns an `NSLogicalTest` object initialized to perform an `AND` operation with the `NSSpecifierTest` objects in a given array.

`- initNotTestWithTest:` (page 838)
Returns an `NSLogicalTest` object initialized to perform a `NOT` operation on the given `NSScriptWhoseTest` object.

- `initOrTestWithTests:` (page 839)
  Returns an `NSLogicalTest` object initialized to perform an `OR` operation with the `NSSpecifierTest` objects in a given array.

Instance Methods

initAndTestWithTests:

Returns an `NSLogicalTest` object initialized to perform an `AND` operation with the `NSSpecifierTest` objects in a given array.

```
- (id)initAndTestWithTests:(NSArray *)subTests
```

Parameters

subTests
  An array of `NSSpecifierTest` objects representing Boolean expressions.

Return Value
An `NSLogicalTest` object initialized to perform an `AND` operation with the `NSSpecifierTest` objects in *subTests*.

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

Declared In
`NSScriptWhoseTests.h`

initNotTestWithTest:

Returns an `NSLogicalTest` object initialized to perform a `NOT` operation on the given `NSScriptWhoseTest` object.

```
- (id)initNotTestWithTest:(NSScriptWhoseTest *)subTest
```

Parameters

subTest
  The `NSScriptWhoseTest` object to invert.

Return Value
An `NSLogicalTest` object initialized to perform a `NOT` operation on *subTest*.

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

Declared In
`NSScriptWhoseTests.h`

initOrTestWithTests:

Returns an `NSLogicalTest` object initialized to perform an `OR` operation with the `NSSpecifierTest` objects in a given array.

```
- (id)initOrTestWithTests:(NSArray *)subTests
```

Parameters

subTests

An array of `NSSpecifierTest` objects representing Boolean expressions.

Return Value

An `NSLogicalTest` object initialized to perform an `OR` operation with the `NSSpecifierTest` objects in *subTests*.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSScriptWhoseTests.h`

NSMachBootstrapServer Class Reference

Inherits from	NSPortNameServer : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSPortNameServer.h
Companion guide	Distributed Objects Programming Topics

Overview

This port name server takes and returns instances of `NSMachPort`.

Port removal functionality is not supported in `NSMachBootstrapServer`; if you want to cancel a service, you have to destroy the port (invalidate the `NSMachPort` given to `registerPort:name:` (page 843)).

Tasks

Getting the Server Object

`+ sharedInstance` (page 842)
Returns the shared instance of the bootstrap server.

Looking Up Ports

`- portForName:` (page 842)
Looks up and returns the port registered under the specified name on the local host.

`- portForName:host:` (page 843)
Looks up and returns the port registered under the specified name.

`- servicePortWithName:` (page 843)
Looks up and returns the port for the vended service that is registered under the specified name.

Registering Ports

`- registerPort:name:` (page 843)
Registers a port with a specified name.

Class Methods

sharedInstance

Returns the shared instance of the bootstrap server.

```
+ (id)sharedInstance
```

Return Value
The shared instance of `NSMachBootstrapServer` with which you register and look up `NSMachPort` objects.

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

Declared In
`NSPortNameServer.h`

Instance Methods

portForName:

Looks up and returns the port registered under the specified name on the local host.

```
- (NSPort *)portForName:(NSString *)portName
```

Parameters
portName
The name of the desired port.

Return Value
The port associated with *portName* on the local host. Returns `nil` if no such port exists.

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

See Also
`- portForName:host:` (page 843)

Declared In
`NSPortNameServer.h`

portForName:host:

Looks up and returns the port registered under the specified name.

```
- (NSPort *)portForName:(NSString *)portName host:(NSString *)hostName
```

Parameters

portName

The name of the desired port.

hostName

Because `NSMachBootstrapServer` is a local-only server; *hostName* must be the empty string or `nil`.

Return Value

The port associated with *portName* on the local host. Returns `nil` if no such port exists.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSPortNameServer.h`

registerPort:name:

Registers a port with a specified name.

```
- (BOOL)registerPort:(NSPort *)port name:(NSString *)portName
```

Parameters

port

The port object to register with the bootstrap server.

portName

The name to associate with *port*.

Return Value

`YES` if the registration succeeded, `NO` otherwise.

Special Considerations

Once registered, a port cannot be unregistered; instead, you need to invalidate the port.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSPortNameServer.h`

servicePortWithName:

Looks up and returns the port for the vended service that is registered under the specified name.

```
- (NSPort *)servicePortWithName:(NSString *)name
```

Parameters

name

The name of the vended service.

Return Value

The port associated with *name*. Returns `nil` if no such port exists.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSPortNameServer.h`

NSMachPort Class Reference

Inherits from	NSPort : NSObject
Conforms to	NSCoding (NSPort) NSCopying (NSPort) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSPort.h
Companion guide	Distributed Objects Programming Topics

Overview

`NSMachPort` is a subclass of `NSPort` that can be used as an endpoint for distributed object connections (or raw messaging). `NSMachPort` is an object wrapper for a Mach port, the fundamental communication port in Mac OS X. `NSMachPort` allows for local (on the same machine) communication only. A companion class, `NSSocketPort`, allows for both local and remote distributed object communication, but may be more expensive than `NSMachPort` for the local case.

To use `NSMachPort` effectively, you should be familiar with Mach ports, port access rights, and Mach messages. See the Mach OS documentation for more information.

Note: `NSMachPort` conforms to the `NSCoding` protocol, but only supports coding by an `NSPortCoder`. `NSPort` and its subclasses do not support archiving.

Tasks

Creating and Initializing

`+ portWithMachPort:` (page 846)
Creates and returns a port object configured with the given Mach port.

`+ portWithMachPort:options:` (page 847)
Creates and returns a port object configured with the specified options and the given Mach port.

- `initWithMachPort:` (page 847)
  Initializes a newly allocated `NSMachPort` object with a given Mach port.
- `initWithMachPort:options:` (page 848)
  Initializes a newly allocated `NSMachPort` object with a given Mach port and the specified options.

Getting the Mach Port

- `machPort` (page 848)
  Returns as an `int` the Mach port used by the receiver.

Scheduling the Port on a Run Loop

- `removeFromRunLoop:forMode:` (page 848)
  Removes the receiver from the run loop mode *`mode`* of *`runLoop`*.
- `scheduleInRunLoop:forMode:` (page 849)
  Schedules the receiver into the run loop mode *`mode`* of *`runLoop`*.

Handling Mach Messages

- `handleMachMessage:` (page 849) *delegate method*
  Process an incoming Mach message.

Class Methods

portWithMachPort:

Creates and returns a port object configured with the given Mach port.

```
+ (NSPort *)portWithMachPort:(uint32_t)machPort
```

Parameters

`machPort`
The Mach port for the new port. This parameter should originally be of type `mach_port_t`.

Return Value
An `NSMachPort` object that uses *`machPort`* to send or receive messages.

Discussion
Creates the port object if necessary. Depending on the access rights associated with *`machPort`*, the new port object may be usable only for sending messages.

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

Declared In
`NSPort.h`

portWithMachPort:options:

Creates and returns a port object configured with the specified options and the given Mach port.

```
+ (NSPort *)portWithMachPort:(uint32_t)machPort options:(NSUInteger)options
```

Parameters

machPort

The Mach port for the new port. This parameter should originally be of type `mach_port_t`.

options

Specifies options for what to do with the underlying port rights when the `NSMachPort` object is invalidated or destroyed. For a list of constants, see "Mach Port Rights" (page 850).

Return Value

An `NSMachPort` object that uses *machPort* to send or receive messages.

Discussion

Creates the port object if necessary. Depending on the access rights associated with *machPort*, the new port object may be usable only for sending messages.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSPort.h`

Instance Methods

initWithMachPort:

Initializes a newly allocated `NSMachPort` object with a given Mach port.

```
- (id)initWithMachPort:(uint32_t)machPort
```

Parameters

machPort

The Mach port for the new port. This parameter should originally be of type `mach_port_t`.

Return Value

Returns an initialized `NSMachPort` object that uses *machPort* to send or receive messages. The returned object might be different than the original receiver

Discussion

Depending on the access rights for *machPort*, the new port may be able to only send messages. If a port with *machPort* already exists, this method deallocates the receiver, then retains and returns the existing port.

This method is the designated initializer for the `NSMachPort` class.

Availability

Available in Mac OS X v10.0 and later.

Declared In
NSPort.h

initWithMachPort:options:

Initializes a newly allocated NSMachPort object with a given Mach port and the specified options.

```
- (id)initWithMachPort:(uint32_t)machPort options:(NSUInteger)options
```

Parameters

machPort
The Mach port for the new port. This parameter should originally be of type mach_port_t.

options
Specifies options for what to do with the underlying port rights when the NSMachPort object is invalidated or destroyed. For a list of constants, see "Mach Port Rights" (page 850).

Return Value
Returns an initialized NSMachPort object that uses *machPort* to send or receive messages. The returned object might be different than the original receiver

Discussion
Depending on the access rights for *machPort*, the new port may be able to only send messages. If a port with *machPort* already exists, this method deallocates the receiver, then retains and returns the existing port.

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

Declared In
NSPort.h

machPort

Returns as an int the Mach port used by the receiver.

```
- (uint32_t)machPort
```

Return Value
The Mach port used by the receiver. Cast this value to a mach_port_t when using it with Mach system calls.

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

Declared In
NSPort.h

removeFromRunLoop:forMode:

Removes the receiver from the run loop mode *mode* of *runLoop*.

```
- (void)removeFromRunLoop:(NSRunLoop *)runLoop forMode:(NSString *)mode
```

Parameters

runLoop

The run loop from which to remove the receiver.

mode

The run loop mode from which to remove the receiver.

Discussion

When the receiver is removed, the run loop stops monitoring the Mach port for incoming messages.

Availability

Available in Mac OS X v10.0 and later.

See Also

- scheduleInRunLoop:forMode: (page 849)

Declared In

NSPort.h

scheduleInRunLoop:forMode:

Schedules the receiver into the run loop mode *mode* of *runLoop*.

```
- (void)scheduleInRunLoop:(NSRunLoop *)runLoop forMode:(NSString *)mode
```

Parameters

runLoop

The run loop to which to add the receiver.

mode

The run loop mode in which to add the receiver.

Discussion

When the receiver is scheduled, the run loop monitors the mach port for incoming messages and, when a message arrives, invokes the delegate method handleMachMessage: (page 849).

Availability

Available in Mac OS X v10.0 and later.

See Also

- removeFromRunLoop:forMode: (page 848)

Declared In

NSPort.h

Delegate Methods

handleMachMessage:

Process an incoming Mach message.

```
- (void)handleMachMessage:(void *)machMessage
```

Parameters

machMessage

A pointer to a Mach message, cast as a pointer to `void`.

Discussion

The delegate should interpret this data as a pointer to a Mach message beginning with a `msg_header_t` structure and should handle the message appropriately.

The delegate should implement only one of `handleMachMessage:` and `handlePortMessage:` (page 1255).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSPort.h`

Constants

Mach Port Rights

Used to remove access rights to a mach port when the `NSMachPort` object is invalidated or destroyed.

```
enum {
    NSMachPortDeallocateNone = 0,
    NSMachPortDeallocateSendRight = (1 << 0),
    NSMachPortDeallocateReceiveRight = (1 << 1)
};
```

Constants

`NSMachPortDeallocateNone`

Do not remove any send or receive rights.

Available in Mac OS X v10.5 and later.

Declared in `NSPort.h`.

`NSMachPortDeallocateSendRight`

Deallocate a send right when the `NSMachPort` object is invalidated or destroyed.

Available in Mac OS X v10.5 and later.

Declared in `NSPort.h`.

`NSMachPortDeallocateReceiveRight`

Remove a receive right when the `NSMachPort` object is invalidated or destroyed.

Available in Mac OS X v10.5 and later.

Declared in `NSPort.h`.

Declared In

`NSPort.h`

NSMapTable Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSFastEnumeration NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSMapTable.h
Availability	Available in Mac OS X v10.5 and later.
Companion guides	Garbage Collection Programming Guide Collections Programming Topics for Cocoa

Overview

`NSMapTable` is a mutable collection modeled after `NSDictionary` but provides different options, in particular to support weak relationships in a garbage-collected environment.

`NSMapTable` is modeled after `NSDictionary` but offers different behaviors:

- It can hold weak references to its keys and/or values.

  Keys and/or values held "weakly" in a manner that entries are removed when one of the objects is collected under garbage collection.

  If you are not using garbage collection, you must explicitly remove entries as you would from a dictionary. In addition to being held weakly, keys or values may be copied on input or may use pointer identity for equality and hashing.

- It can contain arbitrary pointers (its contents are not constrained to being objects).

  You can configure an `NSMapTable` instance to operate on arbitrary pointers and not just objects, although typically you are encouraged to use the C function API for `void *` pointers. The object-based API (such as `setObject:forKey:` (page 859)) will not work for non-object pointers without type-casting.

To configure an `NSMapTable` instance for pointer use, you can: create or initialize it using `mapTableWithKeyOptions:valueOptions:` (page 853) or `initWithKeyOptions:valueOptions:capacity:` (page 856) and the appropriate `NSPointerFunctionsOptions` (page 1244) options; or initialize it with `initWithKeyPointerFunctions:valuePointerFunctions:capacity:` (page 856) and appropriate instances of `NSPointerFunctions`. Note that only the options listed in "`Personality Options`" (page

860) guarantee that the rest of the API will work correctly—including copying, archiving, and fast enumeration. If you use other `NSPointerFunctions` options, the map table may not work correctly, or may not even be initialized correctly.

Tasks

Creating and Initializing a Map Table

- `initWithKeyOptions:valueOptions:capacity:` (page 856)
  Returns a map table, initialized with the given options.
+ `mapTableWithKeyOptions:valueOptions:` (page 853)
  Returns a new map table, initialized with the given options
- `initWithKeyPointerFunctions:valuePointerFunctions:capacity:` (page 856)
  Returns a map table, initialized with the given functions.
+ `mapTableWithStrongToStrongObjects` (page 854)
  Returns a new map table object which has strong references to the keys and values.
+ `mapTableWithWeakToStrongObjects` (page 854)
  Returns a new map table object which has weak references to the keys and strong references to the values.
+ `mapTableWithStrongToWeakObjects` (page 854)
  Returns a new map table object which has strong references to the keys and weak references to the values.
+ `mapTableWithWeakToWeakObjects` (page 855)
  Returns a new map table object which has weak references to the keys and values.

Accessing Content

- `objectForKey:` (page 858)
  Returns a the value associated with a given key.
- `keyEnumerator` (page 857)
  Returns an enumerator object that lets you access each key in the receiver.
- `objectEnumerator` (page 858)
  Returns an enumerator object that lets you access each value in the receiver.
- `count` (page 855)
  Returns the number of key-value pairs in the receiver.

Manipulating Content

- `setObject:forKey:` (page 859)
  Adds a given key-value pair to the receiver.
- `removeObjectForKey:` (page 859)
  Removes a given key and its associated value from the receiver.

- `removeAllObjects` (page 859)
  Empties the receiver of its entries.

Creating a Dictionary Representation

- `dictionaryRepresentation` (page 855)
  Returns a dictionary representation of the receiver.

Accessing Pointer Functions

- `keyPointerFunctions` (page 858)
  Returns the pointer functions the receiver uses to manage keys.
- `valuePointerFunctions` (page 860)
  Returns the pointer functions the receiver uses to manage values.

Class Methods

mapTableWithKeyOptions:valueOptions:

Returns a new map table, initialized with the given options

```
+ (id)mapTableWithKeyOptions:(NSPointerFunctionsOptions)keyOptions
    valueOptions:(NSPointerFunctionsOptions)valueOptions
```

Parameters

keys
  A bit field that specifies the options for the keys in the map table.

Important: Not all values of `NSPointerFunctionsOptions` (page 1244) are valid for `NSMapTable`. For values that are guaranteed to work correctly, see "`Personality Options`" (page 860).

values
  A bit field that specifies the options for the values in the map table.

Important: Not all values of `NSPointerFunctionsOptions` (page 1244) are valid for `NSMapTable`. For values that are guaranteed to work correctly, see "`Personality Options`" (page 860).

Return Value
A new map table, initialized with the given options.

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

See Also
- `initWithKeyOptions:valueOptions:capacity:` (page 856)

- initWithKeyPointerFunctions:valuePointerFunctions:capacity: (page 856)

Declared In
NSMapTable.h

mapTableWithStrongToStrongObjects

Returns a new map table object which has strong references to the keys and values.

```
+ (id)mapTableWithStrongToStrongObjects
```

Return Value
A new map table object which has strong references to the keys and values.

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

Declared In
NSMapTable.h

mapTableWithStrongToWeakObjects

Returns a new map table object which has strong references to the keys and weak references to the values.

```
+ (id)mapTableWithStrongToWeakObjects
```

Return Value
A new map table object which has strong references to the keys and weak references to the values.

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

Declared In
NSMapTable.h

mapTableWithWeakToStrongObjects

Returns a new map table object which has weak references to the keys and strong references to the values.

```
+ (id)mapTableWithWeakToStrongObjects
```

Return Value
A new map table object which has weak references to the keys and strong references to the values.

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

Declared In
NSMapTable.h

mapTableWithWeakToWeakObjects

Returns a new map table object which has weak references to the keys and values.

```
+ (id)mapTableWithWeakToWeakObjects
```

Return Value
A new map table object which has weak references to the keys and values.

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

Declared In
`NSMapTable.h`

Instance Methods

count

Returns the number of key-value pairs in the receiver.

```
- (NSUInteger)count
```

Return Value
The number of key-value pairs in the receiver.

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

Declared In
`NSMapTable.h`

dictionaryRepresentation

Returns a dictionary representation of the receiver.

```
- (NSDictionary *)dictionaryRepresentation
```

Return Value
A dictionary representation of the receiver.

Discussion
The receiver's contents must be objects.

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

Declared In
`NSMapTable.h`

initWithKeyOptions:valueOptions:capacity:

Returns a map table, initialized with the given options.

```
- (id)initWithKeyOptions:(NSPointerFunctionsOptions)keyOptions
    valueOptions:(NSPointerFunctionsOptions)valueOptions
    capacity:(NSUInteger)initialCapacity
```

Parameters

keys

A bit field that specifies the options for the keys in the map table.

Important: Not all values of `NSPointerFunctionsOptions` (page 1244) are valid for `NSMapTable`. For values that are guaranteed to work correctly, see "Personality Options" (page 860).

values

A bit field that specifies the options for the values in the map table.

Important: Not all values of `NSPointerFunctionsOptions` (page 1244) are valid for `NSMapTable`. For values that are guaranteed to work correctly, see "Personality Options" (page 860).

capacity

The initial capacity of the receiver. This is just a hint; the map table may subsequently grow and shrink as required.

Return Value

A map table initialized using the given options.

Discussion

values must contain entries at all the indexes specified in *keys*.

Availability

Available in Mac OS X v10.5 and later.

See Also

`+ mapTableWithKeyOptions:valueOptions:` (page 853)

`- initWithKeyPointerFunctions:valuePointerFunctions:capacity:` (page 856)

Declared In

`NSMapTable.h`

initWithKeyPointerFunctions:valuePointerFunctions:capacity:

Returns a map table, initialized with the given functions.

```
- (id)initWithKeyPointerFunctions:(NSPointerFunctions *)keyFunctions
    valuePointerFunctions:(NSPointerFunctions *)valueFunctions
    capacity:(NSUInteger)initialCapacity
```

Parameters

keyFunctions

The functions the receiver uses to manage keys.

Important: Not all values of `NSPointerFunctionsOptions` (page 1244) are valid for `NSMapTable`. For values that are guaranteed to work correctly, see "`Personality Options`" (page 860).

valueFunctions

The functions the receiver uses to manage values.

Important: Not all values of `NSPointerFunctionsOptions` (page 1244) are valid for `NSMapTable`. For values that are guaranteed to work correctly, see "`Personality Options`" (page 860).

initialCapacity

The initial capacity of the receiver. This is just a hint; the map table may subsequently grow and shrink as required.

Return Value
A map table, initialized with the given functions.

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

Declared In
`NSMapTable.h`

keyEnumerator

Returns an enumerator object that lets you access each key in the receiver.

```
- (NSEnumerator *)keyEnumerator
```

Return Value
An enumerator object that lets you access each key in the receiver.

Discussion
The following code fragment illustrates how you might use the method.

```
NSEnumerator *enumerator = [myMapTable keyEnumerator];
id value;

while ((value = [enumerator nextObject])) {
    /* code that acts on the map table's keys */
}
```

See also `NSFastEnumeration`.

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

Declared In
`NSMapTable.h`

keyPointerFunctions

Returns the pointer functions the receiver uses to manage keys.

```
- (NSPointerFunctions *)keyPointerFunctions
```

Return Value
The pointer functions the receiver uses to manage keys.

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

See Also
- `valuePointerFunctions` (page 860)

Declared In
`NSMapTable.h`

objectEnumerator

Returns an enumerator object that lets you access each value in the receiver.

```
- (NSEnumerator *)objectEnumerator
```

Return Value
An enumerator object that lets you access each value in the receiver.

Discussion
The following code fragment illustrates how you might use the method.

```
NSEnumerator *enumerator = [myMapTable objectEnumerator];
id value;

while ((value = [enumerator nextObject])) {
    /* code that acts on the map table's values */
}
```

See also `NSFastEnumeration`.

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

Declared In
`NSMapTable.h`

objectForKey:

Returns a the value associated with a given key.

```
- (id)objectForKey:(id)aKey
```

Parameters
aKey
The key for which to return the corresponding value.

Return Value
The value associated with *aKey*, or `nil` if no value is associated with *aKey*.

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

Declared In
`NSMapTable.h`

removeAllObjects

Empties the receiver of its entries.

```
- (void)removeAllObjects
```

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

Declared In
`NSMapTable.h`

removeObjectForKey:

Removes a given key and its associated value from the receiver.

```
- (void)removeObjectForKey:(id)aKey
```

Parameters

aKey
The key to remove.

Discussion
Does nothing if *aKey* does not exist.

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

Declared In
`NSMapTable.h`

setObject:forKey:

Adds a given key-value pair to the receiver.

```
- (void)setObject:(id)anObject
    forKey:(id)aKey
```

Parameters

anObject
The value for *aKey*. This value must not be `nil`.

aKey
The key for *anObject*. This value must not be `nil`.

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

Declared In
NSMapTable.h

valuePointerFunctions

Returns the pointer functions the receiver uses to manage values.

```
- (NSPointerFunctions *)valuePointerFunctions
```

Return Value
The pointer functions the receiver uses to manage values.

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

See Also
- keyPointerFunctions (page 858)

Declared In
NSMapTable.h

Constants

Personality Options

Constants used as components in a bitfield to specify the behavior of elements (keys and values) in an NSMapTable object.

```
enum {
    NSMapTableStrongMemory             = 0,
    NSMapTableZeroingWeakMemory        = NSPointerFunctionsZeroingWeakMemory,
    NSMapTableCopyIn                   = NSPointerFunctionsCopyIn,
   NSMapTableObjectPointerPersonality = NSPointerFunctionsObjectPointerPersonality
};
```

Constants
NSMapTableStrongMemory
Specifies a strong reference from the map table to its contents.
Equal to NSPointerFunctionsStrongMemory.
Available in Mac OS X v10.5 and later.
Declared in NSMapTable.h.

`NSMapTableZeroingWeakMemory`
: Specifies a zeroing weak reference from the map table to its contents.

  Equal to `NSPointerFunctionsZeroingWeakMemory`.

  Available in Mac OS X v10.5 and later.

  Declared in `NSMapTable.h`.

`NSMapTableCopyIn`
: Use the memory acquire function to allocate and copy items on input (see `acquireFunction` [`NSPointerFunctions`]).

  Equal to `NSPointerFunctionsCopyIn`.

  Available in Mac OS X v10.5 and later.

  Declared in `NSMapTable.h`.

`NSMapTableObjectPointerPersonality`
: Use shifted pointer hash and direct equality, object description.

  Equal to `NSPointerFunctionsObjectPointerPersonality`.

  Available in Mac OS X v10.5 and later.

  Declared in `NSMapTable.h`.

Declared In
`NSMapTable.h`

NSMessagePort Class Reference

Inherits from	NSPort : NSObject
Conforms to	NSCoding (NSPort) NSCopying (NSPort) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSPort.h
Companion guide	Distributed Objects Programming Topics

Overview

`NSMessagePort` is a subclass of `NSPort` that can be used as an endpoint for distributed object connections (or raw messaging). `NSMessagePort` allows for local (on the same machine) communication only. A companion class, `NSSocketPort`, allows for both local and remote communication, but may be more expensive than `NSMessagePort` for the local case.

`NSMessagePort` defines no additional methods over those already defined by `NSPort`.

Note: `NSMessagePort` conforms to the `NSCoding` protocol, but only supports coding by an `NSPortCoder` object. `NSPort` and its subclasses do not support archiving.

NSMessagePortNameServer Class Reference

Inherits from	NSPortNameServer : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSPortNameServer.h
Companion guide	Distributed Objects Programming Topics

Overview

This port name server takes and returns instances of `NSMessagePort`. Port removal functionality is not supported in `NSMessagePortNameServer`; if you want to cancel a service, you have to destroy the port (invalidate the `NSMessagePort` object given to `registerPort:name:` (page 1271)).

Tasks

Getting the Server Object

`+ sharedInstance` (page 866)
Returns the singleton instance of `NSMessagePortNameServer`.

Getting Ports By Name

`- portForName:` (page 866)
Returns the `NSPort` object registered under a given name on the local host.

`- portForName:host:` (page 866)
Returns the `NSPort` object registered under a given name on the local host.

Class Methods

sharedInstance

Returns the singleton instance of `NSMessagePortNameServer`.

```
+ (id)sharedInstance
```

Return Value
The singleton instance of `NSMessagePortNameServer` with which you register and look up `NSMessagePort` objects.

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

Declared In
`NSPortNameServer.h`

Instance Methods

portForName:

Returns the `NSPort` object registered under a given name on the local host.

```
- (NSPort *)portForName:(NSString *)portName
```

Parameters
portName
The port name.

Return Value
The NSPort registered under *portName* on the local host Returns `nil` if a port named *portName* does not exist.

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

See Also
`- portForName:host:` (page 866)

Declared In
`NSPortNameServer.h`

portForName:host:

Returns the `NSPort` object registered under a given name on the local host.

```
- (NSPort *)portForName:(NSString *)portName host:(NSString *)hostName
```

Parameters

portName

The port name.

hostName

The host name. Because `NSMessagePortNameServer` is a local-only server, *hostName* must be the empty string or `nil`.

Return Value

The `NSPort` object registered under a given name on the local host. Returns `nil` if a port named *portName* does not exist.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSPortNameServer.h`

NSMetadataItem Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSMetadata.h
Availability	Available in Mac OS X v10.4 and later.
Companion guides	Spotlight Query Programming Guide Spotlight Metadata Attributes Reference
Related sample code	CoreRecipes iSpend PredicateEditorSample

Overview

The `NSMetadataItem` class represents the metadata associated with a file, providing a simple interface to retrieve the available attribute names and values.

Adopted Protocols

NSCopying
- `copyWithZone:` (page 2042)

Tasks

Getting Item Attributes

- `attributes` (page 870)
  Returns an array containing the attribute names of the receiver's values.
- `valueForAttribute:` (page 870)
  Returns the receiver's metadata attribute name specified by a given key.

Instance Methods

attributes

Returns an array containing the attribute names of the receiver's values.

```
- (NSArray *)attributes
```

Return Value
An array containing the attribute names of the receiver's values.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSMetadata.h

valueForAttribute:

Returns the receiver's metadata attribute name specified by a given key.

```
- (id)valueForAttribute:(NSString *)key
```

Parameters
key
The name of a metadata attribute.

Return Value
The receiver's metadata attribute name specified by *key*.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
CoreRecipes
PredicateEditorSample

Declared In
NSMetadata.h

valuesForAttributes:

Returns a dictionary containing the key-value pairs for the attribute names specified by a given array of keys.

```
- (NSDictionary *)valuesForAttributes:(NSArray *)keys
```

Parameters

keys

An array containing `NSString` objects that specify the names of a metadata attributes.

Return Value

A dictionary containing the key-value pairs for the attribute names specified by *keys*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`NSMetadata.h`

NSMetadataQuery Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSMetadata.h
Availability	Available in Mac OS X v10.4 and later.
Companion guide	Spotlight Query Programming Guide
Related sample code	CoreRecipes iSpend PredicateEditorSample SpotlightFortunes

Overview

The `NSMetadataQuery` class encapsulates the functionality provided by the `MDQuery` opaque type for querying the Spotlight metadata.

`NSMetadataQuery` objects provide metadata query results in several ways:

- As individual attribute values for requested attributes.
- As value lists that contain the distinct values for given attributes in the query results.
- A result array proxy, containing all the query results. This is suitable for use with Cocoa bindings.
- As a hierarchical collection of results, grouping together items with the same values for specified grouping attributes. This is also suitable for use with Cocoa bindings.

Queries have two phases: the initial gathering phase that collects all currently matching results and a second live-update phase.

By default the receiver has no limitation on its search scope. Use `setSearchScopes:` (page 883) to customize.

By default, notification of updated results occurs at 1.0 seconds. Use `setNotificationBatchingInterval:` (page 882) to customize.

You must set a predicate with the `setPredicate:` (page 883) method before starting a query.

Tasks

Creating Metadata Queries

- `init` (page 878)
  Initializes an allocated `NSMetadataQuery` object.

Configuring Queries

- `searchScopes` (page 881)
  Returns an array containing the receiver's search scopes.
- `setSearchScopes:` (page 883)
  Resctrict the search scope of the receiver.
- `predicate` (page 879)
  Returns the predicate the receiver uses to filter query results.
- `setPredicate:` (page 883)
  Sets the predicate used by the receiver to filter the query results.
- `sortDescriptors` (page 884)
  Returns an array containing the receiver's sort descriptors.
- `setSortDescriptors:` (page 883)
  Sets the sort descriptors to be used by the receiver.
- `valueListAttributes` (page 885)
  Returns an array containing the value list attributes the receiver generates.
- `setValueListAttributes:` (page 884)
  Sets the value list attributes for the receiver to the specific attribute names.
- `groupingAttributes` (page 877)
  Returns the receiver's grouping attributes.
- `setGroupingAttributes:` (page 882)
  Sets the receiver's grouping attributes to specific attribute names.
- `notificationBatchingInterval` (page 879)
  Returns the interval that the receiver provides notification of updated query results.
- `setNotificationBatchingInterval:` (page 882)
  Sets the interval between update notifications sent by the receiver.
- `delegate` (page 875)
  Returns the receiver's delegate.
- `setDelegate:` (page 881)
  Sets the receiver's delegate

Running Queries

- `isStarted` (page 878)
  Returns a Boolean value that indicates whether the receiver has started the query.

- `startQuery` (page 885)
  Attempts to start the query.
- `isGathering` (page 878)
  Returns a Boolean value that indicates whether the receiver is in the initial gathering phase of the query.
- `isStopped` (page 879)
  Returns a Boolean value that indicates whether the receiver has stopped the query.
- `stopQuery` (page 885)
  Stops the receiver's current query from gathering any further results.

Getting Query Results

- `resultCount` (page 880)
  Returns the number of results returned by the receiver.
- `resultAtIndex:` (page 880)
  Returns the query result at a specific index.
- `results` (page 880)
  Returns an array containing the result objects for the receiver.
- `groupedResults` (page 876)
  Returns an array containing hierarchical groups of query results based on the receiver's grouping attributes.
- `indexOfResult:` (page 877)
  Returns the index of a query result object in the receiver's results array.
- `valueLists` (page 886)
  Returns a dictionary containing the value lists generated by the receiver.
- `metadataQuery:replacementObjectForResultObject:` (page 887) *delegate method*
  Implemented by the delegate to return a different object for a specific query result object.
- `valueOfAttribute:forResultAtIndex:` (page 886)
  Returns the value for the attribute name *`attrName`* at the index in the results specified by *`idx`*.
- `metadataQuery:replacementValueForAttribute:value:` (page 887) *delegate method*
  Implemented by the delegate to return a different value for a specific attribute.
- `enableUpdates` (page 876)
  Enables updates to the query results.
- `disableUpdates` (page 876)
  Disables updates to the query results.

Instance Methods

delegate

Returns the receiver's delegate.

```
- (id)delegate
```

Return Value
The receiver's delegate, or `nil` if there is none.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setDelegate:` (page 881)

Declared In
`NSMetadata.h`

disableUpdates

Disables updates to the query results.

```
- (void)disableUpdates
```

Discussion
You should invoke this method before iterating over query results that could change due to live updates.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- enableUpdates` (page 876)

Declared In
`NSMetadata.h`

enableUpdates

Enables updates to the query results.

```
- (void)enableUpdates
```

Discussion
You should invoke this method after you're done iterating over the query results.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- disableUpdates` (page 876)

Declared In
`NSMetadata.h`

groupedResults

Returns an array containing hierarchical groups of query results based on the receiver's grouping attributes.

```
- (NSArray *)groupedResults
```

Return Value
Array containing hierarchical groups of query results.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `groupingAttributes` (page 877)
- `setGroupingAttributes:` (page 882)

Declared In
`NSMetadata.h`

groupingAttributes

Returns the receiver's grouping attributes.

```
- (NSArray *)groupingAttributes
```

Return Value
Array containing grouping attributes.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setGroupingAttributes:` (page 882)

Declared In
`NSMetadata.h`

indexOfResult:

Returns the index of a query result object in the receiver's results array.

```
- (NSUInteger)indexOfResult:(id)result
```

Parameters
result
Query result object being inquired about.

Return Value
Index of *result* in the query result array.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `resultAtIndex:` (page 880)

Declared In
`NSMetadata.h`

init

Initializes an allocated `NSMetadataQuery` object.

```
- (id)init
```

Return Value
An initialized `NSMetadataQuery` object.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSMetadata.h`

isGathering

Returns a Boolean value that indicates whether the receiver is in the initial gathering phase of the query.

```
- (BOOL)isGathering
```

Return Value
`YES` when the query is in the initial gathering phase; `NO` otherwise.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `isStarted` (page 878)
- `isStopped` (page 879)
- `startQuery` (page 885)

Declared In
`NSMetadata.h`

isStarted

Returns a Boolean value that indicates whether the receiver has started the query.

```
- (BOOL)isStarted
```

Return Value
`YES` when the receiver has executed the `startQuery` method; `NO` otherwise.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `isGathering` (page 878)
- `isStopped` (page 879)
- `startQuery` (page 885)

Declared In
NSMetadata.h

isStopped

Returns a Boolean value that indicates whether the receiver has stopped the query.

```
- (BOOL)isStopped
```

Return Value
YES when the receiver has stopped the query, NO otherwise.

Availability
Available in Mac OS X v10.4 and later.

See Also
- isGathering (page 878)
- isStarted (page 878)
- stopQuery (page 885)

Declared In
NSMetadata.h

notificationBatchingInterval

Returns the interval that the receiver provides notification of updated query results.

```
- (NSTimeInterval)notificationBatchingInterval
```

Return Value
The interval at which notification of updated results occurs.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setNotificationBatchingInterval: (page 882)

Declared In
NSMetadata.h

predicate

Returns the predicate the receiver uses to filter query results.

```
- (NSPredicate *)predicate
```

Return Value
The predicate used to filter query results.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setPredicate:` (page 883)

Declared In
`NSMetadata.h`

resultAtIndex:

Returns the query result at a specific index.

```
- (id)resultAtIndex:(NSUInteger)index
```

Parameters

index
Index of the desired result in the query result array.

Return Value
Query result at the position specified by *index*.

Discussion
For performance reasons, you should use this method when retrieving a specific result, rather than they array returned by `results` (page 880).

Availability
Available in Mac OS X v10.4 and later.

See Also
- `indexOfResult:` (page 877)

Declared In
`NSMetadata.h`

resultCount

Returns the number of results returned by the receiver.

```
- (NSUInteger)resultCount
```

Return Value
The number of objects the query produced.

Discussion
For performance reasons, you should use this method, rather than invoking `count` on `results` (page 880).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSMetadata.h`

results

Returns an array containing the result objects for the receiver.

```
- (NSArray *)results
```

Return Value
Proxy array containing query result objects.

Discussion
The results array is a proxy object that is primarily intended for use with Cocoa bindings. While it is possible to copy the proxy array and receive a "snapshot" of the complete current query results, it is generally not recommended due to performance and memory issues. To access individual result array elements you should instead use the `resultCount` (page 880) and `resultAtIndex:` (page 880) methods.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- groupedResults` (page 876)

Declared In
`NSMetadata.h`

searchScopes

Returns an array containing the receiver's search scopes.

```
- (NSArray *)searchScopes
```

Return Value
An array containing the receiver's search scopes.

Discussion
The array can contain `NSString` or `NSURL` objects that represent file system directories or the search scopes specified in "Constants" (page 888). An empty array indicates that there is no limitation on where the receiver searches.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setSearchScopes:` (page 883)

Declared In
`NSMetadata.h`

setDelegate:

Sets the receiver's delegate

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

Parameters
delegate
An object to serve as the receiver's delegate. Pass `nil` to remove the current delegate.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `delegate` (page 875)

Declared In
`NSMetadata.h`

setGroupingAttributes:

Sets the receiver's grouping attributes to specific attribute names.

```
- (void)setGroupingAttributes:(NSArray *)attributes
```

Parameters

attributes
Array containing attribute names.

Discussion
Invoking this method on a receiver while it's running a query, stops the query and discards current results, and immediately starts a new query.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `groupingAttributes` (page 877)

Declared In
`NSMetadata.h`

setNotificationBatchingInterval:

Sets the interval between update notifications sent by the receiver.

```
- (void)setNotificationBatchingInterval:(NSTimeInterval)timeInterval
```

Parameters

Term
The Interval at which notification of updated results is to occur.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `notificationBatchingInterval` (page 879)

Declared In
`NSMetadata.h`

setPredicate:

Sets the predicate used by the receiver to filter the query results.

```
- (void)setPredicate:(NSPredicate *)predicate
```

Parameters

predicate

A predicate to be used to filter query results.

Discussion

Invoking this method on a receiver running a query causes the existing query to stop, all current results are discarded, and a new query is started immediately.

Availability

Available in Mac OS X v10.4 and later.

See Also

- predicate (page 879)

Declared In

NSMetadata.h

setSearchScopes:

Resctrict the search scope of the receiver.

```
- (void)setSearchScopes:(NSArray *)scopes
```

Parameters

scopes

Array of NSString or NSURL objects that specify file system directories. You can also include the predefined search scopes specified in "Constants" (page 888). An empty array removes search scope limitations.

Availability

Available in Mac OS X v10.4 and later.

See Also

- searchScopes (page 881)

Declared In

NSMetadata.h

setSortDescriptors:

Sets the sort descriptors to be used by the receiver.

```
- (void)setSortDescriptors:(NSArray *)descriptors
```

Parameters

descriptors

Array of sort descriptors.

Discussion
Invoking this method on the receiver running a query causes the existing query to stop, all current results are discarded, and a new query is started immediately.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `sortDescriptors` (page 884)

Declared In
`NSMetadata.h`

setValueListAttributes:

Sets the value list attributes for the receiver to the specific attribute names.

```
- (void)setValueListAttributes:(NSArray *)attributes
```

Parameters
attributes
Array of value list attributes.

Discussion
The query collects the values of these attributes into uniqued lists that can be used to summarize the results of the query. If *attributess* is `nil`, the query generates no value lists. Note that value list collection increases CPU usage and significantly increases the memory usage of an `NSMetadataQuery` object.

Invoking this method on the receiver while it's running a query, stops the query and discards current results, and immediately starts a new query.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `valueListAttributes` (page 885)

Declared In
`NSMetadata.h`

sortDescriptors

Returns an array containing the receiver's sort descriptors.

```
- (NSArray *)sortDescriptors
```

Return Value
An array containing sort descriptors.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setSortDescriptors:` (page 883)

Declared In
NSMetadata.h

startQuery

Attempts to start the query.

```
- (BOOL)startQuery
```

Return Value
YES when successful; NO otherwise.

Discussion
A query can't be started if the receiver is already running a query or no predicate has been specified.

Availability
Available in Mac OS X v10.4 and later.

See Also
- stopQuery (page 885)
- isStarted (page 878)

Declared In
NSMetadata.h

stopQuery

Stops the receiver's current query from gathering any further results.

```
- (void)stopQuery
```

Discussion
The receiver first completes gathering any unprocessed results. If a query is stopped before the gathering phase finishes, it will not post an NSMetadataQueryDidStartGatheringNotification notification.

You would call this function to stop a query that is generating too many results to be useful but still want to access the available results. If the receiver is sent a startQuery message after performing this method, the existing results are discarded.

Availability
Available in Mac OS X v10.4 and later.

See Also
- startQuery (page 885)
- isStopped (page 879)

Declared In
NSMetadata.h

valueListAttributes

Returns an array containing the value list attributes the receiver generates.

```
- (NSArray *)valueListAttributes
```

Return Value
Array containing value list attributes.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setValueListAttributes:` (page 884)

Declared In
`NSMetadata.h`

valueLists

Returns a dictionary containing the value lists generated by the receiver.

```
- (NSDictionary *)valueLists
```

Return Value
Dictionary of `NSMetadataQueryAttributeValueTuple` objects.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSMetadata.h`

valueOfAttribute:forResultAtIndex:

Returns the value for the attribute name *attrName* at the index in the results specified by *idx*.

```
- (id)valueOfAttribute:(NSString *)attributeName forResultAtIndex:(NSUInteger)index
```

Parameters

attributeName
The attribute of the result object at *index* being inquired about. The attribute must be specified in `setValueListAttributes:` (page 884), as a sorting key in a specified sort descriptor, or as one of the grouping attributes specified set for the query.

index
Index of the desired return object in the query results array.

Return Value
Value for `attributeName` in the result object at `index` in the query result array.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSMetadata.h`

Delegate Methods

metadataQuery:replacementObjectForResultObject:

Implemented by the delegate to return a different object for a specific query result object.

```
- (id)metadataQuery:(NSMetadataQuery *)query
    replacementObjectForResultObject:(NSMetadataItem *)result
```

Parameters

query

The query that produced the result object to replace.

result

The query result object to replace.

Return Value

Object that replaces the query result object.

Discussion

By default query result objects are instances of the `NSMetadataItem` class. By implementing this method, you can return an object of a different class type for the specified result object.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`NSMetadata.h`

metadataQuery:replacementValueForAttribute:value:

Implemented by the delegate to return a different value for a specific attribute.

```
- (id)metadataQuery:(NSMetadataQuery *)query replacementValueForAttribute:(NSString
    *)attribute value:(id)attributeValue
```

Parameters

query

The query that produced the result object with *attribute*.

attribute

The attribute in question.

attributeValue

The attribute value to replace.

Return Value

Object that replaces the value of *attribute* in the result object

Discussion

The delegate implementation of this method could convert specific query attribute values to other attribute values, for example, converting date object values to formatted strings for display.

Availability

Available in Mac OS X v10.4 and later.

Declared In
NSMetadata.h

Constants

Metadata Query Search Scopes

Constants for the predefined search scopes used by `setSearchScopes:` (page 883).

```
NSString * const NSMetadataQueryUserHomeScope;
NSString * const NSMetadataQueryLocalComputerScope;
NSString * const NSMetadataQueryNetworkScope;
```

Constants

`NSMetadataQueryUserHomeScope`
Search the user's home directory.
Available in Mac OS X v10.4 and later.
Declared in `NSMetadata.h`.

`NSMetadataQueryLocalComputerScope`
Search all local mounted volumes, including the user home directory. The user's home directory is searched even if it is a remote volume.
Available in Mac OS X v10.4 and later.
Declared in `NSMetadata.h`.

`NSMetadataQueryNetworkScope`
Search all user-mounted remote volumes.
Available in Mac OS X v10.4 and later.
Declared in `NSMetadata.h`.

Content Relevance

In addition to the requested metadata attributes, a query result also includes content relevance, accessed with the following key.

```
NSString * const NSMetadataQueryResultContentRelevanceAttribute;
```

Constants

`NSMetadataQueryResultContentRelevanceAttribute`
Key used to retrieve an `NSNumber` object with a floating point value between 0.0 and 1.0 inclusive. The relevance value indicates the relevance of the content of a result object. The relevance is computed based on the value of the result itself, not on its relevance to the other results returned by the query. If the value is not computed, it is treated as an attribute on the item that does not exist.
Available in Mac OS X v10.4 and later.
Declared in `NSMetadata.h`.

Notifications

NSMetadataQueryDidFinishGatheringNotification

Posted when the receiver has finished with the initial result-gathering phase of the query.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSMetadata.h`

NSMetadataQueryDidStartGatheringNotification

Posted when the receiver begins with the initial result-gathering phase of the query.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSMetadata.h`

NSMetadataQueryDidUpdateNotification

Posted when the receiver's results have changed during the live-update phase of the query.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSMetadata.h`

NSMetadataQueryGatheringProgressNotification

Posted as the receiver's is collecting results during the initial result-gathering phase of the query.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSMetadata.h`

NSMetadataQueryAttributeValueTuple Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSMetadata.h
Availability	Available in Mac OS X v10.4 and later.
Companion guide	Spotlight Metadata Attributes Reference

Overview

The `NSMetadataQueryAttributeValueTuple` class represents attribute-value tuples, which are objects that contain the attribute name and value of a metadata attribute.

Attribute-value tuples are returned by `NSMetadataQuery` objects as the results in the value lists. Each attribute/value tuple contains the attribute name, the value, and the number of instances of that value that exist for the attribute name.

Tasks

Getting Query Attribute/Value Information

- `attribute` (page 892)
  Returns the receiver's attribute name.
- `count` (page 892)
  Returns the number of instances of the value that exist for the attribute name of the receiver.
- `value` (page 892)
  Returns the receiver's attribute value.

Instance Methods

attribute

Returns the receiver's attribute name.

```
- (NSString *)attribute
```

Return Value
The receiver's attribute name.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSMetadata.h`

count

Returns the number of instances of the value that exist for the attribute name of the receiver.

```
- (NSUInteger)count
```

Return Value
The number of instantes of the value that exist for the attribute name of the receiver.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSMetadata.h`

value

Returns the receiver's attribute value.

```
- (id)value
```

Return Value
The receiver's attribute value.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSMetadata.h`

NSMetadataQueryResultGroup Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSMetadata.h
Availability	Available in Mac OS X v10.4 and later.
Companion guide	Spotlight Query Programming Guide

Overview

The `NSMetadataQueryResultGroup` class represents a collection of grouped attribute results returned by an `NSMetadataQuery` object.

Tasks

Getting Query Results

- `attribute` (page 894)
  Returns the attribute name for the receiver's result group.
- `value` (page 895)
  Returns the value of the attribute name for the receiver.
- `results` (page 895)
  Returns an array containing the result objects for the receiver.
- `resultCount` (page 894)
  Returns the number of results returned by the receiver.
- `resultAtIndex:` (page 894)
  Returns the query result at a specific index.
- `subgroups` (page 895)
  Returns an array containing the subgroups of the receiver.

Instance Methods

attribute

Returns the attribute name for the receiver's result group.

```
- (NSString *)attribute
```

Return Value
The attribute name for the receiver's result group.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSMetadata.h`

resultAtIndex:

Returns the query result at a specific index.

```
- (id)resultAtIndex:(NSUInteger)index
```

Parameters
index
The index of the desired result.

Return Value
The query result at a specific index.

Discussion
For performance reasons, you should use this method when retrieving a specific result, rather than they array returned by `results` (page 895).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSMetadata.h`

resultCount

Returns the number of results returned by the receiver.

```
- (NSUInteger)resultCount
```

Return Value
The number of results returned by the receiver.

Discussion
For performance reasons, you should use this method, rather than invoking `count` on `results` (page 895).

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSMetadata.h

results

Returns an array containing the result objects for the receiver.

```
- (NSArray *)results
```

Return Value
An array containing the result objects for the receiver.

Discussion
The results array is a proxy object that is primarily intended for use with Cocoa bindings. While it is possible to copy the proxy array to get a "snapshot" of the complete current query results, it is generally not recommended due to performance and memory issues. To access individual result array elements you should instead use the `resultCount` and `resultAtIndex:` methods.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `resultCount` (page 894)
- `resultAtIndex:` (page 894)

Declared In
NSMetadata.h

subgroups

Returns an array containing the subgroups of the receiver.

```
- (NSArray *)subgroups
```

Return Value
An array containing the subgroups of the receiver.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSMetadata.h

value

Returns the value of the attribute name for the receiver.

```
- (id)value
```

Return Value
The value of the attribute name for the receiver.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSMetadata.h`

NSMethodSignature Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSMethodSignature.h
Companion guides	Distributed Objects Programming Topics The Objective-C 2.0 Programming Language

Overview

An `NSMethodSignature` object records type information for the arguments and return value of a method. It is used to forward messages that the receiving object does not respond to—most notably in the case of distributed objects. You typically create an `NSMethodSignature` object using `NSObject`'s `methodSignatureForSelector:` (page 1181) instance method (on Mac OS X v10.5 and later you can also use `signatureWithObjCTypes:` (page 898)). It is then used to create an `NSInvocation` object, which is passed as the argument to a `forwardInvocation:` (page 1177) message to send the invocation on to whatever other object can handle the message. In the default case, `NSObject` invokes `doesNotRecognizeSelector:` (page 1175), which raises an exception. For distributed objects, the `NSInvocation` object is encoded using the information in the `NSMethodSignature` object and sent to the real object represented by the receiver of the message.

An `NSMethodSignature` object presents its argument types by index with the `getArgumentTypeAtIndex:` (page 899) method. The hidden arguments for every method, *`self`* and *`_cmd`*, are at indices `0` and `1`, respectively. The arguments normally specified in a message invocation follow these. In addition to the argument types, an `NSMethodSignature` object offers the total number of arguments with `numberOfArguments` (page 901), the total stack frame length occupied by all arguments with `frameLength` (page 899) (this varies with hardware architecture), and the length and type of the return value with `methodReturnLength` (page 900) and `methodReturnType` (page 901). Finally, applications using distributed objects can determine if the method is asynchronous with the `isOneway` (page 900) method.

For more information about the nature of a method, including the hidden arguments, see "How Messaging Works" in *The Objective-C 2.0 Programming Language*.

Tasks

Creating a Method Signature Object

`+ signatureWithObjCTypes:` (page 898)
Returns an `NSMethodSignature` object for the given Objective C method type string.

Getting Information on Argument Types

`- getArgumentTypeAtIndex:` (page 899)
Returns the type encoding for the argument at a given index.

`- numberOfArguments` (page 901)
Returns the number of arguments recorded in the receiver.

`- frameLength` (page 899)
Returns the number of bytes that the arguments, taken together, occupy on the stack.

Getting Information on Return Types

`- methodReturnType` (page 901)
Returns a C string encoding the return type of the method in Objective-C type encoding.

`- methodReturnLength` (page 900)
Returns the number of bytes required for the return value.

Determining Synchronous Status

`- isOneway` (page 900)
Returns a Boolean value that indicates whether the receiver is asynchronous when invoked through distributed objects.

Class Methods

signatureWithObjCTypes:

Returns an `NSMethodSignature` object for the given Objective C method type string.

```
+ (NSMethodSignature *)signatureWithObjCTypes:(const char *)types
```

Parameters

types
An array of characters containing the type encodings for the method arguments.

Indices begin with `0`. The hidden arguments *self* (of type `id`) and *_cmd* (of type SEL) are at indices `0` and `1`; method-specific arguments begin at index `2`.

Return Value
An `NSMethodSignature` object for the given Objective C method type string in *types*.

Discussion

Special Considerations

This method, available since Mac OS X v10.0, is exposed in Mac OS X v10.5. Only type encoding strings of the style of the runtime that the application is running against are supported. In exposing this method there is no commitment to binary compatibily supporting any "old-style" type encoding strings after such changes occur.

It is your responsibility to pass in type strings which are either from the current runtime data or match the style of type string in use by the runtime that the application is running on.

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

Declared In
`NSMethodSignature.h`

Instance Methods

frameLength

Returns the number of bytes that the arguments, taken together, occupy on the stack.

```
- (NSUInteger)frameLength
```

Return Value
The number of bytes that the arguments, taken together, occupy on the stack.

Discussion
This number varies with the hardware architecture the application runs on.

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

Declared In
`NSMethodSignature.h`

getArgumentTypeAtIndex:

Returns the type encoding for the argument at a given index.

```
- (const char *)getArgumentTypeAtIndex:(NSUInteger)index
```

Parameters

index
The index of the argument to get.

Return Value
The type encoding for the argument at *index*.

Discussion
Indices begin with `0`. The hidden arguments *`self`* (of type `id`) and *`_cmd`* (of type SEL) are at indices `0` and `1`; method-specific arguments begin at index `2`. Raises `NSInvalidArgumentException` if *`index`* is too large for the actual number of arguments.

Argument types are given as C strings with Objective-C type encoding. This encoding is implementation-specific, so applications should use it with caution.

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

Declared In
`NSMethodSignature.h`

isOneway

Returns a Boolean value that indicates whether the receiver is asynchronous when invoked through distributed objects.

```
- (BOOL)isOneway
```

Return Value
`YES` if the receiver is asynchronous when invoked through distributed objects, otherwise `NO`.

Discussion
If the method is `oneway`, the sender of the remote message doesn't block awaiting a reply.

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

Declared In
`NSMethodSignature.h`

methodReturnLength

Returns the number of bytes required for the return value.

```
- (NSUInteger)methodReturnLength
```

Return Value
The number of bytes required for the return value.

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

See Also
`- methodReturnType` (page 901)

Declared In
`NSMethodSignature.h`

methodReturnType

Returns a C string encoding the return type of the method in Objective-C type encoding.

```
- (const char *)methodReturnType
```

Return Value
A C string encoding the return type of the method in Objective-C type encoding.

Discussion
This encoding is implementation-specific, so applications should use it with caution.

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

See Also
`- methodReturnLength` (page 900)

Declared In
`NSMethodSignature.h`

numberOfArguments

Returns the number of arguments recorded in the receiver.

```
- (NSUInteger)numberOfArguments
```

Return Value
The number of arguments recorded in the receiver.

Discussion
There are always at least `2` arguments, because an `NSMethodSignature` object includes the hidden arguments *`self`* and *`_cmd`*, which are the first two arguments passed to every method implementation.

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

Declared In
`NSMethodSignature.h`

NSMiddleSpecifier Class Reference

Inherits from	NSScriptObjectSpecifier : NSObject
Conforms to	NSCoding (NSScriptObjectSpecifier) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptObjectSpecifiers.h
Companion guide	Cocoa Scripting Guide

Overview

Specifies the middle object in a collection or, if not a one-to-many relationship, the sole object. You don't normally subclass `NSMiddleSpecifier`.

NSMoveCommand Class Reference

Inherits from	NSScriptCommand : NSObject
Conforms to	NSCoding (NSScriptCommand) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptStandardSuiteCommands.h
Companion guide	Cocoa Scripting Guide

Overview

An instance of `NSMoveCommand` moves the specified scriptable object or objects; for example, it may move words to a new location in a document or a file to a new directory.

`NSMoveCommand` is part of Cocoa's built-in scripting support. It works automatically to support the `move` AppleScript command through key-value coding. Most applications don't need to subclass `NSMoveCommand` or invoke its methods. However, for circumstances where you might choose to subclass this command, see "Modifying a Standard Command" in Script Commands in *Cocoa Scripting Guide*.

When an instance of `NSMoveCommand` is executed, it does not make copies of moved objects. It removes objects from the source container or containers, then inserts them into the destination container.

Tasks

Working with Specifiers

- `keySpecifier` (page 906)
  Returns a specifier for the object or objects to be moved.
- `setReceiversSpecifier:` (page 906)
  Sets the receiver's object specifier.

Instance Methods

keySpecifier

Returns a specifier for the object or objects to be moved.

```
- (NSScriptObjectSpecifier *)keySpecifier
```

Return Value
A specifier for the object or objects to be moved.

Discussion
Note that this specifier may be different than the specifier set by `setReceiversSpecifier:` (page 906), which sets the container specifier. For example, for a command such as `move the third circle to the location of the first circle`, the receiver might identify a document (which has a list of graphics), while the key specifier identifies the particular graphic to be moved.

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

Declared In
`NSScriptStandardSuiteCommands.h`

setReceiversSpecifier:

Sets the receiver's object specifier.

```
- (void)setReceiversSpecifier:(NSScriptObjectSpecifier *)receiversRef
```

Parameters
receiversRef
The receiver's object specifier.

Discussion
When evaluated, *receiversRef* indicates the receiver or receivers of the `move` AppleScript command.

This method overrides `setReceiversSpecifier:` (page 1390) in `NSScriptCommand`. It performs the same function as the overridden method, with a critical difference: it causes the container specifier part of the passed-in object specifier to become the receiver specifier of the command, and the key part of the passed-in object specifier to become the key specifier. If, for example, *receiversRef* is a specifier for `the third paragraph of the first document`, the receiver specifier is `the first document` while the key specifier is `the third paragraph`.

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

Declared In
`NSScriptStandardSuiteCommands.h`

NSMutableArray Class Reference

Inherits from	NSArray : NSObject
Conforms to	NSCoding (NSArray) NSCopying (NSArray) NSMutableCopying (NSArray) NSFastEnumeration (NSArray) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSArray.h Foundation/NSPredicate.h Foundation/NSSortDescriptor.h
Companion guides	Collections Programming Topics for Cocoa Key-Value Coding Programming Guide
Related sample code	iSpend Quartz Composer WWDC 2005 TextEdit Sketch-112 StickiesExample TextEditPlus

Overview

The `NSMutableArray` class declares the programmatic interface to objects that manage a modifiable array of objects. This class adds insertion and deletion operations to the basic array-handling behavior inherited from `NSArray`.

`NSArray` and `NSMutableArray` are part of a class cluster, so arrays are not actual instances of the `NSArray` or `NSMutableArray` classes but of one of their private subclasses. Although an array's class is private, its interface is public, as declared by these abstract superclasses, `NSArray` and `NSMutableArray`. `NSMutableArray`'s methods are conceptually based on these primitive methods:

- `insertObject:atIndex:` (page 913)
- `removeObjectAtIndex:` (page 918)
- `addObject:` (page 911)
- `removeLastObject` (page 916)
- `replaceObjectAtIndex:withObject:` (page 922)

In a subclass, you must override all these methods, although you can implement the required functionality using just the first two (however this is likely to be inefficient).

The other methods in `NSMutableArray`'s interface provide convenient ways of inserting an object into a specific slot in the array and removing an object based on its identity or position in the array.

Like `NSArray`, instances of `NSMutableArray` maintain strong references to their contents. If you do not use garbage collection, when you add an object to an array, the object receives a `retain` (page 2108) message. When an object is removed from a mutable array, it receives a `release` (page 2106) message. If there are no further references to the object, this means that the object is deallocated. If your program keeps a reference to such an object, the reference will become invalid unless you send the object a `retain` (page 2108) message before it's removed from the array. For example, if `anObject` is not retained before it is removed from the array, the third statement below could result in a runtime error:

```
id anObject = [[anArray objectAtIndex:0] retain];
[anArray removeObjectAtIndex:0];
[anObject someMessage];
```

Mac OS X Note: The `filterUsingPredicate:` (page 912) method provides in-place in-memory filtering of an array using an `NSPredicate` object. If you use the Core Data framework, this provides an efficient means of filtering an existing array of objects without—as a fetch does—requiring a round trip to a persistent data store. This method and the `NSPredicate` class are not available in iPhone OS.

Tasks

Creating and Initializing a Mutable Array

`+ arrayWithCapacity:` (page 910)
Creates and returns an `NSMutableArray` object with enough allocated memory to initially hold a given number of objects.

`- initWithCapacity:` (page 913)
Returns an array, initialized with enough memory to initially hold a given number of objects.

Adding Objects

`- addObject:` (page 911)
Inserts a given object at the end of the receiver.

`- addObjectsFromArray:` (page 911)
Adds the objects contained in another given array to the end of the receiver's content.

`- insertObject:atIndex:` (page 913)
Inserts a given object into the receiver's contents at a given index.

`- insertObjects:atIndexes:` (page 914)
Inserts the objects in in a given array into the receiver at the specified indexes.

Removing Objects

- `removeAllObjects` (page 916)
  Empties the receiver of all its elements.
- `removeLastObject` (page 916)
  Removes the object with the highest-valued index in the receiver
- `removeObject:` (page 916)
  Removes all occurrences in the receiver of a given object.
- `removeObject:inRange:` (page 917)
  Removes all occurrences within a specified range in the receiver of a given object.
- `removeObjectAtIndex:` (page 918)
  Removes the object at *index*.
- `removeObjectsAtIndexes:` (page 920)
  Removes the objects at the specified indexes from the receiver.
- `removeObjectIdenticalTo:` (page 919)
  Removes all occurrences of a given object in the receiver.
- `removeObjectIdenticalTo:inRange:` (page 919)
  Removes all occurrences of *anObject* within the specified range in the receiver.
- `removeObjectsFromIndices:numIndices:` (page 921)
  Removes the specified number of objects from the receiver, beginning at the specified index.
- `removeObjectsInArray:` (page 921)
  Removes from the receiver the objects in another given array.
- `removeObjectsInRange:` (page 922)
  Removes from the receiver each of the objects within a given range.

Replacing Objects

- `replaceObjectAtIndex:withObject:` (page 922)
  Replaces the object at *index* with *anObject*.
- `replaceObjectsAtIndexes:withObjects:` (page 923)
  Replaces the objects in the receiver at specified locations specified with the objects from a given array.
- `replaceObjectsInRange:withObjectsFromArray:range:` (page 924)
  Replaces the objects in the receiver specified by one given range with the objects in another array specified by another range.
- `replaceObjectsInRange:withObjectsFromArray:` (page 924)
  Replaces the objects in the receiver specified by a given range with all of the objects from a given array.
- `setArray:` (page 925)
  Sets the receiver's elements to those in another given array.

Filtering Content

- `filterUsingPredicate:` (page 912)
  Evaluates a given predicate against the receiver's content and leaves only objects that match

Rearranging Content

- `exchangeObjectAtIndex:withObjectAtIndex:` (page 912)
  Exchanges the objects in the receiver at given indices.
- `sortUsingDescriptors:` (page 925)
  Sorts the receiver using a given array of sort descriptors.
- `sortUsingFunction:context:` (page 926)
  Sorts the receiver's elements in ascending order as defined by the comparison function *compare*.
- `sortUsingSelector:` (page 926)
  Sorts the receiver's elements in ascending order, as determined by the comparison method specified by a given selector.

Class Methods

arrayWithCapacity:

Creates and returns an `NSMutableArray` object with enough allocated memory to initially hold a given number of objects.

```
+ (id)arrayWithCapacity:(NSUInteger)numItems
```

Parameters

numItems
  The initial capacity of the new array.

Return Value
A new `NSMutableArray` object with enough allocated memory to hold *numItems* objects.

Discussion
Mutable arrays expand as needed; *numItems* simply establishes the object's initial capacity.

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

See Also
- `initWithCapacity:` (page 913)

Related Sample Code
Birthdays
EnhancedAudioBurn
Fiendishthngs
Sketch-112
TimelineToTC

Declared In
NSArray.h

Instance Methods

addObject:

Inserts a given object at the end of the receiver.

```
- (void)addObject:(id)anObject
```

Parameters

anObject
The object to add to the end of the receiver's content. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *anObject* is `nil`.

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

See Also
- `addObjectsFromArray:` (page 911)
- `removeObject:` (page 916)
- `setArray:` (page 925)

Related Sample Code
CoreRecipes
Quartz Composer WWDC 2005 TextEdit
Sketch-112
StickiesExample
TextEditPlus

Declared In
NSArray.h

addObjectsFromArray:

Adds the objects contained in another given array to the end of the receiver's content.

```
- (void)addObjectsFromArray:(NSArray *)otherArray
```

Parameters

otherArray
An array of objects to add to the end of the receiver's content.

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

See Also
- `setArray:` (page 925)
- `removeObject:` (page 916)

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
SimpleCalendar
Sketch-112
StickiesExample
TextEditPlus

Declared In
`NSArray.h`

exchangeObjectAtIndex:withObjectAtIndex:

Exchanges the objects in the receiver at given indices.

```
- (void)exchangeObjectAtIndex:(NSUInteger)idx1 withObjectAtIndex:(NSUInteger)idx2
```

Parameters

idx1
The index of the object with which to replace the object at index *idx2*.

idx2
The index of the object with which to replace the object at index *idx1*.

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

Declared In
`NSArray.h`

filterUsingPredicate:

Evaluates a given predicate against the receiver's content and leaves only objects that match

```
- (void)filterUsingPredicate:(NSPredicate *)predicate
```

Parameters

predicate
The predicate to evaluate against the receiver's elements.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `filteredArrayUsingPredicate:` (page 121) (`NSArray`)

Declared In
`NSPredicate.h`

initWithCapacity:

Returns an array, initialized with enough memory to initially hold a given number of objects.

```
- (id)initWithCapacity:(NSUInteger)numItems
```

Parameters

numItems

The initial capacity of the new array.

Return Value

An array initialized with enough memory to hold *numItems* objects. The returned object might be different than the original receiver.

Discussion

Mutable arrays expand as needed; *numItems* simply establishes the object's initial capacity.

Availability

Available in Mac OS X v10.0 and later.

See Also

+ arrayWithCapacity: (page 910)

Declared In

NSArray.h

insertObject:atIndex:

Inserts a given object into the receiver's contents at a given index.

```
- (void)insertObject:(id)anObject  atIndex:(NSUInteger)index
```

Parameters

anObject

The object to add to the receiver's content. This value must not be nil.

Important: Raises an NSInvalidArgumentException if *anObject* is nil.

index

The index in the receiver at which to insert *anObject*. This value must not be greater than the count of elements in the array.

Important: Raises an NSRangeException if *index* is greater than the number of elements in the array.

Discussion

If *index* is already occupied, the objects at *index* and beyond are shifted by adding 1 to their indices to make room.

Note that `NSArray` objects are not like C arrays. That is, even though you specify a size when you create an array, the specified size is regarded as a "hint"; the actual size of the array is still 0. This means that you cannot insert an object at an index greater than the current count of an array. For example, if an array contains two objects, its size is 2, so you can add objects at indices 0, 1, or 2. Index 3 is illegal and out of bounds; if you try to add an object at index 3 (when the size of the array is 2), `NSMutableArray` raises an exception.

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

See Also
`- removeObjectAtIndex:` (page 918)

Related Sample Code
SimpleCocoaMovie
SimpleCocoaMovieQT
SimpleScriptingObjects
ThreadsExporter
WhackedTV

Declared In
`NSArray.h`

insertObjects:atIndexes:

Inserts the objects in in a given array into the receiver at the specified indexes.

```
- (void)insertObjects:(NSArray *)objects  atIndexes:(NSIndexSet *)indexes
```

Parameters

objects
An array of objects to insert into the receiver.

indexes
The indexes at which the objects in *objects* should be inserted. The count of locations in *indexes* must equal the count of *objects*. For more details, see the Discussion.

Discussion
Each object in *objects* is inserted into the receiver in turn at the corresponding location specified in *indexes* after earlier insertions have been made. The implementation is conceptually similar to that illustrated in the following example.

```
- void insertObjects:(NSArray *)additions atIndexes:(NSIndexSet *)indexes
{
    NSUInteger currentIndex = [indexes firstIndex];
    NSUInteger i, count = [indexes count];

    for (i = 0; i < count; i++)
    {
        [self insertObject:[additions objectAtIndex:i] atIndex:currentIndex];
        currentIndex = [indexes indexGreaterThanIndex:currentIndex];
    }
}
```

The resulting behavior is illustrated by the following example.

```
NSMutableArray *array = [NSMutableArray arrayWithObjects: @"one", @"two",
@"three", @"four", nil];
NSArray *newAdditions = [NSArray arrayWithObjects: @"a", @"b", nil];
NSMutableIndexSet *indexes = [NSMutableIndexSet indexSetWithIndex:1];
[indexes addIndex:3];
[array insertObjects:newAdditions atIndexes:indexes];
NSLog(@"array: %@", array);

// Output: array: (one, a, two, b, three, four)
```

The locations specified by *indexes* may therefore only exceed the bounds of the receiver if one location specifies the count of the array or the count of the array after preceding insertions, and other locations exceeding the bounds do so in a contiguous fashion from that location, as illustrated in the following examples.

In this example, both new objects are appended to the end of the array.

```
NSMutableArray *array = [NSMutableArray arrayWithObjects: @"one", @"two",
@"three", @"four", nil];
NSArray *newAdditions = [NSArray arrayWithObjects: @"a", @"b", nil];
NSMutableIndexSet *indexes = [NSMutableIndexSet indexSetWithIndex:5];
[indexes addIndex:4];
[array insertObjects:newAdditions atIndexes:indexes];
NSLog(@"array: %@", array);

// Output: array: (one, two, three, four, a, b)
```

If you replace `[indexes addIndex:4]` with `[indexes addIndex:6]` (so that the indexes are 5 and 6), then the application will fail with an out of bounds exception.

In this example, two objects are added into the middle of the array, and another at the current end of the array (index 4) which means that it is third from the end of the modified array.

```
NSMutableArray *array = [NSMutableArray arrayWithObjects: @"one", @"two",
@"three", @"four", nil];
NSArray *newAdditions = [NSArray arrayWithObjects: @"a", @"b", @"c", nil];
NSMutableIndexSet *indexes = [NSMutableIndexSet indexSetWithIndex:1];
[indexes addIndex:2];
[indexes addIndex:4];
[array insertObjects:newAdditions atIndexes:indexes];
NSLog(@"array: %@", array);

// Output: array: (one, a, b, two, c, three, four)
```

If you replace `[indexes addIndex:4]` with `[indexes addIndex:6]` (so that the indexes are 1, 2, and 6), then the output is `(one, a, b, two, three, four, c)`.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- insertObject:atIndex:` (page 913)

Declared In
`NSArray.h`

removeAllObjects

Empties the receiver of all its elements.

```
- (void)removeAllObjects
```

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

See Also
- `removeObject:` (page 916)
- `removeLastObject` (page 916)
- `removeObjectAtIndex:` (page 918)
- `removeObjectIdenticalTo:` (page 919)

Related Sample Code
ABPresence
WhackedTV

Declared In
`NSArray.h`

removeLastObject

Removes the object with the highest-valued index in the receiver

```
- (void)removeLastObject
```

Discussion
`removeLastObject` raises an `NSRangeException` if there are no objects in the receiver.

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

See Also
- `removeAllObjects` (page 916)
- `removeObject:` (page 916)
- `removeObjectAtIndex:` (page 918)
- `removeObjectIdenticalTo:` (page 919)

Related Sample Code
WhackedTV

Declared In
`NSArray.h`

removeObject:

Removes all occurrences in the receiver of a given object.

```
- (void)removeObject:(id)anObject
```

Parameters

anObject

The object to remove from the receiver.

Discussion

This method uses `indexOfObject:` (page 123) to locate matches and then removes them by using `removeObjectAtIndex:` (page 918). Thus, matches are determined on the basis of an object's response to the `isEqual:` message. If the receiver does not contain *anObject*, the method has no effect (although it does incur the overhead of searching the contents).

Availability

Available in Mac OS X v10.0 and later.

See Also

- `removeAllObjects` (page 916)
- `removeLastObject` (page 916)
- `removeObjectAtIndex:` (page 918)
- `removeObjectIdenticalTo:` (page 919)
- `removeObjectsInArray:` (page 921)

Related Sample Code

CoreRecipes

GLChildWindowDemo

Squiggles

WhackedTV

Declared In

`NSArray.h`

removeObject:inRange:

Removes all occurrences within a specified range in the receiver of a given object.

```
- (void)removeObject:(id)anObject  inRange:(NSRange)aRange
```

Parameters

anObject

The object to remove from the receiver's content.

aRange

The range from which to remove *anObject*.

Important: Raises an `NSRangeException` if *aRange* exceeds the bounds of the receiver.

Discussion

Matches are determined on the basis of an object's response to the `isEqual:` message. If the receiver does not contain *anObject* within *aRange*, the method has no effect (although it does incur the overhead of searching the contents).

Availability

Available in Mac OS X v10.0 and later.

See Also
- `removeAllObjects` (page 916)
- `removeLastObject` (page 916)
- `removeObjectAtIndex:` (page 918)
- `removeObjectIdenticalTo:` (page 919)
- `removeObjectsInArray:` (page 921)

Declared In
`NSArray.h`

removeObjectAtIndex:

Removes the object at *index*.

```
- (void)removeObjectAtIndex:(NSUInteger)index
```

Parameters

index

The index from which to remove the object in the receiver. The value must not exceed the bounds of the receiver.

Important: Raises an `NSRangeException` if *index* is beyond the end of the receiver.

Discussion
To fill the gap, all elements beyond *index* are moved by subtracting 1 from their index.

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

See Also
- `insertObject:atIndex:` (page 913)
- `removeAllObjects` (page 916)
- `removeLastObject` (page 916)
- `removeObject:` (page 916)
- `removeObjectIdenticalTo:` (page 919)
- `removeObjectsFromIndices:numIndices:` (page 921)

Related Sample Code
EnhancedAudioBurn
EnhancedDataBurn
ImageBackground
QTKitMovieShuffler
SimpleScriptingObjects

Declared In
`NSArray.h`

removeObjectIdenticalTo:

Removes all occurrences of a given object in the receiver.

```
- (void)removeObjectIdenticalTo:(id)anObject
```

Parameters

anObject

The object to remove from the receiver.

Discussion

This method uses the `indexOfObjectIdenticalTo:` (page 124) method to locate matches and then removes them by using `removeObjectAtIndex:` (page 918). Thus, matches are determined using object addresses. If the receiver does not contain *anObject*, the method has no effect (although it does incur the overhead of searching the contents).

Availability

Available in Mac OS X v10.0 and later.

See Also

- `removeAllObjects` (page 916)
- `removeLastObject` (page 916)
- `removeObject:` (page 916)
- `removeObjectAtIndex:` (page 918)

Related Sample Code

EnhancedDataBurn

ImageBackground

QTKitMovieShuffler

TrackBall

Declared In

`NSArray.h`

removeObjectIdenticalTo:inRange:

Removes all occurrences of *anObject* within the specified range in the receiver.

```
- (void)removeObjectIdenticalTo:(id)anObject  inRange:(NSRange)aRange
```

Parameters

anObject

The object to remove from the receiver within *aRange*.

aRange

The range in the receiver from which to remove *anObject*.

Important: Raises an `NSRangeException` if *aRange* exceeds the bounds of the receiver.

Discussion
This method uses the `indexOfObjectIdenticalTo:` (page 124) method to locate matches and then removes them by using `removeObjectAtIndex:` (page 918). Thus, matches are determined using object addresses. If the receiver does not contain *anObject* within *aRange*, the method has no effect (although it does incur the overhead of searching the contents).

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

See Also
- `removeAllObjects` (page 916)
- `removeLastObject` (page 916)
- `removeObject:` (page 916)
- `removeObjectAtIndex:` (page 918)
- `removeObjectsAtIndexes:` (page 920)

Declared In
`NSArray.h`

removeObjectsAtIndexes:

Removes the objects at the specified indexes from the receiver.

```
- (void)removeObjectsAtIndexes:(NSIndexSet *)indexes
```

Parameters
indexes

The indexes of the objects to remove from the receiver. The locations specified by *indexes* must lie within the bounds of the receiver.

Discussion
This method is similar to `removeObjectAtIndex:` (page 918), but allows you to efficiently remove multiple objects with a single operation. *indexes* specifies the locations of objects to be removed given the state of the receiver when the method is invoked, as illustrated in the following example.

```
NSMutableArray *array = [NSMutableArray arrayWithObjects: @"one", @"a", @"two",
 @"b", @"three", @"four", nil];
NSMutableIndexSet *indexes = [NSMutableIndexSet indexSetWithIndex:1];
[indexes addIndex:3];
[array removeObjectsAtIndexes:indexes];
NSLog(@"array: %@", array);

// Output: array: (one, two, three, four)
```

Availability
Available in Mac OS X v10.4 and later.

See Also
- `initWithCapacity:` (page 913)
- `removeObjectAtIndex:` (page 918)
- `removeObject:inRange:` (page 917)

Declared In
`NSArray.h`

removeObjectsFromIndices:numIndices:

Removes the specified number of objects from the receiver, beginning at the specified index.

```
- (void)removeObjectsFromIndices:(NSUInteger *)indices numIndices:(NSUInteger)count
```

Parameters

indices
A C array of the indices of the objects to remove from the receiver.

count
The number of objects to remove from the receiver.

Discussion
This method is similar to `removeObjectAtIndex:` (page 918), but allows you to efficiently remove multiple objects with a single operation. If you sort the list of indices in ascending order, you will improve the speed of this operation.

This method cannot be sent to a remote object with distributed objects.

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

See Also
- `initWithCapacity:` (page 913)
- `removeObjectAtIndex:` (page 918)
- `removeObject:inRange:` (page 917)
- `removeObjectsAtIndexes:` (page 920)

Declared In
`NSArray.h`

removeObjectsInArray:

Removes from the receiver the objects in another given array.

```
- (void)removeObjectsInArray:(NSArray *)otherArray
```

Parameters

otherArray
An array containing the objects to be removed from the receiver.

Discussion
This method is similar to `removeObject:` (page 916), but allows you to efficiently remove large sets of objects with a single operation. If the receiver does not contain objects in *otherArray*, the method has no effect (although it does incur the overhead of searching the contents).

This method assumes that all elements in *otherArray* respond to `hash` and `isEqual:`.

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

See Also
- `removeAllObjects` (page 916)
- `removeObjectIdenticalTo:` (page 919)
- `removeObjectsAtIndexes:` (page 920)

Related Sample Code
QTKitAdvancedDocument
SimpleCalendar
StickiesExample

Declared In
`NSArray.h`

removeObjectsInRange:

Removes from the receiver each of the objects within a given range.

```
- (void)removeObjectsInRange:(NSRange)aRange
```

Parameters
aRange
The range of the objects to remove from the receiver.

Discussion
The objects are removed using `removeObjectAtIndex:` (page 918).

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

Declared In
`NSArray.h`

replaceObjectAtIndex:withObject:

Replaces the object at *index* with *anObject*.

```
- (void)replaceObjectAtIndex:(NSUInteger)index  withObject:(id)anObject
```

Parameters

index

The index of the object to be replaced. This value must not exceed the bounds of the receiver.

Important: Raises an `NSRangeException` if *index* is beyond the end of the receiver.

anObject

The object with which to replace the object at index *index* in the receiver. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *anObject* is `nil`.

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

See Also
- `insertObject:atIndex:` (page 913)
- `removeObjectAtIndex:` (page 918)
- `removeObjectsAtIndexes:` (page 920)
- `replaceObjectsAtIndexes:withObjects:` (page 923)

Related Sample Code
ABPresence
TrackBall

Declared In
`NSArray.h`

replaceObjectsAtIndexes:withObjects:

Replaces the objects in the receiver at specified locations specified with the objects from a given array.

```
- (void)replaceObjectsAtIndexes:(NSIndexSet *)indexes  withObjects:(NSArray *)objects
```

Parameters

indexes

The indexes of the objects to be replaced.

objects

The objects with which to replace the objects in the receiver at the indexes specified by *indexes*. The count of locations in *indexes* must equal the count of *objects*.

Discussion
The indexes in *indexes* are used in the same order as the objects in *objects*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `insertObject:atIndex:` (page 913)
- `removeObjectAtIndex:` (page 918)

- replaceObjectAtIndex:withObject: (page 922)

Declared In
NSArray.h

replaceObjectsInRange:withObjectsFromArray:

Replaces the objects in the receiver specified by a given range with all of the objects from a given array.

```
- (void)replaceObjectsInRange:(NSRange)aRange  withObjectsFromArray:(NSArray
    *)otherArray
```

Parameters

aRange

The range of objects to replace in (or remove from) the receiver.

otherArray

The array of objects from which to select replacements for the objects in *aRange*.

Discussion
If *otherArray* has fewer objects than are specified by *aRange*, the extra objects in the receiver are removed. If *otherArray* has more objects than are specified by *aRange*, the extra objects from *otherArray* are inserted into the receiver.

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

See Also
- insertObject:atIndex: (page 913)
- removeObjectAtIndex: (page 918)
- replaceObjectAtIndex:withObject: (page 922)
- replaceObjectsAtIndexes:withObjects: (page 923)

Declared In
NSArray.h

replaceObjectsInRange:withObjectsFromArray:range:

Replaces the objects in the receiver specified by one given range with the objects in another array specified by another range.

```
- (void)replaceObjectsInRange:(NSRange)aRange  withObjectsFromArray:(NSArray
    *)otherArray  range:(NSRange)otherRange
```

Parameters

aRange

The range of objects to replace in (or remove from) the receiver.

otherArray

The array of objects from which to select replacements for the objects in *aRange*.

otherRange

The range of objects to select from *otherArray* as replacements for the objects in *aRange*.

Discussion
The lengths of *aRange* and *otherRange* don't have to be equal: if *aRange* is longer than *otherRange*, the extra objects in the receiver are removed; if *otherRange* is longer than *aRange*, the extra objects from *otherArray* are inserted into the receiver.

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

See Also
- `insertObject:atIndex:` (page 913)
- `removeObjectAtIndex:` (page 918)
- `replaceObjectAtIndex:withObject:` (page 922)
- `replaceObjectsAtIndexes:withObjects:` (page 923)

Declared In
`NSArray.h`

setArray:

Sets the receiver's elements to those in another given array.

```
- (void)setArray:(NSArray *)otherArray
```

Parameters
otherArray
The array of objects with which to replace the receiver's content.

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

See Also
- `addObjectsFromArray:` (page 911)
- `insertObject:atIndex:` (page 913)

Declared In
`NSArray.h`

sortUsingDescriptors:

Sorts the receiver using a given array of sort descriptors.

```
- (void)sortUsingDescriptors:(NSArray *)sortDescriptors
```

Parameters
sortDescriptors
An array containing the `NSSortDescriptor` objects to use to sort the receiver's contents.

Discussion
See `NSSortDescriptor` for additional information.

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

See Also
- `sortUsingFunction:context:` (page 926)
- `sortUsingSelector:` (page 926)
- `sortedArrayUsingDescriptors:` (page 135) (`NSArray`)

Related Sample Code
CoreRecipes

Declared In
`NSSortDescriptor.h`

sortUsingFunction:context:

Sorts the receiver's elements in ascending order as defined by the comparison function *compare*.

```
- (void)sortUsingFunction:(NSInteger (*)(id, id, void *))compare  context:(void
    *)context
```

Parameters

compare

The comparison function to use to compare two elements at a time.

The function's parameters are two objects to compare and the context parameter, *context*. The function should return `NSOrderedAscending` if the first element is smaller than the second, `NSOrderedDescending` if the first element is larger than the second, and `NSOrderedSame` if the elements are equal.

context

The context argument to pass to the compare function.

Discussion
This approach allows the comparison to be based on some outside parameter, such as whether character sorting is case-sensitive or case-insensitive.

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

See Also
- `sortUsingDescriptors:` (page 925)
- `sortUsingSelector:` (page 926)
- `sortedArrayUsingFunction:context:` (page 136) (`NSArray`)

Related Sample Code
Reminders

Declared In
`NSArray.h`

sortUsingSelector:

Sorts the receiver's elements in ascending order, as determined by the comparison method specified by a given selector.

```
- (void)sortUsingSelector:(SEL)comparator
```

Parameters

comparator

A selector that specifies the comparison method to use to compare elements in the receiver.

The *comparator* message is sent to each object in the receiver and has as its single argument another object in the array. The *comparator* method should return `NSOrderedAscending` if the receiver is smaller than the argument, `NSOrderedDescending` if the receiver is larger than the argument, and `NSOrderedSame` if they are equal.

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

See Also
- `sortUsingDescriptors:` (page 925)
- `sortUsingFunction:context:` (page 926)
- `sortedArrayUsingSelector:` (page 138) (`NSArray`)

Related Sample Code
ABPresence
SearchField

Declared In
`NSArray.h`

NSMutableAttributedString Class Reference

Inherits from	NSAttributedString : NSObject
Conforms to	NSCoding (NSAttributedString) NSCopying (NSAttributedString) NSMutableCopying (NSAttributedString) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSAttributedString.h
Companion guide	Attributed Strings Programming Guide
Related sample code	CocoaVideoFrameToGWorld CoreRecipes iSpend NSGLImage OpenGL Screensaver

Overview

`NSMutableAttributedString` declares the programmatic interface to objects that manage mutable attributed strings. You can add and remove characters (raw strings) and attributes separately or together as attributed strings. See the class description for NSAttributedString for more information about attributed strings.

When working with the Application Kit, you must also clean up changed attributes using the various `fix...` methods. See "Changing an Attributed String" for more information on fixing attributes. These methods, as well as others involving setting graphical attributes, are described in NSMutableAttributedString Additions in the Application Kit.

`NSMutableAttributedString` adds two primitive methods to those of `NSAttributedString`. These primitive methods provide the basis for all the other methods in its class. The primitive `replaceCharactersInRange:withString:` (page 936) method replaces a range of characters with those from a string, leaving all attribute information outside that range intact. The primitive `setAttributes:range:` (page 937) method sets attributes and values for a given range of characters, replacing any previous attributes and values for that range.

The Application Kit also uses NSParagraphStyle and its subclass NSMutableParagraphStyle to encapsulate the paragraph or ruler attributes used by the `NSAttributedString` classes.

Note that the default font for `NSAttributedString` objects is Helvetica 12-point, which differs from the Mac OS X system font Lucida Grande, so you may wish to create the string with non-default attributes suitable for your application using, for example, `initWithString:attributes:` (page 154).

Tasks

Retrieving Character Information

- `mutableString` (page 935)
  Returns the character contents of the receiver as an `NSMutableString` object.

Changing Characters

- `replaceCharactersInRange:withString:` (page 936)
  Replaces the characters in the given range with the characters of the given string.
- `deleteCharactersInRange:` (page 933)
  Deletes the characters in the given range along with their associated attributes.

Changing Attributes

- `setAttributes:range:` (page 937)
  Sets the attributes for the characters in the specified range to the specified attributes.
- `addAttribute:value:range:` (page 931)
  Adds an attribute with the given name and value to the characters in the specified range.
- `addAttributes:range:` (page 932)
  Adds the given collection of attributes to the characters in the specified range.
- `removeAttribute:range:` (page 935)
  Removes the named attribute from the characters in the specified range.

Changing Characters and Attributes

- `appendAttributedString:` (page 932)
  Adds the characters and attributes of a given attributed string to the end of the receiver.
- `insertAttributedString:atIndex:` (page 934)
  Inserts the characters and attributes of the given attributed string into the receiver at the given index.
- `replaceCharactersInRange:withAttributedString:` (page 936)
  Replaces the characters and attributes in a given range with the characters and attributes of the given attributed string.
- `setAttributedString:` (page 937)
  Replaces the receiver's entire contents with the characters and attributes of the given attributed string.

Grouping Changes

- `beginEditing` (page 933)
  Overridden by subclasses to buffer or optimize a series of changes to the receiver's characters or attributes, until it receives a matching `endEditing` (page 934) message, upon which it can consolidate changes and notify any observers that it has changed.
- `endEditing` (page 934)
  Overridden by subclasses to consolidate changes made since a previous `beginEditing` (page 933) message and to notify any observers of the changes.

Instance Methods

addAttribute:value:range:

Adds an attribute with the given name and value to the characters in the specified range.

```
- (void)addAttribute:(NSString *)name value:(id)value range:(NSRange)aRange
```

Parameters

name
  A string specifying the attribute name.

value
  The attribute value associated with *name*.

aRange
  The range of characters to which the specified attribute/value pair applies.

Discussion
You may assign any *name*/*value* pair you wish to a range of characters, in addition to the standard attributes described in the "Constants" section of NSAttributedString Additions. Raises an `NSInvalidArgumentException` if *name* or *value* is `nil` and an `NSRangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

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

See Also
- `addAttributes:range:` (page 932)
- `removeAttribute:range:` (page 935)

Related Sample Code
CoreRecipes
IBFragmentView
iSpend
TipWrapper

Declared In
`NSAttributedString.h`

addAttributes:range:

Adds the given collection of attributes to the characters in the specified range.

```
- (void)addAttributes:(NSDictionary *)attributes range:(NSRange)aRange
```

Parameters

attributes

A dictionary containing the attributes to add.

aRange

The range of characters to which the specified attributes apply.

Discussion

You may assign any name/value pair you wish to a range of characters, in addition to the standard attributes described in the "Constants" section of NSAttributedString Additions. Raises an `NSInvalidArgumentException` if *attributes* is `nil` and an `NSRangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- addAttribute:value:range:` (page 931)

`- removeAttribute:range:` (page 935)

Related Sample Code

TextLinks

VertexPerformanceTest

Declared In

`NSAttributedString.h`

appendAttributedString:

Adds the characters and attributes of a given attributed string to the end of the receiver.

```
- (void)appendAttributedString:(NSAttributedString *)attributedString
```

Parameters

attributedString

The string whose characters and attributes are added.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- insertAttributedString:atIndex:` (page 934)

`+ attributedStringWithAttachment:` (NSAttributedString Additions)

Related Sample Code

BackgroundExporter

CoreRecipes

iSpend

OpenGL Screensaver

Declared In
NSAttributedString.h

beginEditing

Overridden by subclasses to buffer or optimize a series of changes to the receiver's characters or attributes, until it receives a matching `endEditing` (page 934) message, upon which it can consolidate changes and notify any observers that it has changed.

```
- (void)beginEditing
```

Discussion
You can nest pairs of `beginEditing` and `endEditing` (page 934) messages.

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

Related Sample Code
CoreRecipes
Quartz Composer WWDC 2005 TextEdit
TextEditPlus
VertexPerformanceTest

Declared In
NSAttributedString.h

deleteCharactersInRange:

Deletes the characters in the given range along with their associated attributes.

```
- (void)deleteCharactersInRange:(NSRange)aRange
```

Parameters
aRange
A range specifying the characters to delete.

Discussion
Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

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

See Also
- `replaceCharactersInRange:withAttributedString:` (page 936)
- `replaceCharactersInRange:withString:` (page 936)

Declared In
NSAttributedString.h

endEditing

Overridden by subclasses to consolidate changes made since a previous `beginEditing` (page 933) message and to notify any observers of the changes.

```
- (void)endEditing
```

Discussion
The `NSMutableAttributedString` implementation does nothing. `NSTextStorage`, for example, overrides this method to invoke `fixAttributesInRange:` and to inform its `NSLayoutManager` objects that they need to re-lay the text.

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

See Also
`- processEditing` (`NSTextStorage`)

Related Sample Code
CoreRecipes
Quartz Composer WWDC 2005 TextEdit
TextEditPlus
VertexPerformanceTest

Declared In
`NSAttributedString.h`

insertAttributedString:atIndex:

Inserts the characters and attributes of the given attributed string into the receiver at the given index.

```
- (void)insertAttributedString:(NSAttributedString *)attributedString
    atIndex:(NSUInteger)index
```

Parameters

attributedString
The string whose characters and attributes are inserted.

index
The index at which the characters and attributes are inserted.

Discussion
The new characters and attributes begin at the given index and the existing characters and attributes from the index to the end of the receiver are shifted by the length of the attributed string. Raises an `NSRangeException` if *index* lies beyond the end of the receiver's characters.

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

See Also
`- appendAttributedString:` (page 932)
`+ attributedStringWithAttachment:` (NSAttributedString Additions)

Related Sample Code
CoreRecipes

Declared In
NSAttributedString.h

mutableString

Returns the character contents of the receiver as an `NSMutableString` object.

```
- (NSMutableString *)mutableString
```

Return Value
The mutable string object.

Discussion
The receiver tracks changes to this string and keeps its attribute mappings up to date.

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

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
NSAttributedString.h

removeAttribute:range:

Removes the named attribute from the characters in the specified range.

```
- (void)removeAttribute:(NSString *)name range:(NSRange)aRange
```

Parameters

name
A string specifying the attribute name to remove.

aRange
The range of characters from which the specified attribute is removed.

Discussion
Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

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

See Also
- `addAttribute:value:range:` (page 931)
- `addAttributes:range:` (page 932)

Declared In
NSAttributedString.h

replaceCharactersInRange:withAttributedString:

Replaces the characters and attributes in a given range with the characters and attributes of the given attributed string.

```
- (void)replaceCharactersInRange:(NSRange)aRange
    withAttributedString:(NSAttributedString *)attributedString
```

Parameters

aRange

The range of characters and attributes replaced.

attributedString

The attributed string whose characters and attributes replace those in the specified range.

Discussion

Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- insertAttributedString:atIndex:` (page 934)

Declared In

`NSAttributedString.h`

replaceCharactersInRange:withString:

Replaces the characters in the given range with the characters of the given string.

```
- (void)replaceCharactersInRange:(NSRange)aRange withString:(NSString *)aString
```

Parameters

aRange

A range specifying the characters to replace.

aString

A string specifying the characters to replace those in `aRange`.

Discussion

The new characters inherit the attributes of the first replaced character from *aRange*. Where the length of *aRange* is 0, the new characters inherit the attributes of the character preceding *aRange* if it has any, otherwise of the character following *aRange*.

Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- deleteCharactersInRange:` (page 933)

Related Sample Code

iSpend

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In
NSAttributedString.h

setAttributedString:

Replaces the receiver's entire contents with the characters and attributes of the given attributed string.

```
- (void)setAttributedString:(NSAttributedString *)attributedString
```

Parameters

attributedString
The attributed string whose characters and attributes replace those in the receiver.

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

See Also
- appendAttributedString: (page 932)

Declared In
NSAttributedString.h

setAttributes:range:

Sets the attributes for the characters in the specified range to the specified attributes.

```
- (void)setAttributes:(NSDictionary *)attributes range:(NSRange)aRange
```

Parameters

attributes
A dictionary containing the attributes to set.

aRange
The range of characters whose attributes are set.

Discussion
These new attributes replace any attributes previously associated with the characters in *aRange*. Raises an NSRangeException if any part of *aRange* lies beyond the end of the receiver's characters.

To set attributes for a zero-length NSMutableAttributedString displayed in a text view, use the NSTextView method setTypingAttributes:.

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

See Also
- addAttributes:range: (page 932)
- removeAttribute:range: (page 935)

Declared In
NSAttributedString.h

Constants

An attributed string identifies attributes by name, storing a value under the name in an `NSDictionary` object. You can assign any attribute name/value pair you wish to a range of characters, in addition to the standard attributes described in the "Constants" section of NSAttributedString Additions.

NSMutableCharacterSet Class Reference

Inherits from	NSCharacterSet : NSObject
Conforms to	NSCopying NSMutableCopying NSCoding (NSCharacterSet) NSCopying (NSCharacterSet) NSMutableCopying (NSCharacterSet) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSCharacterSet.h
Companion guide	String Programming Guide for Cocoa
Related sample code	ImageMapExample

Overview

The `NSMutableCharacterSet` class declares the programmatic interface to objects that manage a modifiable set of Unicode characters. You can add or remove characters from a mutable character set as numeric values in `NSRange` structures or as character values in strings, combine character sets by union or intersection, and invert a character set.

Mutable character sets are less efficient to use than immutable character sets. If you don't need to change a character set after creating it, create an immutable copy with `copy` and use that.

`NSMutableCharacterSet` defines no primitive methods. Subclasses must implement all methods declared by this class in addition to the primitives of `NSCharacterSet`. They must also implement `mutableCopyWithZone:` (page 2094).

Tasks

Adding and Removing Characters

`- addCharactersInRange:` (page 940)
Adds to the receiver the characters whose Unicode values are in a given range.

- `removeCharactersInRange:` (page 942)
  Removes from the receiver the characters whose Unicode values are in a given range.
- `addCharactersInString:` (page 941)
  Adds to the receiver the characters in a given string.
- `removeCharactersInString:` (page 943)
  Removes from the receiver the characters in a given string.

Combining Character Sets

- `formIntersectionWithCharacterSet:` (page 941)
  Modifies the receiver so it contains only characters that exist in both the receiver and *`otherSet`*.
- `formUnionWithCharacterSet:` (page 942)
  Modifies the receiver so it contains all characters that exist in either the receiver or *`otherSet`*.

Inverting a Character Set

- `invert` (page 942)
  Replaces all the characters in the receiver with all the characters it didn't previously contain.

Instance Methods

addCharactersInRange:

Adds to the receiver the characters whose Unicode values are in a given range.

```
- (void)addCharactersInRange:(NSRange)aRange
```

Parameters

`aRange`

The range of characters to add.

`aRange``.location` is the value of the first character to add; *`aRange`*`.location + `*`aRange`*`.length-1` is the value of the last. If *`aRange`*`.length` is `0`, this method has no effect.

Discussion

This code excerpt adds to a character set the lowercase English alphabetic characters:

```
NSMutableCharacterSet *aCharacterSet = [[NSMutableCharacterSet alloc] init];
NSRange lcEnglishRange;

lcEnglishRange.location = (unsigned int)'a';
lcEnglishRange.length = 26;
[aCharacterSet addCharactersInRange:lcEnglishRange];
```

Availability

Available in Mac OS X v10.0 and later.

See Also
- `removeCharactersInRange:` (page 942)
- `addCharactersInString:` (page 941)

Declared In
`NSCharacterSet.h`

addCharactersInString:

Adds to the receiver the characters in a given string.

```
- (void)addCharactersInString:(NSString *)aString
```

Parameters

aString

The characters to add to the receiver.

Discussion
This method has no effect if *aString* is empty.

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

See Also
- `removeCharactersInString:` (page 943)
- `addCharactersInRange:` (page 940)

Related Sample Code
ImageMapExample

Declared In
`NSCharacterSet.h`

formIntersectionWithCharacterSet:

Modifies the receiver so it contains only characters that exist in both the receiver and *otherSet*.

```
- (void)formIntersectionWithCharacterSet:(NSCharacterSet *)otherSet
```

Parameters

otherSet

The character set with which to perform the intersection.

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

See Also
- `formUnionWithCharacterSet:` (page 942)

Declared In
`NSCharacterSet.h`

formUnionWithCharacterSet:

Modifies the receiver so it contains all characters that exist in either the receiver or *otherSet*.

```
- (void)formUnionWithCharacterSet:(NSCharacterSet *)otherSet
```

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

See Also
- formIntersectionWithCharacterSet: (page 941)

Declared In
NSCharacterSet.h

invert

Replaces all the characters in the receiver with all the characters it didn't previously contain.

```
- (void)invert
```

Discussion
Inverting a mutable character set, whether by `invert` or by `invertedSet` (page 254), is much less efficient than inverting an immutable character set with `invertedSet`.

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

See Also
- invertedSet (page 254) (NSCharacterSet)

Declared In
NSCharacterSet.h

removeCharactersInRange:

Removes from the receiver the characters whose Unicode values are in a given range.

```
- (void)removeCharactersInRange:(NSRange)aRange
```

Parameters

aRange

The range of characters to remove.

aRange`.location` is the value of the first character to remove; *aRange*`.location + `*aRange*`.length- 1` is the value of the last. If *aRange*`.length` is `0`, this method has no effect.

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

See Also
- addCharactersInRange: (page 940)
- removeCharactersInString: (page 943)

Declared In
NSCharacterSet.h

removeCharactersInString:

Removes from the receiver the characters in a given string.

```
- (void)removeCharactersInString:(NSString *)aString
```

Parameters

aString
The characters to remove from the receiver.

Discussion
This method has no effect if *aString* is empty.

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

See Also
- addCharactersInString: (page 941)
- removeCharactersInRange: (page 942)

Declared In
NSCharacterSet.h

NSMutableData Class Reference

Inherits from	NSData : NSObject
Conforms to	NSCoding (NSData) NSCopying (NSData) NSMutableCopying (NSData) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSData.h Foundation/NSSerialization.h (Deprecated)
Companion guide	Binary Data Programming Guide for Cocoa
Related sample code	CocoaHTTPServer CocoaSOAP GridCalendar ImageClient URL CacheInfo

Overview

`NSMutableData` (and its superclass `NSData`) provide data objects, object-oriented wrappers for byte buffers. Data objects let simple allocated buffers (that is, data with no embedded pointers) take on the behavior of Foundation objects. They are typically used for data storage and are also useful in Distributed Objects applications, where data contained in data objects can be copied or moved between applications. `NSData` creates static data objects, and `NSMutableData` creates dynamic data objects. You can easily convert one type of data object to the other with the initializer that takes an `NSData` object or an `NSMutableData` object as an argument.

`NSMutableData` is "toll-free bridged" with its Core Foundation counterpart, `CFData`. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSMutableData *` parameter, you can pass a `CFDataRef`, and in a function where you see a `CFDataRef` parameter, you can pass an `NSMutableData` instance (you cast one type to the other to suppress compiler warnings). See Interchangeable Data Types for more information on toll-free bridging.

Tasks

Creating and Initializing an NSMutableData Object

`+ dataWithCapacity:` (page 947)
Creates and returns an `NSMutableData` object capable of holding the specified number of bytes.

`+ dataWithLength:` (page 947)
Creates and returns an `NSMutableData` object containing a given number of zeroed bytes.

`- initWithCapacity:` (page 949)
Returns an initialized `NSMutableData` object capable of holding the specified number of bytes.

`- initWithLength:` (page 950)
Initializes and returns an `NSMutableData` object containing a given number of zeroed bytes.

Adjusting Capacity

`- increaseLengthBy:` (page 949)
Increases the length of the receiver by a given number of bytes.

`- setLength:` (page 953)
Extends or truncates a mutable data object to a given length.

Accessing Data

`- mutableBytes` (page 950)
Returns a pointer to the receiver's data.

Adding Data

`- appendBytes:length:` (page 948)
Appends to the receiver a given number of bytes from a given buffer.

`- appendData:` (page 948)
Appends the content of another `NSData` object to the receiver.

Modifying Data

`- replaceBytesInRange:withBytes:` (page 951)
Replaces with a given set of bytes a given range within the contents of the receiver.

`- replaceBytesInRange:withBytes:length:` (page 951)
Replaces with a given set of bytes a given range within the contents of the receiver.

`- resetBytesInRange:` (page 952)
Replaces with zeroes the contents of the receiver in a given range.

`- setData:` (page 952)
Replaces the entire contents of the receiver with the contents of another data object.

Class Methods

dataWithCapacity:

Creates and returns an `NSMutableData` object capable of holding the specified number of bytes.

```
+ (id)dataWithCapacity:(NSUInteger)aNumItems
```

Parameters

aNumItems
The number of bytes the new data object can initially contain.

Return Value
A new `NSMutableData` object capable of holding *aNumItems* bytes.

Discussion
This method doesn't necessarily allocate the requested memory right away. Mutable data objects allocate additional memory as needed, so *aNumItems* simply establishes the object's initial capacity. When it does allocate the initial memory, though, it allocates the specified amount. This method sets the length of the data object to `0`.

If the capacity specified in *aNumItems* is greater than four memory pages in size, this method may round the amount of requested memory up to the nearest full page.

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

See Also
`+ dataWithLength:` (page 947)
`- initWithCapacity:` (page 949)
`- initWithLength:` (page 950)

Declared In
`NSData.h`

dataWithLength:

Creates and returns an `NSMutableData` object containing a given number of zeroed bytes.

```
+ (id)dataWithLength:(NSUInteger)length
```

Parameters

length
The number of bytes the new data object initially contains.

Return Value
A new `NSMutableData` object of *length* bytes, filled with zeros.

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

See Also
`+ dataWithCapacity:` (page 947)
`- initWithCapacity:` (page 949)
`- initWithLength:` (page 950)

Declared In
`NSData.h`

Instance Methods

appendBytes:length:

Appends to the receiver a given number of bytes from a given buffer.

```
- (void)appendBytes:(const void *)bytes length:(NSUInteger)length
```

Parameters

bytes
A buffer containing data to append to the receiver's content.

length
The number of bytes from *bytes* to append.

Discussion
A sample using this method can be found in Working With Mutable Binary Data.

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

See Also
`- appendData:` (page 948)

Related Sample Code
Core Data HTML Store
QTSSConnectionMonitor
QTSSInspector

Declared In
`NSData.h`

appendData:

Appends the content of another `NSData` object to the receiver.

```
- (void)appendData:(NSData *)otherData
```

Parameters

otherData

The data object whose content is to be appended to the contents of the receiver.

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

See Also
- appendBytes:length: (page 948)

Related Sample Code
GridCalendar

Declared In
NSData.h

increaseLengthBy:

Increases the length of the receiver by a given number of bytes.

```
- (void)increaseLengthBy:(NSUInteger)extraLength
```

Parameters

extraLength

The number of bytes by which to increase the receiver's length.

Discussion
The additional bytes are all set to 0.

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

See Also
- setLength: (page 953)

Declared In
NSData.h

initWithCapacity:

Returns an initialized NSMutableData object capable of holding the specified number of bytes.

```
- (id)initWithCapacity:(NSUInteger)capacity
```

Parameters

capacity

The number of bytes the data object can initially contain.

Return Value
An initialized NSMutableData object capable of holding *capacity* bytes.

Discussion
This method doesn't necessarily allocate the requested memory right away. Mutable data objects allocate additional memory as needed, so *aNumItems* simply establishes the object's initial capacity. When it does allocate the initial memory, though, it allocates the specified amount. This method sets the length of the data object to 0.

If the capacity specified in *aNumItems* is greater than four memory pages in size, this method may round the amount of requested memory up to the nearest full page.

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

See Also
+ dataWithCapacity: (page 947)
- initWithLength: (page 950)

Declared In
NSData.h

initWithLength:

Initializes and returns an NSMutableData object containing a given number of zeroed bytes.

```
- (id)initWithLength:(NSUInteger)length
```

Parameters
length
The number of bytes the object initially contains.

Return Value
An initialized NSMutableData object containing *length* zeroed bytes.

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

See Also
+ dataWithCapacity: (page 947)
+ dataWithLength: (page 947)
- initWithCapacity: (page 949)

Declared In
NSData.h

mutableBytes

Returns a pointer to the receiver's data.

```
- (void *)mutableBytes
```

Return Value
A pointer to the receiver's data.

Discussion
If the length of the receiver's data is not zero, this function is guaranteed to return a pointer to the object's internal bytes. If the length of receiver's data *is* zero, this function may or may not return `NULL` dependent upon many factors related to how the object was created (moreover, in this case the method result might change between different releases).

A sample using this method can be found in Working With Mutable Binary Data.

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

Declared In
`NSData.h`

replaceBytesInRange:withBytes:

Replaces with a given set of bytes a given range within the contents of the receiver.

```
- (void)replaceBytesInRange:(NSRange)range withBytes:(const void *)bytes
```

Parameters

range
The range within the receiver's contents to replace with `bytes`. The range must not exceed the bounds of the receiver.

bytes
The data to insert into the receiver's contents.

Discussion
If the location of *range* isn't within the receiver's range of bytes, an `NSRangeException` is raised. The receiver is resized to accommodate the new bytes, if necessary.

A sample using this method is given in Working With Mutable Binary Data.

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

See Also
- `replaceBytesInRange:withBytes:length:` (page 951)
- `resetBytesInRange:` (page 952)

Declared In
`NSData.h`

replaceBytesInRange:withBytes:length:

Replaces with a given set of bytes a given range within the contents of the receiver.

```
- (void)replaceBytesInRange:(NSRange)range withBytes:(const void *)replacementBytes
    length:(NSUInteger)replacementLength
```

Parameters

range

The range within the receiver's contents to replace with `bytes`. The range must not exceed the bounds of the receiver.

replacementBytes

The data to insert into the receiver's contents.

replacementLength

The number of bytes to take from *replacementBytes*.

Discussion

If the length of *range* is not equal to *replacementLength*, the receiver is resized to accommodate the new bytes. Any bytes past *range* in the receiver are shifted to accommodate the new bytes. You can therefore pass `NULL` for *replacementBytes* and `0` for *replacementLength* to delete bytes in the receiver in the range *range*. You can also replace a range (which might be zero-length) with more bytes than the length of the range, which has the effect of insertion (or "replace some and insert more").

Availability

Available in Mac OS X v10.2 and later.

See Also

`- replaceBytesInRange:withBytes:` (page 951)

Declared In

`NSData.h`

resetBytesInRange:

Replaces with zeroes the contents of the receiver in a given range.

```
- (void)resetBytesInRange:(NSRange)range
```

Parameters

range

The range within the contents of the receiver to be replaced by zeros. The range must not exceed the bounds of the receiver.

Discussion

If the location of *range* isn't within the receiver's range of bytes, an `NSRangeException` is raised. The receiver is resized to accommodate the new bytes, if necessary.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- replaceBytesInRange:withBytes:` (page 951)

Declared In

`NSData.h`

setData:

Replaces the entire contents of the receiver with the contents of another data object.

```
- (void)setData:(NSData *)aData
```

Parameters

aData

The data object whose content replaces that of the receiver.

Discussion

As part of its implementation, this method calls `replaceBytesInRange:withBytes:` (page 951).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSData.h`

setLength:

Extends or truncates a mutable data object to a given length.

```
- (void)setLength:(NSUInteger)length
```

Parameters

length

The new length for the receiver.

Discussion

If the mutable data object is extended, the additional bytes are filled with zeros.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- increaseLengthBy:` (page 949)

Declared In

`NSData.h`

NSMutableDictionary Class Reference

Inherits from	NSDictionary : NSObject
Conforms to	NSCoding (NSDictionary) NSCopying (NSDictionary) NSMutableCopying (NSDictionary) NSFastEnumeration (NSDictionary) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSDictionary.h
Companion guide	Collections Programming Topics for Cocoa
Related sample code	EnhancedAudioBurn GridCalendar Quartz Composer WWDC 2005 TextEdit StickiesExample TextEditPlus

Class at a Glance

An NSDictionary object stores a mutable set of entries.

Principal Attributes

- A count of the number of entries in the dictionary
- The set of keys contained in the dictionary
- The objects that correspond to the keys in the dictionary

`dictionaryWithCapacity:` (page 957)
Returns an empty dictionary with enough allocated space to hold a specified number of objects.

Commonly Used Methods

`removeObjectForKey:` (page 959)
Removes the specified entry from the dictionary.

`removeObjectsForKeys:` (page 960)
Removes multiple entries from the dictionary.

Overview

The `NSMutableDictionary` class declares the programmatic interface to objects that manage mutable associations of keys and values. With its two efficient primitive methods—`setObject:forKey:` (page 961) and `removeObjectForKey:` (page 959)—this class adds modification operations to the basic operations it inherits from `NSDictionary`.

The other methods declared here operate by invoking one or both of these primitives. The non-primitive methods provide convenient ways of adding or removing multiple entries at a time.

When an entry is removed from a mutable dictionary, the key and value objects that make up the entry receive `release` (page 2106) messages. If there are no further references to the objects, they're deallocated. Note that if your program keeps a reference to such an object, the reference will become invalid unless you remember to send the object a `retain` message before it's removed from the dictionary. For example, the third statement below would result in a runtime error if `anObject` was not retained before it was removed:

```
id anObject = [[aDictionary objectForKey:theKey] retain];

[aDictionary removeObjectForKey:theKey];
[anObject someMessage];
```

Tasks

Creating and Initializing a Mutable Dictionary

`+ dictionaryWithCapacity:` (page 957)
Creates and returns a mutable dictionary, initially giving it enough allocated memory to hold a given number of entries.

`- initWithCapacity:` (page 958)
Initializes a newly allocated mutable dictionary, allocating enough memory to hold *`numItems`* entries.

Adding Entries to a Mutable Dictionary

`- setObject:forKey:` (page 961)
Adds a given key-value pair to the receiver.

`- setValue:forKey:` (page 962)
Adds a given key-value pair to the receiver.

- `addEntriesFromDictionary:` (page 958)
  Adds to the receiver the entries from another dictionary.
- `setDictionary:` (page 961)
  Sets the contents of the receiver to entries in a given dictionary.

Removing Entries From a Mutable Dictionary

- `removeObjectForKey:` (page 959)
  Removes a given key and its associated value from the receiver.
- `removeAllObjects` (page 959)
  Empties the receiver of its entries.
- `removeObjectsForKeys:` (page 960)
  Removes from the receiver entries specified by elements in a given array.

Class Methods

dictionaryWithCapacity:

Creates and returns a mutable dictionary, initially giving it enough allocated memory to hold a given number of entries.

```
+ (id)dictionaryWithCapacity:(NSUInteger)numItems
```

Parameters

numItems
  The initial capacity of the new dictionary.

Return Value
A new mutable dictionary with enough allocated memory to hold *numItems* entries.

Discussion
Mutable dictionaries allocate additional memory as needed, so *numItems* simply establishes the object's initial capacity.

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

See Also
`dictionary` (page 498) (NSDictionary)
`dictionaryWithContentsOfFile:` (page 499) (NSDictionary)
`dictionaryWithContentsOfURL:` (page 500): (NSDictionary)
`dictionaryWithObject:forKey:` (page 500) (NSDictionary)
`dictionaryWithObjects:forKeys:` (page 501): (NSDictionary)
`dictionaryWithObjects:forKeys:count:` (page 502) (NSDictionary)
`dictionaryWithObjectsAndKeys:` (page 503) (NSDictionary)
`- initWithCapacity:` (page 958)

Related Sample Code
Dicey
EnhancedAudioBurn
QTKitPlayer
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSDictionary.h`

Instance Methods

addEntriesFromDictionary:

Adds to the receiver the entries from another dictionary.

```
- (void)addEntriesFromDictionary:(NSDictionary *)otherDictionary
```

Parameters
otherDictionary
The dictionary from which to add entries

Discussion
Each value object from *otherDictionary* is sent a `retain` (page 2108) message before being added to the receiver. In contrast, each key object is copied (using `copyWithZone:` (page 2042)—keys must conform to the `NSCopying` protocol), and the copy is added to the receiver.

If both dictionaries contain the same key, the receiver's previous value object for that key is sent a `release` message, and the new value object takes its place.

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

See Also
`- setObject:forKey:` (page 961)

Related Sample Code
EnhancedAudioBurn
EnhancedDataBurn
Sketch-112

Declared In
`NSDictionary.h`

initWithCapacity:

Initializes a newly allocated mutable dictionary, allocating enough memory to hold *numItems* entries.

```
- (id)initWithCapacity:(NSUInteger)numItems
```

Parameters

numItems

The initial capacity of the initialized dictionary.

Return Value

An initialized mutable dictionary, which might be different than the original receiver.

Discussion

Mutable dictionaries allocate additional memory as needed, so *numItems* simply establishes the object's initial capacity.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ dictionaryWithCapacity:` (page 957)

Declared In

`NSDictionary.h`

removeAllObjects

Empties the receiver of its entries.

```
- (void)removeAllObjects
```

Discussion

Each key and corresponding value object is sent a `release` (page 2106) message.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- removeObjectForKey:` (page 959)

`- removeObjectsForKeys:` (page 960)

Related Sample Code

QTSSConnectionMonitor

QTSSInspector

Declared In

`NSDictionary.h`

removeObjectForKey:

Removes a given key and its associated value from the receiver.

```
- (void)removeObjectForKey:(id)aKey
```

Parameters

aKey

The key to remove.

Discussion
Does nothing if *aKey* does not exist.

For example, assume you have an archived dictionary that records the call letters and associated frequencies of radio stations. To remove an entry for a defunct station, you could write code similar to the following:

```
NSMutableDictionary *stations = nil;

stations = [[NSMutableDictionary alloc]
        initWithContentsOfFile: pathToArchive];
[stations removeObjectForKey:@"KIKT"];
```

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

See Also
- `removeAllObjects` (page 959)
- `removeObjectsForKeys:` (page 960)

Related Sample Code
AnimatedSlider
CoreRecipes
EnhancedAudioBurn
GridCalendar

Declared In
`NSDictionary.h`

removeObjectsForKeys:

Removes from the receiver entries specified by elements in a given array.

```
- (void)removeObjectsForKeys:(NSArray *)keyArray
```

Parameters
keyArray
An array of objects specifying the keys to remove.

Discussion
If a key in *keyArray* does not exist, the entry is ignored.

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

See Also
- `removeObjectForKey:` (page 959)
- `removeObjectForKey:` (page 959)

Related Sample Code
CoreRecipes

Declared In
`NSDictionary.h`

setDictionary:

Sets the contents of the receiver to entries in a given dictionary.

```
- (void)setDictionary:(NSDictionary *)otherDictionary
```

Parameters

otherDictionary

A dictionary containing the new entries.

Discussion

All entries are removed from the receiver (with `removeAllObjects` (page 959)), then each entry from *otherDictionary* added into the receiver.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSDictionary.h`

setObject:forKey:

Adds a given key-value pair to the receiver.

```
- (void)setObject:(id)anObject forKey:(id)aKey
```

Parameters

anObject

The value for *key*. The object receives a `retain` message before being added to the receiver. This value must not be `nil`.

aKey

The key for *value*. The key is copied (using `copyWithZone:` (page 2042); keys must conform to the `NSCopying` protocol). The key must not be *nil*.

Discussion

Raises an `NSInvalidArgumentException` if *aKey* or *anObject* is `nil`. If you need to represent a nil value in the dictionary, use `NSNull`.

If *aKey* already exists in the receiver, the receiver's previous value object for that key is sent a `release` (page 2106) message and *anObject* takes its place.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- removeObjectForKey:` (page 959)

Related Sample Code

Dicey
GridCalendar
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus

Declared In
NSDictionary.h

setValue:forKey:

Adds a given key-value pair to the receiver.

```
- (void)setValue:(id)value forKey:(NSString *)key
```

Parameters

value

The value for *key*.

key

The key for *value*. Note that when using key-value coding, the key must be a string (see Key-Value Coding Fundamentals).

Discussion
This method adds *value* and *key* to the receiver using `setObject:forKey:` (page 961), unless *value* is `nil` in which case the method instead attempts to remove *key* using `removeObjectForKey:` (page 959).

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

See Also
`valueForKey:` (page 522) (NSDictionary)

Related Sample Code
CustomAtomicStoreSubclass
Dicey
SimpleCalendar
Spotlight
StickiesExample

Declared In
NSKeyValueCoding.h

NSMutableIndexSet Class Reference

Inherits from	NSIndexSet : NSObject
Conforms to	NSCoding (NSIndexSet) NSCopying (NSIndexSet) NSMutableCopying (NSIndexSet) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSIndexSet.h
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Collections Programming Topics for Cocoa
Related sample code	Core Data HTML Store MyPhoto

Overview

The `NSMutableIndexSet` class represents a mutable collection of unique unsigned integers, known as **indexes** because of the way they are used. This collection is referred to as a **mutable index set**.

The values in a mutable index set are always sorted, so the order in which values are added is irrelevant.

You must not subclass the `NSMutableIndexSet` class.

Tasks

Adding Indexes

- `addIndex:` (page 964)
  Adds an index to the receiver.
- `addIndexes:` (page 964)
  Adds the indexes in an index set to the receiver.
- `addIndexesInRange:` (page 965)
  Adds the indexes in an index range to the receiver.

Removing Indexes

- `removeIndex:` (page 966)
  Removes an index from the receiver.
- `removeIndexes:` (page 966)
  Removes the indexes in an index set from the receiver.
- `removeAllIndexes` (page 965)
  Removes the receiver's indexes.
- `removeIndexesInRange:` (page 966)
  Removes the indexes in an index range from the receiver.

Shifting Index Groups

- `shiftIndexesStartingAtIndex:by:` (page 967)
  Shifts a group of indexes to the left or the right within the receiver.

Instance Methods

addIndex:

Adds an index to the receiver.

```
- (void)addIndex:(NSUInteger)index
```

Parameters

index
  Index to add.

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

See Also
- `addIndexes:` (page 964)
- `addIndexesInRange:` (page 965)

Related Sample Code
Core Data HTML Store

Declared In
`NSIndexSet.h`

addIndexes:

Adds the indexes in an index set to the receiver.

```
- (void)addIndexes:(NSIndexSet *)indexSet
```

Parameters

indexSet

Index set to add.

Availability

Available in Mac OS X v10.3 and later.

See Also

- `addIndex:` (page 964)
- `addIndexesInRange:` (page 965)

Declared In

`NSIndexSet.h`

addIndexesInRange:

Adds the indexes in an index range to the receiver.

```
- (void)addIndexesInRange:(NSRange)indexRange
```

Parameters

indexRange

Index range to add. Must include only indexes representable as unsigned integers.

Discussion

This method raises an `NSRangeException` (page 2306) when *indexRange* would add an index that exceeds the maximum allowed value for unsigned integers.

Availability

Available in Mac OS X v10.3 and later.

See Also

- `addIndex:` (page 964)
- `addIndexes:` (page 964)

Declared In

`NSIndexSet.h`

removeAllIndexes

Removes the receiver's indexes.

```
- (void)removeAllIndexes
```

Availability

Available in Mac OS X v10.3 and later.

See Also

- `removeIndex:` (page 966)
- `removeIndexes:` (page 966)
- `removeIndexesInRange:` (page 966)

Declared In
NSIndexSet.h

removeIndex:

Removes an index from the receiver.

```
- (void)removeIndex:(NSUInteger)index
```

Parameters

index
Index to remove.

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

See Also
- removeAllIndexes (page 965)
- removeIndexes: (page 966)
- removeIndexesInRange: (page 966)

Declared In
NSIndexSet.h

removeIndexes:

Removes the indexes in an index set from the receiver.

```
- (void)removeIndexes:(NSIndexSet *)indexSet
```

Parameters

indexSet
Index set to remove.

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

See Also
- removeIndex: (page 966)
- removeAllIndexes (page 965)
- removeIndexesInRange: (page 966)

Declared In
NSIndexSet.h

removeIndexesInRange:

Removes the indexes in an index range from the receiver.

```
- (void)removeIndexesInRange:(NSRange)indexRange
```

Parameters

indexRange

Index range to remove.

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

See Also
- `removeIndex:` (page 966)
- `removeIndexes:` (page 966)
- `removeAllIndexes` (page 965)

Declared In
`NSIndexSet.h`

shiftIndexesStartingAtIndex:by:

Shifts a group of indexes to the left or the right within the receiver.

```
- (void)shiftIndexesStartingAtIndex:(NSUInteger)startIndex by:(NSInteger)delta
```

Parameters

startIndex

Head of the group of indexes to shift.

delta

Amount and direction of the shift. Positive integers shift the indexes to the right. Negative integers shift the indexes to the left.

Discussion
The group of indexes shifted is made up by *startIndex* and the indexes that follow it in the receiver.

A left shift deletes the indexes in the range `(startIndex-delta,delta)` from the receiver.

A right shift inserts empty space in the range `(indexStart,delta)` in the receiver.

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

Declared In
`NSIndexSet.h`

NSMutableSet Class Reference

Inherits from	NSSet : NSObject
Conforms to	NSCoding (NSSet) NSCopying (NSSet) NSMutableCopying (NSSet) NSFastEnumeration (NSSet) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSSet.h
Companion guide	Collections Programming Topics for Cocoa
Related sample code	Core Data HTML Store CoreRecipes CustomAtomicStoreSubclass MyPhoto QTMetadataEditor

Overview

The `NSMutableSet` class declares the programmatic interface to an object that manages a mutable set of objects. `NSMutableSet` provides support for the mathematical concept of a set. A set, both in its mathematical sense and in the `NSMutableSet` implementation, is an unordered collection of distinct elements.

The `NSCountedSet` class, which is a concrete subclass of `NSMutableSet`, supports mutable sets that can contain multiple instances of the same element. The `NSSet` class supports creating and managing immutable sets.

You add objects to an `NSMutableSet` object with `addObject:` (page 971), which adds a single object to the set; `addObjectsFromArray:` (page 972), which adds all objects from a specified array to the set; or `unionSet:` (page 975), which adds all the objects from another set. You remove objects from an `NSMutableSet` object using any of the methods `intersectSet:` (page 973), `minusSet:` (page 974), `removeAllObjects` (page 974), or `removeObject:` (page 975).

When an object is added to a set, it receives a `retain` (page 2108) message. When an object is removed from a mutable set, it receives a `release` (page 2106) message. If there are no further references to the object, this means that the object is deallocated. If your program keeps a reference to such an object, the reference will

become invalid unless you send the object a `retain` (page 2108) message before it's removed from the array. For example, if `anObject` is not retained before it is removed from the set, the third statement below could result in a runtime error:

```
id anObject = [[aSet anyObject] retain];
[aSet removeObject:anObject];
[anObject someMessage];
```

Tasks

Creating a Mutable Set

`+ setWithCapacity:` (page 971)
Creates and returns a mutable set with a given initial capacity.

`- initWithCapacity:` (page 973)
Returns an initialized mutable set with a given initial capacity.

Adding and Removing Entries

`- addObject:` (page 971)
Adds a given object to the receiver, if it is not already a member.

`- filterUsingPredicate:` (page 972)
Evaluates a given predicate against the receiver's content and removes from the receiver those objects for which the predicate returns false.

`- removeObject:` (page 975)
Removes a given object from the receiver.

`- removeAllObjects` (page 974)
Empties the receiver of all of its members.

`- addObjectsFromArray:` (page 972)
Adds to the receiver each object contained in a given array that is not already a member.

Combining and Recombining Sets

`- unionSet:` (page 975)
Adds to the receiver each object contained in another given set that is not already a member.

`- minusSet:` (page 974)
Removes from the receiver each object contained in another given set that is present in the receiver.

`- intersectSet:` (page 973)
Removes from the receiver each object that isn't a member of another given set.

`- setSet:` (page 975)
Empties the receiver, then adds to the receiver each object contained in another given set.

Class Methods

setWithCapacity:

Creates and returns a mutable set with a given initial capacity.

```
+ (id)setWithCapacity:(NSUInteger)numItems
```

Parameters

numItems

The initial capacity of the new set.

Return Value

A mutable set with initial capacity to hold *numItems* members.

Discussion

Mutable sets allocate additional memory as needed, so *numItems* simply establishes the object's initial capacity.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `initWithCapacity:` (page 973)
+ `set` (page 1445) (`NSSet`)
+ `setWithObjects:count:` (page 1447) (`NSSet`)

Declared In

`NSSet.h`

Instance Methods

addObject:

Adds a given object to the receiver, if it is not already a member.

```
- (void)addObject:(id)anObject
```

Parameters

anObject

The object to add to the receiver.

Discussion

If *anObject* is already present in the set, this method has no effect on either the set or *anObject*.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `addObjectsFromArray:` (page 972)

- unionSet: (page 975)

Related Sample Code
Core Data HTML Store
CoreRecipes
CustomAtomicStoreSubclass
QTMetadataEditor
Sketch-112

Declared In
NSSet.h

addObjectsFromArray:

Adds to the receiver each object contained in a given array that is not already a member.

```
- (void)addObjectsFromArray:(NSArray *)anArray
```

Parameters

anArray
An array of objects to add to the receiver.

Discussion
If a given element of the array is already present in the set, this method has no effect on either the set or the array element.

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

See Also
- addObject: (page 971)
- unionSet: (page 975)

Related Sample Code
Core Data HTML Store
CoreRecipes

Declared In
NSSet.h

filterUsingPredicate:

Evaluates a given predicate against the receiver's content and removes from the receiver those objects for which the predicate returns false.

```
- (void)filterUsingPredicate:(NSPredicate *)predicate
```

Parameters

predicate
A predicate.

Discussion
The following example illustrates the use of this method.

```
NSMutableSet *mutableSet =
    [NSMutableSet setWithObjects:@"One", @"Two", @"Three", @"Four", nil];
NSPredicate *predicate =
    [NSPredicate predicateWithFormat:@"SELF beginswith 'T'"];
[mutableSet filterUsingPredicate:predicate];
// mutableSet contains (Two, Three)
```

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

Declared In
NSPredicate.h

initWithCapacity:

Returns an initialized mutable set with a given initial capacity.

```
- (id)initWithCapacity:(NSUInteger)numItems
```

Parameters

numItems
The initial capacity of the set.

Return Value
An initialized mutable set with initial capacity to hold *numItems* members. The returned object might be different than the original receiver.

Discussion
Mutable sets allocate additional memory as needed, so *numItems* simply establishes the object's initial capacity.

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

See Also
+ setWithCapacity: (page 971)

Declared In
NSSet.h

intersectSet:

Removes from the receiver each object that isn't a member of another given set.

```
- (void)intersectSet:(NSSet *)otherSet
```

Parameters

otherSet
The set with which to perform the intersection.

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

See Also
- `removeObject:` (page 975)
- `removeAllObjects` (page 974)
- `minusSet:` (page 974)

Declared In
`NSSet.h`

minusSet:

Removes from the receiver each object contained in another given set that is present in the receiver.

```
- (void)minusSet:(NSSet *)otherSet
```

Parameters

otherSet
The set of objects to remove from the receiver.

Discussion
If any member of *otherSet* isn't present in the receiving set, this method has no effect on either the receiver or the *otherSet* member.

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

See Also
- `removeObject:` (page 975)
- `removeAllObjects` (page 974)
- `intersectSet:` (page 973)

Related Sample Code
CoreRecipes
MyPhoto

Declared In
`NSSet.h`

removeAllObjects

Empties the receiver of all of its members.

```
- (void)removeAllObjects
```

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

See Also
- `removeObject:` (page 975)
- `minusSet:` (page 974)

- intersectSet: (page 973)

Related Sample Code
CoreRecipes

Declared In
NSSet.h

removeObject:

Removes a given object from the receiver.

```
- (void)removeObject:(id)anObject
```

Parameters

anObject
The object to remove from the receiver.

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

See Also
- removeAllObjects (page 974)
- minusSet: (page 974)
- intersectSet: (page 973)

Related Sample Code
CoreRecipes

Declared In
NSSet.h

setSet:

Empties the receiver, then adds to the receiver each object contained in another given set.

```
- (void)setSet:(NSSet *)otherSet
```

Parameters

otherSet
The set whose members replace the receiver's content.

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

Declared In
NSSet.h

unionSet:

Adds to the receiver each object contained in another given set that is not already a member.

```
- (void)unionSet:(NSSet *)otherSet
```

Parameters

otherSet

The set of objects to add to the receiver.

Discussion

If any member of *otherSet* is already present in the receiver, this method has no effect on either the receiver or the *otherSet* member.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `addObject:` (page 971)
- `addObjectsFromArray:` (page 972)

Declared In

`NSSet.h`

NSMutableString Class Reference

Inherits from	NSString : NSObject
Conforms to	NSCoding (NSString) NSCopying (NSString) NSMutableCopying (NSString) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSString.h
Companion guide	String Programming Guide for Cocoa
Related sample code	CoreRecipes CubePuzzle NSGLImage NumberInput_IMKit_Sample QTAudioExtractionPanel

Overview

The `NSMutableString` class declares the programmatic interface to an object that manages a mutable string—that is, a string whose contents can be edited—that conceptually represents an array of Unicode characters. To construct and manage an immutable string—or a string that cannot be changed after it has been created—use an object of the `NSString` class.

The `NSMutableString` class adds one primitive method—`replaceCharactersInRange:withString:` (page 982)—to the basic string-handling behavior inherited from `NSString`. All other methods that modify a string work through this method. For example, `insertString:atIndex:` (page 981) simply replaces the characters in a range of `0` length, while `deleteCharactersInRange:` (page 980) replaces the characters in a given range with no characters.

Tasks

Creating and Initializing a Mutable String

`+ stringWithCapacity:` (page 978)
Returns an empty `NSMutableString` object with initial storage for a given number of characters.

`- initWithCapacity:` (page 981)
Returns an `NSMutableString` object initialized with initial storage for a given number of characters,

Modifying a String

`- appendFormat:` (page 979)
Adds a constructed string to the receiver.

`- appendString:` (page 980)
Adds to the end of the receiver the characters of a given string.

`- deleteCharactersInRange:` (page 980)
Removes from the receiver the characters in a given range.

`- insertString:atIndex:` (page 981)
Inserts into the receiver the characters of a given string at a given location.

`- replaceCharactersInRange:withString:` (page 982)
Replaces the characters from *`aRange`* with those in *`aString`*.

`- replaceOccurrencesOfString:withString:options:range:` (page 982)
Replaces all occurrences of a given string in a given range with another given string, returning the number of replacements.

`- setString:` (page 983)
Replaces the characters of the receiver with those in a given string.

Class Methods

stringWithCapacity:

Returns an empty `NSMutableString` object with initial storage for a given number of characters.

```
+ (id)stringWithCapacity:(NSUInteger)capacity
```

Parameters

`capacity`
The number of characters the string is expected to initially contain.

Return Value
An empty `NSMutableString` object with initial storage for *`capacity`* characters.

Discussion
The number of characters indicated by *capacity* is simply a hint to increase the efficiency of data storage. The value does *not* limit the length of the string.

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

Related Sample Code
PDFKitLinker2
QTRecorder

Declared In
NSString.h

Instance Methods

appendFormat:

Adds a constructed string to the receiver.

```
- (void)appendFormat:(NSString *)format ...
```

Parameters

format
A format string. See Formatting String Objects for more information. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *format* is `nil`.

...
A comma-separated list of arguments to substitute into *format*.

Discussion
The appended string is formed using `NSString`'s `stringWithFormat:` (page 1536) method with the arguments listed.

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

See Also
`- appendString:` (page 980)

Related Sample Code
CoreRecipes
iSpend
ThreadsImporter
ThreadsImportMovie
TimelineToTC

Declared In
NSString.h

appendString:

Adds to the end of the receiver the characters of a given string.

```
- (void)appendString:(NSString *)aString
```

Parameters

aString

The string to append to the receiver. *aString* must not be `nil`

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

See Also
`- appendFormat:` (page 979)

Related Sample Code
CoreRecipes
JavaSplashScreen
QTKitPlayer
SampleScannerApp
TimelineToTC

Declared In
`NSString.h`

deleteCharactersInRange:

Removes from the receiver the characters in a given range.

```
- (void)deleteCharactersInRange:(NSRange)aRange
```

Parameters

aRange

The range of characters to delete. *aRange* must not exceed the bounds of the receiver.

Important: Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the string.

Discussion
This method treats the length of the string as a valid range value that returns an empty string.

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

Related Sample Code
NumberInput_IMKit_Sample

Declared In
`NSString.h`

initWithCapacity:

Returns an `NSMutableString` object initialized with initial storage for a given number of characters,

```
- (id)initWithCapacity:(NSUInteger)capacity
```

Parameters

capacity

The number of characters the string is expected to initially contain.

Return Value

An initialized `NSMutableString` object with initial storage for *capacity* characters. The returned object might be different than the original receiver.

Discussion

The number of characters indicated by *capacity* is simply a hint to increase the efficiency of data storage. The value does *not* limit the length of the string.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSString.h`

insertString:atIndex:

Inserts into the receiver the characters of a given string at a given location.

```
- (void)insertString:(NSString *)aString atIndex:(NSUInteger)anIndex
```

Parameters

aString

The string to insert into the receiver. *aString* must not be `nil`.

anIndex

The location at which *aString* is inserted. The location must not exceed the bounds of the receiver.

Important: Raises an `NSRangeException` if *anIndex* lies beyond the end of the string.

Discussion

The new characters begin at *anIndex* and the existing characters from *anIndex* to the end are shifted by the length of *aString*.

This method treats the length of the string as a valid index value that returns an empty string.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Aperture Image Resizer

CustomSave

Declared In

`NSString.h`

replaceCharactersInRange:withString:

Replaces the characters from *aRange* with those in *aString*.

```
- (void)replaceCharactersInRange:(NSRange)aRange withString:(NSString *)aString
```

Parameters

aRange

The range of characters to replace. *aRange* must not exceed the bounds of the receiver.

Important: Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the receiver.

aString

The string with which to replace the characters in `aRange`. *aString* must not be `nil`.

Discussion
This method treats the length of the string as a valid range value that returns an empty string.

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

Related Sample Code
LSMSmartCategorizer

Declared In
`NSString.h`

replaceOccurrencesOfString:withString:options:range:

Replaces all occurrences of a given string in a given range with another given string, returning the number of replacements.

```
- (NSUInteger)replaceOccurrencesOfString:(NSString *)target withString:(NSString
    *)replacement options:(NSStringCompareOptions)opts range:(NSRange)searchRange
```

Parameters

target

The string to replace.

Important: Raises an `NSInvalidArgumentException` if *target* is `nil`.

replacement

The string with which to replace *target*.

Important: Raises an `NSInvalidArgumentException` if *replacement* is `nil`.

opts

A mask specifying search options. See *String Programming Guide for Cocoa* for details.

If *opts* is `NSBackwardsSearch`, the search is done from the end of the range. If *opts* is `NSAnchoredSearch`, only anchored (but potentially multiple) instances are replaced. `NSLiteralSearch` and `NSCaseInsensitiveSearch` also apply.

searchRange

The range of characters to replace. *aRange* must not exceed the bounds of the receiver. Specify *searchRange* as `NSMakeRange(0, [receiver length])` to process the entire string.

Important: Raises an `NSRangeException` if any part of *searchRange* lies beyond the end of the receiver.

Return Value
The number of replacements made.

Discussion
This method treats the length of the string as a valid range value that returns an empty string.

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

Related Sample Code
CubePuzzle
JavaSplashScreen
NewsReader
SpotlightAPI
VideoHardwareInfo

Declared In
`NSString.h`

setString:

Replaces the characters of the receiver with those in a given string.

```
- (void)setString:(NSString *)aString
```

Parameters

aString

The string with which to replace the receiver's content. *aString* must not be `nil`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

NumberInput_IMKit_Sample

PDFKitLinker2

QTCoreVideo201

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In

`NSString.h`

NSMutableURLRequest Class Reference

Inherits from	NSURLRequest : NSObject
Conforms to	NSCoding (NSURLRequest) NSCopying (NSURLRequest) NSMutableCopying (NSURLRequest) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSURLRequest.h
Availability	Available in Mac OS X v10.2 with Safari 1.0 installed. Available in Mac OS X v10.2.7 and later.
Companion guide	URL Loading System

Overview

NSMutableURLRequest is a subclass of NSURLRequest provided to aid developers who may find it more convenient to mutate a single request object for a series of URL load requests instead of creating an immutable NSURLRequest for each load.

This programming model is supported by the following contract between NSMutableURLRequest and NSURLConnection: NSURLConnection makes a deep copy of each NSMutableURLRequest object passed to one of its initializers.

NSMutableURLRequest, like NSURLRequest, is designed to be extended to support additional protocols by adding categories that access protocol specific values from a property object using NSURLProtocol's `propertyForKey:inRequest:` (page 1818) and `setProperty:forKey:inRequest:` (page 1820) methods.

Tasks

Setting Request Properties

- `setCachePolicy:` (page 987)
  Sets the cache policy of the receiver.
- `setMainDocumentURL:` (page 989)
  Sets the main document URL for the receiver.

- `setTimeoutInterval:` (page 989)
  Sets the receiver's timeout interval, in seconds.
- `setURL:` (page 990)
  Sets the URL of the receiver

Setting HTTP Specific Properties

- `addValue:forHTTPHeaderField:` (page 986)
  Adds an HTTP header to the receiver's HTTP header dictionary.
- `setAllHTTPHeaderFields:` (page 987)
  Replaces the receiver's header fields with the passed values.
- `setHTTPBody:` (page 987)
  Sets the request body of the receiver to the specified data.
- `setHTTPBodyStream:` (page 988)
  Sets the request body of the receiver to the contents of a specified input stream.
- `setHTTPMethod:` (page 988)
  Sets the receiver's HTTP request method.
- `setHTTPShouldHandleCookies:` (page 989)
  Sets whether the receiver should use the default cookie handling for the request.
- `setValue:forHTTPHeaderField:` (page 990)
  Sets the specified HTTP header field.

Instance Methods

addValue:forHTTPHeaderField:

Adds an HTTP header to the receiver's HTTP header dictionary.

```
- (void)addValue:(NSString *)value forHTTPHeaderField:(NSString *)field
```

Parameters

value
  The value for the header field.

field
  The name of the header field. In keeping with the HTTP RFC, HTTP header field names are case-insensitive

Discussion
This method provides the ability to add values to header fields incrementally. If a value was previously set for the specified *field*, the supplied *value* is appended to the existing value using the appropriate field delimiter. In the case of HTTP, this is a comma.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- `setValue:forHTTPHeaderField:` (page 990)

Declared In
`NSURLRequest.h`

setAllHTTPHeaderFields:

Replaces the receiver's header fields with the passed values.

```
- (void)setAllHTTPHeaderFields:(NSDictionary *)headerFields
```

Parameters

headerFields

A dictionary with the new header fields. HTTP header fields must be string values; therefore, each object and key in the *headerFields* dictionary must be a subclass of NSString. If either the key or value for a key-value pair is not a subclass of NSString, the key-value pair is skipped.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- `setValue:forHTTPHeaderField:` (page 990)

Declared In
`NSURLRequest.h`

setCachePolicy:

Sets the cache policy of the receiver.

```
- (void)setCachePolicy:(NSURLRequestCachePolicy)policy
```

Parameters

policy

The new cache policy.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- `cachePolicy` (page 1828)

Declared In
`NSURLRequest.h`

setHTTPBody:

Sets the request body of the receiver to the specified data.

```
- (void)setHTTPBody:(NSData *)data
```

Parameters

data

The new request body for the receiver. This is sent as the message body of the request, as in an HTTP `POST` request.

Discussion

Setting the HTTP body data clears any input stream set by `setHTTPBodyStream:` (page 988). These values are mutually exclusive.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

`NSURLRequest.h`

setHTTPBodyStream:

Sets the request body of the receiver to the contents of a specified input stream.

```
- (void)setHTTPBodyStream:(NSInputStream *)inputStream
```

Parameters

inputStream

The input stream that will be the request body of the receiver. The entire contents of the stream will be sent as the body, as in an HTTP `POST` request. The *inputStream* should be unopened and the receiver will take over as the stream's delegate.

Discussion

Setting a body stream clears any data set by `setHTTPBody:` (page 987). These values are mutually exclusive.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`NSURLRequest.h`

setHTTPMethod:

Sets the receiver's HTTP request method.

```
- (void)setHTTPMethod:(NSString *)method
```

Parameters

method

The new HTTP request method. The default HTTP method is "GET".

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In
NSURLRequest.h

setHTTPShouldHandleCookies:

Sets whether the receiver should use the default cookie handling for the request.

```
- (void)setHTTPShouldHandleCookies:(BOOL)handleCookies
```

Parameters

handleCookies

YES if the receiver should use the default cookie handling for the request, NO otherwise. The default is YES.

Special Considerations

In Mac OS X v10.2 with Safari 1.0 the value set by this method is not respected by the framework.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSURLRequest.h

setMainDocumentURL:

Sets the main document URL for the receiver.

```
- (void)setMainDocumentURL:(NSURL *)theURL
```

Parameters

theURL

The new main document URL. Can be nil.

Discussion
The caller should set the main document URL to an appropriate main document, if known. For example, when loading a web page the URL of the HTML document for the top-level frame would be appropriate. This URL will be used for the "only from same domain as main document" cookie accept policy.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSURLRequest.h

setTimeoutInterval:

Sets the receiver's timeout interval, in seconds.

```
- (void)setTimeoutInterval:(NSTimeInterval)timeoutInterval
```

Parameters

timeoutInterval

The timeout interval, in seconds. If during a connection attempt the request remains idle for longer than the timeout interval, the request is considered to have timed out. The default timeout interval is 60 seconds.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

See Also

- `timeoutInterval` (page 1831)

Declared In

`NSURLRequest.h`

setURL:

Sets the URL of the receiver

```
- (void)setURL:(NSURL *)theURL
```

Parameters

theURL

The new URL.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

See Also

- `URL` (page 1832)

Declared In

`NSURLRequest.h`

setValue:forHTTPHeaderField:

Sets the specified HTTP header field.

```
- (void)setValue:(NSString *)value forHTTPHeaderField:(NSString *)field
```

Parameters

value

The new value for the header field. Any existing value for the field is replaced by the new value.

field

The name of the header field to set. In keeping with the HTTP RFC, HTTP header field names are case-insensitive.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

See Also
- `addValue:forHTTPHeaderField:` (page 986)

Declared In
`NSURLRequest.h`

NSNameSpecifier Class Reference

Inherits from	NSScriptObjectSpecifier : NSObject
Conforms to	NSCoding (NSScriptObjectSpecifier) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSScriptObjectSpecifiers.h
Availability	Available in Mac OS X v10.2 and later.
Companion guide	Cocoa Scripting Guide

Overview

Specifies an object in a collection (or container) by name. For example, the following script specifies both an application and a window by name. In this script, the named window's implicitly specified container is the Finder application's list of open windows.

```
tell application "Finder" -- specifies an application  by name
    close window "Reports" -- specifies a window by name
end tell
```

This specifier works only for objects that have a name property. You don't normally subclass `NSNameSpecifier`.

The evaluation of an instance of `NSNameSpecifier` follows these steps until the specified object is found:

1. If the container implements a method whose selector matches the relevant `valueIn<Key>WithName:` pattern established by scripting key-value coding, the method is invoked. This method can potentially be very fast, and it may be relatively easy to implement.

2. As is the case when evaluating any script object specifier, the container of the specified object is given a chance to evaluate the object specifier. If the container class implements the `indicesOfObjectsByEvaluatingObjectSpecifier` method, the method is invoked. This method can potentially be very fast, but it is relatively difficult to implement.

3. An instance of `NSWhoseSpecifier` that specifies the first object whose relevant `'pnam'` attribute matches the name is synthesized and evaluated. The instance of `NSWhoseSpecifier` must search through all of the keyed elements in the container, looking for a match. The search is potentially very slow.

Tasks

Initializing a Name Specifier

- `initWithContainerClassDescription:containerSpecifier:key:name:` (page 994)
  Invokes the super class's `initWithContainerClassDescription:containerSpecifier:key:` (page 1418) method and then sets the name instance variable to *name*.

Accessing a Name Specifier

- `name` (page 994)
  Returns the name encapsulated by the receiver for the specified object in the container.
- `setName:` (page 995)
  Sets the name encapsulated with the receiver for the specified object in the container.

Instance Methods

initWithContainerClassDescription:containerSpecifier:key:name:

Invokes the super class's `initWithContainerClassDescription:containerSpecifier:key:` (page 1418) method and then sets the name instance variable to *name*.

```
- (id)initWithContainerClassDescription:(NSScriptClassDescription *)classDesc
    containerSpecifier:(NSScriptObjectSpecifier *)container key:(NSString *)property
     name:(NSString *)name
```

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

Declared In
`NSScriptObjectSpecifiers.h`

name

Returns the name encapsulated by the receiver for the specified object in the container.

```
- (NSString *)name
```

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

See Also
- `setName:` (page 995)

Declared In
NSScriptObjectSpecifiers.h

setName:

Sets the name encapsulated with the receiver for the specified object in the container.

```
- (void)setName:(NSString *)name
```

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

See Also
- name (page 994)

Declared In
NSScriptObjectSpecifiers.h

NSNetService Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.2 and later.
Declared in	Foundation/NSNetServices.h
Companion guides	Bonjour Overview NSNetServices and CFNetServices Programming Guide
Related sample code	CocoaEcho CocoaSOAP GridCalendar PictureSharing PictureSharingBrowser

Overview

The `NSNetService` class represents a network service that your application publishes or uses as a client. This class and the `NSNetServiceBrowser` class use multicast DNS to convey information about network services to and from your application. The API of `NSNetService` provides a convenient way to publish the services offered by your application and to resolve the socket address for a service.

The types of services you access using `NSNetService` are the same types that you access directly using BSD sockets. HTTP and FTP are two services commonly provided by systems. (For a list of common services and the ports used by those services, see the file `/etc/services`.) Applications can also define their own custom services to provide specific data to clients.

You can use the `NSNetService` class as either a publisher of a service or as a client of a service. If your application publishes a service, your code must acquire a port and prepare a socket to communicate with clients. Once your socket is ready, you use the `NSNetService` class to notify clients that your service is ready. If your application is the client of a network service, you can either create an `NSNetService` object directly (if you know the exact host and port information) or you can use an `NSNetServiceBrowser` object to browse for services.

To publish a service, you must initialize your `NSNetService` object with the service name, domain, type, and port information. All of this information must be valid for the socket created by your application. Once initialized, you call the `publish` (page 1006) method to broadcast your service information out to the network.

When connecting to a service, you would normally use the `NSNetServiceBrowser` class to locate the service on the network and obtain the corresponding `NSNetService` object. Once you have the object, you proceed to call the `resolveWithTimeout:` (page 1008) method to verify that the service is available and ready for your application. If it is, the `addresses` (page 1001) method returns the socket information you can use to connect to the service.

The methods of `NSNetService` operate asynchronously so that your application is not impacted by the speed of the network. All information about a service is returned to your application through the `NSNetService` object's delegate. You must provide a delegate object to respond to messages and to handle errors appropriately.

Tasks

Creating Network Services

- `initWithDomain:type:name:` (page 1003)
  Returns the receiver, initialized as a network service of a given type and sets the initial host information.
- `initWithDomain:type:name:port:` (page 1004)
  Initializes the receiver as a network service of type *type* at the socket location specified by *domain*, *name*, and *port*.

Configuring Network Services

+ `dataFromTXTRecordDictionary:` (page 1000)
  Returns an `NSData` object representing a TXT record formed from a given dictionary.
+ `dictionaryFromTXTRecordData:` (page 1000)
  Returns a dictionary representing a TXT record given as an `NSData` object.
- `addresses` (page 1001)
  Returns an array containing `NSData` objects, each of which contains a socket address for the service.
- `domain` (page 1002)
  Returns the domain name of the service.
- `getInputStream:outputStream:` (page 1002)
  Retrieves by reference the input and output streams for the receiver and returns a Boolean value that indicates whether they were retrieved successfully.
- `hostName` (page 1003)
  Returns the host name of the computer providing the service.
- `name` (page 1005)
  Returns the name of the service.
- `type` (page 1012)
  Returns the type of the service.
- `TXTRecordData` (page 1011)
  Returns the TXT record for the receiver.

- `setTXTRecordData:` (page 1010)
  Sets the TXT record for the receiver, and returns a Boolean value that indicates whether the operation was successful.
- `delegate` (page 1002)
  Returns the delegate for the receiver.
- `setDelegate:` (page 1009)
  Sets the delegate for the receiver.
- `protocolSpecificInformation` (page 1006) Deprecated in Mac OS X v10.4
  Returns protocol specific information for legacy ZeroConf-style clients. (Deprecated. Use `TXTRecordData` (page 1011) instead.)
- `setProtocolSpecificInformation:` (page 1009) Deprecated in Mac OS X v10.4
  Sets protocol specific information for legacy ZeroConf-style clients. (Deprecated. Use `setTXTRecordData:` (page 1010) instead.)

Managing Run Loops

- `scheduleInRunLoop:forMode:` (page 1008)
  Adds the service to the specified run loop.
- `removeFromRunLoop:forMode:` (page 1007)
  Removes the service from the given run loop for a given mode.

Using Network Services

- `publish` (page 1006)
  Attempts to advertise the receiver's on the network.
- `publishWithOptions:` (page 1006)
  Attempts to advertise the receiver on the network, with the given options.
- `netServiceWillPublish:` (page 1014) *delegate method*
  Notifies the delegate that the network is ready to publish the service.
- `netService:didNotPublish:` (page 1012) *delegate method*
  Notifies the delegate that a service could not be published.
- `netServiceDidPublish:` (page 1013) *delegate method*
  Notifies the delegate that a service was successfully published.
- `resolve` (page 1007)
  Starts a resolve process for the receiver. (Deprecated. Use `resolveWithTimeout:` (page 1008) instead.)
- `resolveWithTimeout:` (page 1008)
  Starts a resolve process of a finite duration for the receiver.
- `netServiceWillResolve:` (page 1015) *delegate method*
  Notifies the delegate that the network is ready to resolve the service.
- `netService:didNotResolve:` (page 1012) *delegate method*
  Informs the delegate that an error occurred during resolution of a given service.
- `netServiceDidResolveAddress:` (page 1014) *delegate method*
  Informs the delegate that the address for a given service was resolved.

- `port` (page 1005)
  Provides the port of the receiver.
- `startMonitoring` (page 1010)
  Starts the monitoring of TXT-record updates for the receiver.
- `netService:didUpdateTXTRecordData:` (page 1013) *delegate method*
  Notifies the delegate that the TXT record for a given service has been updated.
- `stop` (page 1011)
  Halts a currently running attempt to publish or resolve a service.
- `stopMonitoring` (page 1011)
  Stops the monitoring of TXT-record updates for the receiver.
- `netServiceDidStop:` (page 1014) *delegate method*
  Informs the delegate that a `publish` (page 1006) or `resolveWithTimeout:` (page 1008) request was stopped.

Class Methods

dataFromTXTRecordDictionary:

Returns an `NSData` object representing a TXT record formed from a given dictionary.

```
+ (NSData *)dataFromTXTRecordDictionary:(NSDictionary *)txtDictionary
```

Parameters

txtDictionary
  A dictionary containing a TXT record.

Return Value
An `NSData` object representing TXT data formed from *txtDictionary*. Fails an assertion if *txtDictionary* cannot be represented as an `NSData` object.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `TXTRecordData` (page 1011)
+ `dictionaryFromTXTRecordData:` (page 1000)

Declared In
`NSNetServices.h`

dictionaryFromTXTRecordData:

Returns a dictionary representing a TXT record given as an `NSData` object.

```
+ (NSDictionary *)dictionaryFromTXTRecordData:(NSData *)txtData
```

Parameters

txtData

A data object encoding a TXT record.

Return Value

A dictionary representing *txtData*. The dictionary's keys are `NSString` objects using UTF8 encoding. The values associated with all the dictionary's keys are `NSData` objects that encapsulate strings or data.

Fails an assertion if *txtData* cannot be represented as an `NSDictionary` object.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- TXTRecordData` (page 1011)

`+ dataFromTXTRecordDictionary:` (page 1000)

Declared In

`NSNetServices.h`

Instance Methods

addresses

Returns an array containing `NSData` objects, each of which contains a socket address for the service.

```
- (NSArray *)addresses
```

Return Value

An array containing `NSData` objects, each of which contains a socket address for the service. Each `NSData` object in the returned array contains an appropriate `sockaddr` structure that you can use to connect to the socket. The exact type of this structure depends on the service to which you are connecting. If no addresses were resolved for the service, the returned array contains zero elements.

Discussion

It is possible for a single service to resolve to more than one address or not resolve to any addresses. A service might resolve to multiple addresses if the computer publishing the service is currently multihoming.

Availability

Available in Mac OS X v10.2 and later.

See Also

`- resolve` (page ?)

Related Sample Code

CocoaSOAP

PictureSharingBrowser

Declared In

`NSNetServices.h`

delegate

Returns the delegate for the receiver.

```
- (id)delegate
```

Return Value
The delegate for the receiver.

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

See Also
- setDelegate: (page 1009)

Declared In
NSNetServices.h

domain

Returns the domain name of the service.

```
- (NSString *)domain
```

Return Value
The domain name of the service.

This can be an explicit domain name or it can contain the generic local domain name, @"local." (note the trailing period, which indicates an absolute name).

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

Related Sample Code
GridCalendar

Declared In
NSNetServices.h

getInputStream:outputStream:

Retrieves by reference the input and output streams for the receiver and returns a Boolean value that indicates whether they were retrieved successfully.

```
- (BOOL)getInputStream:(NSInputStream **)inputStream outputStream:(NSOutputStream
    **)outputStream
```

Parameters

inputStream
Upon return, the input stream for the receiver.

outputStream
Upon return, the output stream for the receiver.

Return Value
`YES` if the streams are created successfully, otherwise `NO`.

Discussion
After this method is called, no delegate callbacks are called by the receiver.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSNetServices.h`

hostName

Returns the host name of the computer providing the service.

```
- (NSString *)hostName
```

Return Value
The host name of the computer providing the service. Returns `nil` if a successful resolve has not occurred.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSNetServices.h`

initWithDomain:type:name:

Returns the receiver, initialized as a network service of a given type and sets the initial host information.

```
- (id)initWithDomain:(NSString *)domain type:(NSString *)type name:(NSString *)name
```

Parameters

domain
The domain for the service. For the local domain, use `@"local."` not `@""`.

type
The network service type.

type must contain both the service type and transport layer information. To ensure that the mDNS responder searches for services, as opposed to hosts, prefix both the service name and transport layer name with an underscore character ("_"). For example, to search for an HTTP service on TCP, you would use the type string `"_http._tcp."`. Note that the period character at the end of the string, which indicates that the domain name is an absolute name, is required.

name
The name of the service to resolve.

Return Value
The receiver, initialized as a network service named *name* of type *type* in the domain *domain*.

Discussion
This method is the appropriate initializer to use to resolve a service—to publish a service, use initWithDomain:type:name:port: (page 1004).

If you know the values for *domain*, *type*, and *name* of the service you wish to connect to, you can create an `NSNetService` object using this initializer and call `resolveWithTimeout:` (page 1008) on the result.

You cannot use this initializer to publish a service. This initializer passes an invalid port number to the designated initializer, which prevents the service from being registered. Calling `publish` (page 1006) on an `NSNetService` object initialized with this method generates a call to your delegate's `netService:didNotPublish:` (page 1012) method with an `NSNetServicesBadArgumentError` error.

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

See Also
`- initWithDomain:type:name:port:` (page 1004)

Related Sample Code
CocoaSOAP
GridCalendar

Declared In
`NSNetServices.h`

initWithDomain:type:name:port:

Initializes the receiver as a network service of type *type* at the socket location specified by *domain*, *name*, and *port*.

```
- (id)initWithDomain:(NSString *)domain type:(NSString *)type name:(NSString *)name
    port:(int)port
```

Parameters

domain

The domain for the service. For the local domain, use `@"local."` not `@""`.

It is generally preferred to use a `NSNetServiceBrowser` object to obtain the local registration domain in which to publish your service. To use this default domain, simply pass in an empty string (`@""`).

type

The network service type.

type must contain both the service type and transport layer information. To ensure that the mDNS responder searches for services, as opposed to hosts, prefix both the service name and transport layer name with an underscore character ("_"). For example, to search for an HTTP service on TCP, you would use the type string `"_http._tcp."`. Note that the period character at the end of the string, which indicates that the domain name is an absolute name, is required.

name

The name by which the service is identified to the network. The name must be unique.

port

The port on which the service is published.

port must be a port number acquired by your application for the service.

Discussion
You use this method to create a service that you wish to publish on the network. Although you can also use this method to create a service you wish to resolve on the network, it is generally more appropriate to use the `initWithDomain:type:name:` (page 1003) method instead.

When publishing a service, you must provide valid arguments in order to advertise your service correctly. If the host computer has access to multiple registration domains, you must create separate `NSNetService` objects for each domain. If you attempt to publish in a domain for which you do not have registration authority, your request may be denied.

It is acceptable to use an empty string for the *domain* argument when publishing or browsing a service, but do not rely on this for resolution.

This method is the designated initializer.

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

See Also
`- initWithDomain:type:name:` (page 1003)

Related Sample Code
CocoaEcho
CocoaHTTPServer
CocoaSOAP
PictureSharing

Declared In
`NSNetServices.h`

name

Returns the name of the service.

```
- (NSString *)name
```

Return Value
The name of the service.

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

Related Sample Code
GridCalendar

Declared In
`NSNetServices.h`

port

Provides the port of the receiver.

```
- (NSInteger)port
```

Return Value
The receiver's port. `-1` when it has not been resolved.

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

Declared In
NSNetServices.h

protocolSpecificInformation

Returns protocol specific information for legacy ZeroConf-style clients. (Deprecated in Mac OS X v10.4. Use `TXTRecordData` (page 1011) instead.)

```
- (NSString *)protocolSpecificInformation
```

Return Value
Any protocol-specific data associated with the service.

Discussion
This method is provided for legacy support of older ZeroConf-style clients and its use is discouraged.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.

See Also
`- setProtocolSpecificInformation:` (page 1009)

Declared In
NSNetServices.h

publish

Attempts to advertise the receiver's on the network.

```
- (void)publish
```

Discussion
This method returns immediately, with success or failure indicated by the callbacks to the delegate.

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

See Also
`- stop` (page 1011)

Declared In
NSNetServices.h

publishWithOptions:

Attempts to advertise the receiver on the network, with the given options.

```
- (void)publishWithOptions:(NSNetServiceOptions)serviceOptions
```

Parameters

serviceOptions

Options for the receiver.

Discussion

This method returns immediately, with success or failure indicated by the callbacks to the delegate.

Availability

Available in Mac OS X v10.5 and later.

Declared In

NSNetServices.h

removeFromRunLoop:forMode:

Removes the service from the given run loop for a given mode.

```
- (void)removeFromRunLoop:(NSRunLoop *)aRunLoop forMode:(NSString *)mode
```

Parameters

aRunLoop

The run loop from which to remove the receiver.

mode

The run loop mode from which to remove the receiver. Possible values for *mode* are discussed in the "Constants" section of NSRunLoop.

Discussion

You can use this method in conjunction with scheduleInRunLoop:forMode: (page 1008) to transfer the service to a different run loop. Although it is possible to remove an NSNetService object completely from any run loop and then attempt actions on it, it is an error to do so.

Availability

Available in Mac OS X v10.2 and later.

See Also

- scheduleInRunLoop:forMode: (page 1008)

Declared In

NSNetServices.h

resolve

Starts a resolve process for the receiver. (**Deprecated.** Use resolveWithTimeout: (page 1008) instead.)

```
- (void)resolve
```

Discussion

Attempts to determine at least one address for the receiver. This method returns immediately, with success or failure indicated by the callbacks to the delegate.

In Mac OS X v10.4, this method calls resolveWithTimeout: (page 1008) with a timeout value of 5.

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

See Also
- `addresses` (page 1001)
- `stop` (page 1011)
- `resolveWithTimeout:` (page 1008)

Declared In
`NSNetServices.h`

resolveWithTimeout:

Starts a resolve process of a finite duration for the receiver.

```
- (void)resolveWithTimeout:(NSTimeInterval)timeout
```

Parameters

timeout

The maximum number of seconds to attempt a resolve.

Discussion
If the resolve succeeds before the *timeout* period lapses, the receiver sends `netServiceDidResolveAddress:` (page 1014) to the delegate. Otherwise, the receiver sends `netService:didNotResolve:` (page 1012) to the delegate.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `addresses` (page 1001)
- `stop` (page 1011)

Related Sample Code
CocoaSOAP

Declared In
`NSNetServices.h`

scheduleInRunLoop:forMode:

Adds the service to the specified run loop.

```
- (void)scheduleInRunLoop:(NSRunLoop *)aRunLoop forMode:(NSString *)mode
```

Parameters

aRunLoop

The run loop to which to add the receiver.

mode

The run loop mode to which to add the receiver. Possible values for *mode* are discussed in the "Constants" section of `NSRunLoop`.

Discussion
You can use this method in conjunction with `removeFromRunLoop:forMode:` (page 1007) to transfer a service to a different run loop. You should not attempt to run a service on multiple run loops.

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

See Also
`- removeFromRunLoop:forMode:` (page 1007)

Related Sample Code
CocoaSOAP

Declared In
`NSNetServices.h`

setDelegate:

Sets the delegate for the receiver.

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

Parameters
delegate
The delegate for the receiver.

Discussion
The delegate is not retained.

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

See Also
`- delegate` (page 1002)

Declared In
`NSNetServices.h`

setProtocolSpecificInformation:

Sets protocol specific information for legacy ZeroConf-style clients. (Deprecated in Mac OS X v10.4. Use `setTXTRecordData:` (page 1010) instead.)

```
- (void)setProtocolSpecificInformation:(NSString *)specificInformation
```

Parameters
specificInformation
Information for the protocol.

Discussion
Attaches protocol-specific data to the service.

This method is provided for legacy support of older ZeroConf-style clients and its use is discouraged.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.

See Also
- `protocolSpecificInformation` (page 1006)

Declared In
`NSNetServices.h`

setTXTRecordData:

Sets the TXT record for the receiver, and returns a Boolean value that indicates whether the operation was successful.

```
- (BOOL)setTXTRecordData:(NSData *)recordData
```

Parameters
recordData
The TXT record for the receiver.

Return Value
`YES` if *recordData* is successfully set as the TXT record, otherwise `NO`.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `TXTRecordData` (page 1011)

Declared In
`NSNetServices.h`

startMonitoring

Starts the monitoring of TXT-record updates for the receiver.

```
- (void)startMonitoring
```

Discussion
The delegate must implement `netService:didUpdateTXTRecordData:` (page 1013), which is called when the TXT record for the receiver is updated.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `stopMonitoring` (page 1011)

Declared In
`NSNetServices.h`

stop

Halts a currently running attempt to publish or resolve a service.

```
- (void)stop
```

Discussion
This method results in the sending of a `netServiceDidStop:` (page 1014) message to the delegate.

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

Related Sample Code
CocoaSOAP
PictureSharingBrowser

Declared In
`NSNetServices.h`

stopMonitoring

Stops the monitoring of TXT-record updates for the receiver.

```
- (void)stopMonitoring
```

Availability
Available in Mac OS X v10.4 and later.

See Also
`- startMonitoring` (page 1010)

Declared In
`NSNetServices.h`

TXTRecordData

Returns the TXT record for the receiver.

```
- (NSData *)TXTRecordData
```

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setTXTRecordData:` (page 1010)
`+ dictionaryFromTXTRecordData:` (page 1000)
`+ dataFromTXTRecordDictionary:` (page 1000)

Declared In
`NSNetServices.h`

type

Returns the type of the service.

```
- (NSString *)type
```

Return Value
The type of the service.

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

Related Sample Code
GridCalendar

Declared In
`NSNetServices.h`

Delegate Methods

netService:didNotPublish:

Notifies the delegate that a service could not be published.

```
- (void)netService:(NSNetService *)sender didNotPublish:(NSDictionary *)errorDict
```

Parameters

sender
The service that could not be published.

errorDict
A dictionary containing information about the problem. The dictionary contains the keys `NSNetServicesErrorCode` and `NSNetServicesErrorDomain`.

Discussion
This method may be called long after a `netServiceWillPublish:` (page 1014) message has been delivered to the delegate.

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

Declared In
`NSNetServices.h`

netService:didNotResolve:

Informs the delegate that an error occurred during resolution of a given service.

```
- (void)netService:(NSNetService *)sender didNotResolve:(NSDictionary *)errorDict
```

Parameters

sender

The service that did not resolve.

errorDict

A dictionary containing information about the problem. The dictionary contains the keys `NSNetServicesErrorCode` and `NSNetServicesErrorDomain`.

Discussion

Clients may try to resolve again upon receiving this error. For example, a DNS rotary may yield different IP addresses on different resolution requests.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`NSNetServices.h`

netService:didUpdateTXTRecordData:

Notifies the delegate that the TXT record for a given service has been updated.

```
- (void)netService:(NSNetService *)sender didUpdateTXTRecordData:(NSData *)data
```

Parameters

sender

The service whose TXT record was updated.

data

The new TXT record.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- startMonitoring` (page 1010)

Declared In

`NSNetServices.h`

netServiceDidPublish:

Notifies the delegate that a service was successfully published.

```
- (void)netServiceDidPublish:(NSNetService *)sender
```

Parameters

sender

The service that was published.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`NSNetServices.h`

netServiceDidResolveAddress:

Informs the delegate that the address for a given service was resolved.

```
- (void)netServiceDidResolveAddress:(NSNetService *)sender
```

Parameters

sender

The service that was resolved.

Discussion

The delegate can use the `addresses` (page 1001) method to retrieve the service's address.

Availability

Available in Mac OS X v10.2 and later.

See Also

`- addresses` (page 1001)

Declared In

`NSNetServices.h`

netServiceDidStop:

Informs the delegate that a `publish` (page 1006) or `resolveWithTimeout:` (page 1008) request was stopped.

```
- (void)netServiceDidStop:(NSNetService *)sender
```

Parameters

sender

The service that stopped.

Availability

Available in Mac OS X v10.2 and later.

See Also

`- stop` (page 1011)

Declared In

`NSNetServices.h`

netServiceWillPublish:

Notifies the delegate that the network is ready to publish the service.

```
- (void)netServiceWillPublish:(NSNetService *)sender
```

Parameters

sender

The service that is ready to publish.

Discussion

Publication of the service proceeds asynchronously and may still generate a call to the delegate's `netService:didNotPublish:` (page 1012) method if an error occurs.

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

Declared In
`NSNetServices.h`

netServiceWillResolve:

Notifies the delegate that the network is ready to resolve the service.

```
- (void)netServiceWillResolve:(NSNetService *)sender
```

Parameters

sender

The service that the network is ready to resolve.

Discussion
Resolution of the service proceeds asynchronously and may still generate a call to the delegate's `netService:didNotResolve:` (page 1012) method if an error occurs.

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

Declared In
`NSNetServices.h`

Constants

NSNetServices Errors

If an error occurs, the delegate error-handling methods return a dictionary with the following keys.

```
extern NSString *NSNetServicesErrorCode;
extern NSString *NSNetServicesErrorDomain;
```

Constants

`NSNetServicesErrorCode`

This key identifies the error that occurred during the most recent operation.

Available in Mac OS X v10.2 and later.

Declared in `NSNetServices.h`.

`NSNetServicesErrorDomain`

This key identifies the originator of the error, which is either the `NSNetService` object or the mach network layer. For most errors, you should not need the value provided by this key.

Available in Mac OS X v10.2 and later.

Declared in `NSNetServices.h`.

Declared In
`NSNetServices.h`

NSNetServicesError

These constants identify errors that can occur when accessing net services.

```
typedef enum {
    NSNetServicesUnknownError = -72000,
    NSNetServicesCollisionError = -72001,
    NSNetServicesNotFoundError    = -72002,
    NSNetServicesActivityInProgress = -72003,
    NSNetServicesBadArgumentError = -72004,
    NSNetServicesCancelledError = -72005,
    NSNetServicesInvalidError = -72006,
    NSNetServicesTimeoutError = -72007,
} NSNetServicesError;
```

Constants

`NSNetServicesUnknownError`

An unknown error occurred.

Available in Mac OS X v10.2 and later.

Declared in `NSNetServices.h`.

`NSNetServicesCollisionError`

The service could not be published because the name is already in use. The name could be in use locally or on another system.

Available in Mac OS X v10.2 and later.

Declared in `NSNetServices.h`.

`NSNetServicesNotFoundError`

The service could not be found on the network.

Available in Mac OS X v10.2 and later.

Declared in `NSNetServices.h`.

`NSNetServicesActivityInProgress`

The net service cannot process the request at this time. No additional information about the network state is known.

Available in Mac OS X v10.2 and later.

Declared in `NSNetServices.h`.

`NSNetServicesBadArgumentError`

An invalid argument was used when creating the `NSNetService` object.

Available in Mac OS X v10.2 and later.

Declared in `NSNetServices.h`.

`NSNetServicesCancelledError`

The client canceled the action.

Available in Mac OS X v10.2 and later.

Declared in `NSNetServices.h`.

`NSNetServicesInvalidError`

The net service was improperly configured.

Available in Mac OS X v10.2 and later.

Declared in `NSNetServices.h`.

`NSNetServicesTimeoutError`
The net service has timed out.

Available in Mac OS X v10.4 and later.

Declared in `NSNetServices.h`.

Declared In
`NSNetServices.h`

NSNetServiceOptions

These constants specify options for a network service.

```
enum {
   NSNetServiceNoAutoRename = 1 << 0
};
typedef NSUInteger NSNetServiceOptions;
```

Constants
`NSNetServiceNoAutoRename`
Specifies that the network service not rename itself in the event of a name collision.

Available in Mac OS X v10.5 and later.

Declared in `NSNetServices.h`.

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

Declared In
`NSNetServices.h`

NSNetServiceBrowser Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSNetServices.h
Availability	Available in Mac OS X v10.2 and later.
Companion guides	Bonjour Overview NSNetServices and CFNetServices Programming Guide
Related sample code	CocoaEcho GridCalendar PictureSharingBrowser

Overview

The `NSNetServiceBrowser` class defines an interface for finding published services on a network using multicast DNS. An instance of NSNetServiceBrowser is known as a **network service browser**.

Services can range from standard services, such as HTTP and FTP, to custom services defined by other applications. You can use a network service browser in your code to obtain the list of accessible domains and then to obtain an `NSNetService` object for each discovered service. Each network service browser performs one search at a time, so if you want to perform multiple simultaneous searches, use multiple network service browsers.

A network service browser performs all searches asynchronously using the current run loop to execute the search in the background. Results from a search are returned through the associated delegate object, which your client application must provide. Searching proceeds in the background until the object receives a `stop` (page 1025) message.

To use an `NSNetServiceBrowser` object to search for services, allocate it, initialize it, and assign a delegate. (If you wish, you can also use the `scheduleInRunLoop:forMode:` (page 1022) and `removeFromRunLoop:forMode:` (page 1022) methods to execute searches on a run loop other than the current one.) Once your object is ready, you begin by gathering the list of accessible domains using either the `searchForRegistrationDomains` (page 1023) or `searchForBrowsableDomains` (page 1023) methods. From the list of returned domains, you can pick one and use the `searchForServicesOfType:inDomain:` (page 1024) method to search for services in that domain.

The `NSNetServiceBrowser` class provides two ways to search for domains. In most cases, your client should use the `searchForRegistrationDomains` (page 1023) method to search only for local domains to which the host machine has registration authority. This is the preferred method for accessing domains as it guarantees that the host machine can connect to services in the returned domains. Access to domains outside this list may be more limited.

Tasks

Creating Network Service Browsers

- `init` (page 1021)
  Initializes an allocated `NSNetServiceBrowser` (page 1019) object.

Configuring Network Service Browsers

- `delegate` (page 1021)
  Returns the receiver's delegate.
- `setDelegate:` (page 1025)
  Sets the receiver's delegate.

Using Network Service Browsers

- `searchForBrowsableDomains` (page 1023)
  Initiates a search for domains visible to the host. This method returns immediately.
- `searchForRegistrationDomains` (page 1023)
  Initiates a search for domains in which the host may register services.
- `netServiceBrowser:didFindDomain:moreComing:` (page 1026) *delegate method*
  Tells the delegate the sender found a domain.
- `netServiceBrowser:didRemoveDomain:moreComing:` (page 1027) *delegate method*
  Tells the delegate the a domain has disappeared or has become unavailable.
- `searchForServicesOfType:inDomain:` (page 1024)
  Starts a search for services of a particular type within a specific domain.
- `netServiceBrowser:didFindService:moreComing:` (page 1026) *delegate method*
  Tells the delegate the sender found a service.
- `netServiceBrowser:didRemoveService:moreComing:` (page 1028) *delegate method*
  Tells the delegate a service has disappeared or has become unavailable.
- `netServiceBrowserWillSearch:` (page 1029) *delegate method*
  Tells the delegate that a serch is commencing.
- `netServiceBrowser:didNotSearch:` (page 1027) *delegate method*
  Tells the delegate that a search was not successful.
- `stop` (page 1025)
  Halts a currently running search or resolution.

- `netServiceBrowserDidStopSearch:` (page 1028) *delegate method*
  Tells the delegate that a search was stopped.
- `searchForAllDomains` (page 1023) Deprecated in Mac OS X v10.4
  Initiates a search for all domains that are visible to the host. (Deprecated. This method has been deprecated. Use `searchForBrowsableDomains` (page 1023) or `searchForRegistrationDomains` (page 1023) instead.)

Managing Run Loops

- `scheduleInRunLoop:forMode:` (page 1022)
  Adds the receiver to the specified run loop.
- `removeFromRunLoop:forMode:` (page 1022)
  Removes the receiver from the specified run loop.

Instance Methods

delegate

Returns the receiver's delegate.

```
- (id)delegate
```

Return Value
Delegate for the receiver.

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

See Also
`- setDelegate:` (page 1025)

Declared In
`NSNetServices.h`

init

Initializes an allocated `NSNetServiceBrowser` (page 1019) object.

```
- (id)init
```

Return Value
Initialized `NSNetServiceBrowser` (page 1019) object.

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

Declared In
`NSNetServices.h`

removeFromRunLoop:forMode:

Removes the receiver from the specified run loop.

```
- (void)removeFromRunLoop:(NSRunLoop *)runLoop forMode:(NSString *)runLoopMode
```

Parameters

runLoop

Run loop from which to remove the receiver.

runLoopMode

Run loop mode in which to perform this operation, such as `NSDefaultRunLoopMode`. See the "Constants" (page 1340) section of the `NSRunLoop` class for other run loop mode values.

Discussion

You can use this method in conjunction with `scheduleInRunLoop:forMode:` (page 1022) to transfer the receiver to a run loop other than the default one. Although it is possible to remove an `NSNetService` object completely from any run loop and then attempt actions on it, you must not do it.

Availability

Available in Mac OS X v10.2 and later.

See Also

`- scheduleInRunLoop:forMode:` (page 1022)

Declared In

`NSNetServices.h`

scheduleInRunLoop:forMode:

Adds the receiver to the specified run loop.

```
- (void)scheduleInRunLoop:(NSRunLoop *)runLoop forMode:(NSString *)runLoopMode
```

Parameters

runLoop

Run loop from which to remove the receiver.

runLoopMode

Run loop mode in which to perform this operation, such as `NSDefaultRunLoopMode`. See the "Constants" (page 1340) section of the `NSRunLoop` class for other run loop mode values.

Discussion

You can use this method in conjunction with `removeFromRunLoop:forMode:` (page 1022) to transfer the receiver to a run loop other than the default one. You should not attempt to run the receiver on multiple run loops.

Availability

Available in Mac OS X v10.2 and later.

See Also

`- removeFromRunLoop:forMode:` (page 1022)

Declared In

`NSNetServices.h`

searchForAllDomains

Initiates a search for all domains that are visible to the host. (Deprecated in Mac OS X v10.4. This method has been deprecated. Use `searchForBrowsableDomains` (page 1023) or `searchForRegistrationDomains` (page 1023) instead.)

```
- (void)searchForAllDomains
```

Discussion
This method returns immediately, sending a `netServiceBrowserWillSearch:` (page 1029) message to the delegate if the network was ready to initiate the search. The delegate receives a subsequent `netServiceBrowser:didFindDomain:moreComing:` (page 1026) message for each domain discovered.

This method may find domains in which the localhost does not have registration authority.

Availability
Deprecated in Mac OS X v10.4.

See Also
- `searchForRegistrationDomains` (page 1023)
- `netServiceBrowser:didFindDomain:moreComing:` (page 1026)

Declared In
`NSNetServices.h`

searchForBrowsableDomains

Initiates a search for domains visible to the host. This method returns immediately.

```
- (void)searchForBrowsableDomains
```

Discussion
The delegate receives a `netServiceBrowser:didFindDomain:moreComing:` (page 1026) message for each domain discovered.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `searchForRegistrationDomains` (page 1023)

Declared In
`NSNetServices.h`

searchForRegistrationDomains

Initiates a search for domains in which the host may register services.

```
- (void)searchForRegistrationDomains
```

Discussion
This method returns immediately, sending a `netServiceBrowserWillSearch:` (page 1029) message to the delegate if the network was ready to initiate the search. The delegate receives a subsequent `netServiceBrowser:didFindDomain:moreComing:` (page 1026) message for each domain discovered.

Most network service browser clients do not have to use this method—it is sufficient to publish a service with the empty string, which registers it in any available registration domains automatically.

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

See Also
- `searchForBrowsableDomains` (page 1023)
- `searchForServicesOfType:inDomain:` (page 1024)
- `netServiceBrowser:didFindDomain:moreComing:` (page 1026)
- `netServiceBrowserWillSearch:` (page 1029)

Declared In
`NSNetServices.h`

searchForServicesOfType:inDomain:

Starts a search for services of a particular type within a specific domain.

```
- (void)searchForServicesOfType:(NSString *)serviceType inDomain:(NSString
    *)domainName
```

Parameters
serviceType
Type of the service to search for.

domainName
Domain name in which to perform the search.

Discussion
This method returns immediately, sending a `netServiceBrowserWillSearch:` (page 1029) message to the delegate if the network was ready to initiate the search.The delegate receives subsequent `netServiceBrowser:didFindService:moreComing:` (page 1026) messages for each service discovered.

The *serviceType* argument must contain both the service type and transport layer information. To ensure that the mDNS responder searches for services, rather than hosts, make sure to prefix both the service name and transport layer name with an underscore character ("_"). For example, to search for an HTTP service on TCP, you would use the type string "`_http._tcp.`". Note that the period character at the end is required.

The *domainName* argument can be an explicit domain name, the generic local domain `@"local."` (note trailing period, which indicates an absolute name), or the empty string (`@""`), which indicates the default registration domains. Usually, you pass in an empty string. Note that it is acceptable to use an empty string for the *domainName* argument when publishing or browsing a service, but do not rely on this for resolution.

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

See Also
- `netServiceBrowser:didFindService:moreComing:` (page 1026)

- netServiceBrowserWillSearch: (page 1029)

Declared In
NSNetServices.h

setDelegate:

Sets the receiver's delegate.

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

Parameters

delegate

Object to serve as the receiver's delegate. Must not be nil.

Discussion
The delegate is not retained. The receiver calls the methods of your delegate to receive information about discovered domains and services.

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

See Also
- delegate (page 1021)

Declared In
NSNetServices.h

stop

Halts a currently running search or resolution.

- (void)stop

Discussion
This method sends a netServiceBrowserDidStopSearch: (page 1028) message to the delegate and causes the browser to discard any pending search results.

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

See Also
- netServiceBrowserDidStopSearch: (page 1028)

Declared In
NSNetServices.h

Delegate Methods

netServiceBrowser:didFindDomain:moreComing:

Tells the delegate the sender found a domain.

```
- (void)netServiceBrowser:(NSNetServiceBrowser *)netServiceBrowser
    didFindDomain:(NSString *)domainName moreComing:(BOOL)moreDomainsComing
```

Parameters

netServiceBrowser
Sender of this delegate message.

domainName
Name of the domain found by *netServiceBrowser*.

moreDomainsComing
YES when *netServiceBrowser* is waiting for additional domains. NO when there are no additional domains.

Discussion
The delegate uses this message to compile a list of available domains. It should wait until *moreDomainsComing* is NO to do a bulk update of user interface elements.

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

See Also
- searchForBrowsableDomains (page 1023)
- searchForRegistrationDomains (page 1023)

Declared In
NSNetServices.h

netServiceBrowser:didFindService:moreComing:

Tells the delegate the sender found a service.

```
- (void)netServiceBrowser:(NSNetServiceBrowser *)netServiceBrowser
    didFindService:(NSNetService *)netService moreComing:(BOOL)moreServicesComing
```

Parameters

netServiceBrowser
Sender of this delegate message.

netService
Network service found by *netServiceBrowser*. The delegate can use this object to connect to and use the service.

moreServicesComing
YES when *netServiceBrowser* is waiting for additional services. NO when there are no additional services.

Discussion
The delegate uses this message to compile a list of available services. It should wait until *moreServicesComing* is NO to do a bulk update of user interface elements.

Special Considerations

If the delegate chooses to resolve *netService*, it should retain *netService* and set itself as that service's delegate. The delegate should, therefore, release that service when it receives the `netServiceDidResolveAddress:` (page 1014) or `netService:didNotResolve:` (page 1012) delegate messages of the `NSNetService` class.

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

See Also
`- searchForServicesOfType:inDomain:` (page 1024)

Declared In
`NSNetServices.h`

netServiceBrowser:didNotSearch:

Tells the delegate that a search was not successful.

```
- (void)netServiceBrowser:(NSNetServiceBrowser *)netServiceBrowser
    didNotSearch:(NSDictionary *)errorInfo
```

Parameters

netServiceBrowser
Sender of this delegate message.

errorInfo
Dictionary with the reasons the search was unsuccessful. Use the dictionary keys `NSNetServicesErrorCode` and `NSNetServicesErrorDomain` to retrieve the error information from the dictionary.

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

See Also
`- netServiceBrowserWillSearch:` (page 1029)

Declared In
`NSNetServices.h`

netServiceBrowser:didRemoveDomain:moreComing:

Tells the delegate the a domain has disappeared or has become unavailable.

```
- (void)netServiceBrowser:(NSNetServiceBrowser *)netServiceBrowser
    didRemoveDomain:(NSString *)domainName moreComing:(BOOL)moreDomainsComing
```

Parameters

netServiceBrowser

Sender of this delegate message.

domainName

Name of the domain that became unavailable.

moreDomainsComing

YES when *netServiceBrowser* is waiting for additional domains. NO when there are no additional domains.

Discussion

The delegate uses this message to compile a list of unavailable domains. It should wait until *moreDomainsComing* is NO to do a bulk update of user interface elements.

Availability

Available in Mac OS X v10.2 and later.

Declared In

NSNetServices.h

netServiceBrowser:didRemoveService:moreComing:

Tells the delegate a service has disappeared or has become unavailable.

```
- (void)netServiceBrowser:(NSNetServiceBrowser *)netServiceBrowser
    didRemoveService:(NSNetService *)netService moreComing:(BOOL)moreServicesComing
```

Parameters

netServiceBrowser

Sender of this delegate message.

netService

Network service that has become unavailable.

moreServicesComing

YES when *netServiceBrowser* is waiting for additional services. NO when there are no additional services.

Discussion

The delegate uses this message to compile a list of unavailable services. It should wait until *moreServicesComing* is NO to do a bulk update of user interface elements.

Availability

Available in Mac OS X v10.2 and later.

Declared In

NSNetServices.h

netServiceBrowserDidStopSearch:

Tells the delegate that a search was stopped.

```
- (void)netServiceBrowserDidStopSearch:(NSNetServiceBrowser *)netServiceBrowser
```

Parameters

netServiceBrowser

Sender of this delegate message.

Discussion

When *netServiceBrowser* receives a `stop` (page 1025) message from its client, *netServiceBrowser* sends a `netServiceBrowserDidStopSearch:` message to its delegate. The delegate then performs any necessary cleanup.

Availability

Available in Mac OS X v10.2 and later.

See Also

`- stop` (page 1025)

Declared In

`NSNetServices.h`

netServiceBrowserWillSearch:

Tells the delegate that a serch is commencing.

```
- (void)netServiceBrowserWillSearch:(NSNetServiceBrowser *)netServiceBrowser
```

Parameters

netServiceBrowser

Sender of this delegate message.

Discussion

This message is sent to the delegate only if the underlying network layer is ready to begin a search. The delegate can use this notification to prepare its data structures to receive data.

Availability

Available in Mac OS X v10.2 and later.

See Also

`- netServiceBrowser:didNotSearch:` (page 1027)

Declared In

`NSNetServices.h`

NSNotification Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSNotification.h
Companion guide	Notification Programming Topics for Cocoa
Related sample code	EnhancedAudioBurn PDFKitLinker2 Quartz Composer WWDC 2005 TextEdit Sketch-112 TextEditPlus

Overview

NSNotification objects encapsulate information so that it can be broadcast to other objects by an NSNotificationCenter object. An NSNotification object (referred to as a notification) contains a name, an object, and an optional dictionary. The name is a tag identifying the notification. The object is any object that the poster of the notification wants to send to observers of that notification (typically, it is the object that posted the notification). The dictionary stores other related objects, if any. NSNotification objects are immutable objects.

You can create a notification object with the class methods `notificationWithName:object:` (page 1033) or `notificationWithName:object:userInfo:` (page 1033). However, you don't usually create your own notifications directly. The NSNotificationCenter methods `postNotificationName:object:` (page 1042) and `postNotificationName:object:userInfo:` (page 1043) allow you to conveniently post a notification without creating it first.

NSCopying Protocol

The NSNotification class adopts the NSCopying protocol, making it possible to treat notifications as context-independent values that can be copied and reused. You can store a notification for later use or use the distributed objects system to send a notification to another process. The NSCopying protocol essentially allows clients to deal with notifications as first class values that can be copied by collections. You can put notifications in an array and send the `copy` message to that array, which recursively copies every item.

Creating Subclasses

You can subclass NSNotification to contain information in addition to the notification name, object, and dictionary. This extra data must be agreed upon between notifiers and observers.

NSNotification is a class cluster with no instance variables. As such, you must subclass NSNotification and override the primitive methods `name` (page 1034), `object` (page 1034), and `userInfo` (page 1035). You can choose any designated initializer you like, but be sure that your initializer does not call NSNotification's implementation of `init` (via `[super init]`). NSNotification is not meant to be instantiated directly, and its `init` method raises an exception.

Adopted Protocols

NSCoding
- `- encodeWithCoder:` (page 2034)
- `- initWithCoder:` (page 2034)

NSCopying
- `- copyWithZone:` (page 2042)

Tasks

Creating Notifications

`+ notificationWithName:object:` (page 1033)
Returns a new notification object with a specified name and object.

`+ notificationWithName:object:userInfo:` (page 1033)
Returns a notification object with a specified name, object, and user information.

Getting Notification Information

`- name` (page 1034)
Returns the name of the notification.

- `object` (page 1034)
  Returns the object associated with the notification.
- `userInfo` (page 1035)
  Returns the user information dictionary associated with the receiver.

Class Methods

notificationWithName:object:

Returns a new notification object with a specified name and object.

```
+ (id)notificationWithName:(NSString *)aName object:(id)anObject
```

Parameters

aName
  The name for the new notification. May not be `nil`.

anObject
  The object for the new notification.

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

See Also
`- postNotificationName:object:` (page 1042) (NSNotificationCenter)

Related Sample Code
ExtractMovieAudioToAIFF
Link Snoop
MyPhoto
QTExtractAndConvertToAIFF
QTExtractAndConvertToMovieFile

Declared In
`NSNotification.h`

notificationWithName:object:userInfo:

Returns a notification object with a specified name, object, and user information.

```
+ (id)notificationWithName:(NSString *)aName object:(id)anObject
    userInfo:(NSDictionary *)userInfo
```

Parameters

aName
  The name for the new notification. May not be `nil`.

anObject
  The object for the new notification.

userInfo
The user information dictionary for the new notification. May be `nil`.

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

See Also
`+ notificationWithName:object:` (page 1033)
`- postNotificationName:object:userInfo:` (page 1043) (NSNotificationCenter)

Related Sample Code
People

Declared In
`NSNotification.h`

Instance Methods

name

Returns the name of the notification.

```
- (NSString *)name
```

Return Value
The name of the notification. Typically you use this method to find out what kind of notification you are dealing with when you receive a notification.

Special Considerations

Notification names can be any string. To avoid name collisions, you might want to use a prefix that's specific to your application.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CoreRecipes
People
QTAudioExtractionPanel
WhackedTV

Declared In
`NSNotification.h`

object

Returns the object associated with the notification.

```
- (id)object
```

Return Value
The object associated with the notification. This is often the object that posted this notification. It may be `nil`.

Typically you use this method to find out what object a notification applies to when you receive a notification.

Discussion
For example, suppose you've registered an object to receive the message `handlePortDeath:` when the "PortInvalid" notification is posted to the notification center and that `handlePortDeath:` needs to access the object monitoring the port that is now invalid. `handlePortDeath:` can retrieve that object as shown here:

```
- (void)handlePortDeath:(NSNotification *)notification
{
    ...
    [self reclaimResourcesForPort:[notification object]];
    ...
}
```

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ABPresence
NewsReader
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus

Declared In
`NSNotification.h`

userInfo

Returns the user information dictionary associated with the receiver.

```
- (NSDictionary *)userInfo
```

Return Value
Returns the user information dictionary associated with the receiver. May be `nil`.

The user information dictionary stores any additional objects that objects receiving the notification might use.

Discussion
For example, in the Application Kit, NSControl objects post the `NSControlTextDidChangeNotification` whenever the field editor (an NSText object) changes text inside the NSControl. This notification provides the NSControl object as the notification's associated object. In order to provide access to the field editor, the NSControl object posting the notification adds the field editor to the notification's user information dictionary. Objects receiving the notification can access the field editor and the NSControl object posting the notification as follows:

```
- (void)controlTextDidBeginEditing:(NSNotification *)notification
{
```

```
    NSText *fieldEditor = [[notification userInfo]
        objectForKey:@"NSFieldEditor"];               // the field editor
    NSControl *postingObject = [notification object]; // the object that posted
 the notification
    ...
}
```

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ABPresence
CoreRecipes
PDFKitLinker2
SimpleCalendar
WhackedTV

Declared In
`NSNotification.h`

NSNotificationCenter Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSNotification.h
Companion guide	Notification Programming Topics for Cocoa
Related sample code	EnhancedAudioBurn PDF Annotation Editor Quartz Composer WWDC 2005 TextEdit Sketch-112 TextEditPlus

Class at a Glance

The `NSNotificationCenter` class provides a way to send notifications to objects in the same task. It takes `NSNotification` objects and broadcasts them to any objects in the same task that have registered to receive the notification with the task's default notification center.

Principal Attributes

- **Notification dispatch table.** Each entry in this table specifies a notification set for a particular observer. A notification set is a subset of the notifications posted to the notification center. Each table entry contains three items:
  - **Notification observer:** Required. The object to be notified when qualifying notifications are posted to the notification center.
  - **Notification name:** Optional. Specifying a name reduces the set of notifications the entry specifies to those that have this name.
  - **Notification sender:** Optional. Specifying a sender reduces the set of notifications the entry specifies to those sent by this object.

  Table 92-1 shows the four types of dispatch table entries and the notification sets they specify. (This table omits the always present notification observer.)

Table 92-1 Types of dispatch table entries

Notification name	Notification sender	Notification set specified
Specified	Specified	Notifications with a particular name from a specific sender.
Specified	Unspecified	Notifications with a particular name by any sender.
Unspecified	Specified	Notifications posted by a specific sender.
Unspecified	Unspecified	All notifications.

Table 92-2 shows an example dispatch table with four observers.

Table 92-2 Example notification dispatch table

Observer	Notification name	Notification sender
`observerA`	`NSFileHandleReadCompletionNotification`	`nil`
`observerB`	`nil`	`addressTableView`
`observerC`	`NSWindowDidChangeScreenNotification`	`documentWindow`
`observerC`	`nil`	`addressTableView`
`observerD`	`nil`	`nil`

When notifications are posted to the notification center, each of the observers in Table 92-2 are notified of the following notifications:

- `observerA`: Notifications named `NSFileHandleReadCompletionNotification`.
- `observerB`: Notifications sent by `addressTableView`.
- `observerC`: Notifications named `NSWindowDidChangeScreenNotification` sent by `documentWindow` and notifications sent by `addressTableView`.
- `observerD`: All notifications.

Commonly Used Methods

`defaultCenter` (page 1040)
Returns the task's default notification center.

`addObserver:selector:name:object:` (page 1041)
Adds an entry to the notification center's dispatch table specifying at least an observer and a notification message.

`postNotificationName:object:` (page 1042)
Creates and posts a notification to the notification center.

`removeObserver:` (page 1043)
Removes all entries from the notification center's dispatch center that specify a particular observer, so that it no longer receives notifications posted to that notification center.

Overview

An `NSNotificationCenter` object (or simply, **notification center**) provides a mechanism for broadcasting information within a task. An `NSNotificationCenter` object is essentially a notification dispatch table.

Objects register with a notification center to receive notifications (`NSNotification` objects) using the `addObserver:selector:name:object:` (page 1041) method. Each invocation of this method specifies a set of notifications. Therefore, objects may register as observers of different notification sets by calling `addObserver:selector:name:object:` several times.

When an object (known as the **notification sender**) posts a notification, it sends an `NSNotification` object to the notification center. The notification center then notifies any observers for which the notification meets the criteria specified on registration by sending them the specified notification message, passing the notification as the sole argument. The order in which observers receive notifications is undefined. It is possible for the posting object and the observing object to be the same.

A notification center delivers notifications to observers synchronously. In other words, the `postNotification:` (page 1041) methods do not return until all observers have received and processed the notification. To send notifications asynchronously use `NSNotificationQueue`. In a multithreaded application, notifications are always delivered in the thread in which the notification was posted, which may not be the same thread in which an observer registered itself.

Important: The notification center does not retain its observers, therefore, you must ensure that you unregister observers (using `removeObserver:` (page 1043) or `removeObserver:name:object:` (page 1044)) before they are deallocated. (If you don't, you will generate a runtime error if the center sends a message to a freed object.)

Each task has a default notification center. You typically don't create your own. An `NSNotificationCenter` object can deliver notifications only within a single task. If you want to post a notification to other tasks or receive notifications from other tasks, use a `NSDistributedNotificationCenter` object.

Tasks

Getting the Notification Center

`+ defaultCenter` (page 1040)
Returns the task's default notification center.

Managing Notification Observers

- `addObserver:selector:name:object:` (page 1041)
  Adds an entry to the receiver's dispatch table with an observer, a notification selector and optional criteria: notification name and sender.
- `removeObserver:` (page 1043)
  Removes all the entries specifying a given observer from the receiver's dispatch table.
- `removeObserver:name:object:` (page 1044)
  Removes matching entries from the receiver's dispatch table.

Posting Notifications

- `postNotification:` (page 1041)
  Posts a given notification to the receiver.
- `postNotificationName:object:` (page 1042)
  Creates a notification with a given name and sender and posts it to the receiver.
- `postNotificationName:object:userInfo:` (page 1043)
  Creates a notification with a given name, sender, and information and posts it to the receiver.

Class Methods

defaultCenter

Returns the task's default notification center.

```
+ (id)defaultCenter
```

Return Value
The current task's default notification center, which is used for system notifications.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Dicey
PDF Annotation Editor
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus

Declared In
`NSNotification.h`

Instance Methods

addObserver:selector:name:object:

Adds an entry to the receiver's dispatch table with an observer, a notification selector and optional criteria: notification name and sender.

```
- (void)addObserver:(id)notificationObserver selector:(SEL)notificationSelector
    name:(NSString *)notificationName object:(id)notificationSender
```

Parameters

notificationObserver

Object registering as an observer. Must not be `nil`.

notificationSelector

Selector that specifies the message the receiver sends *notificationObserver* to notify it of the notification posting. The method the selector specifies must have one and only one argument.

notificationName

The name of the notification for which to register the observer; that is, only notifications with this name are delivered to the observer. When `nil`, the notification center doesn't use a notification's name to decide whether to deliver it to the observer.

notificationSender

The object whose notifications the observer wants to receive; that is, only notifications sent by this sender are delivered to the observer. When `nil`, the notification center doesn't use a notification's sender to decide whether to deliver it to the observer.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- removeObserver:` (page 1043)

Related Sample Code
ABPresence
CocoaDVDPlayer
ImageMapExample
QTAudioExtractionPanel
VideoViewer

Declared In
`NSNotification.h`

postNotification:

Posts a given notification to the receiver.

```
- (void)postNotification:(NSNotification *)notification
```

Parameters

notification

The notification to post. This value must not be `nil`.

Discussion

You can create a notification with the `NSNotification` class method `notificationWithName:object:` (page 1033) or `notificationWithName:object:userInfo:` (page 1033). An exception is raised if *notification* is `nil`.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `postNotificationName:object:` (page 1042)
- `postNotificationName:object:userInfo:` (page 1043)

Related Sample Code

ExtractMovieAudioToAIFF
QTExtractAndConvertToAIFF
QTExtractAndConvertToMovieFile
QTKitTimeCode

Declared In

`NSNotification.h`

postNotificationName:object:

Creates a notification with a given name and sender and posts it to the receiver.

```
- (void)postNotificationName:(NSString *)notificationName
    object:(id)notificationSender
```

Parameters

notificationName

The name of the notification.

notificationSender

The object posting the notification.

Discussion

This method invokes `postNotificationName:object:userInfo:` (page 1043) with a *userInfo* argument of `nil`.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `postNotification:` (page 1041)

Declared In

`NSNotification.h`

postNotificationName:object:userInfo:

Creates a notification with a given name, sender, and information and posts it to the receiver.

```
- (void)postNotificationName:(NSString *)notificationName
    object:(id)notificationSender userInfo:(NSDictionary *)userInfo
```

Parameters

notificationName

The name of the notification.

notificationSender

The object posting the notification.

userInfo

Information about the the notification. May be `nil`.

Discussion

This method is the preferred method for posting notifications.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- postNotificationName:object:` (page 1042)

Declared In

`NSNotification.h`

removeObserver:

Removes all the entries specifying a given observer from the receiver's dispatch table.

```
- (void)removeObserver:(id)notificationObserver
```

Parameters

notificationObserver

The observer to remove. Must not be `nil`.

Discussion

Be sure to invoke this method (or `removeObserver:name:object:` (page 1044)) before *notificationObserver* or any object specified in `addObserver:selector:name:object:` (page 1041) is deallocated.

The following example illustrates how to unregister `someObserver` for all notifications for which it had previously registered:

```
[[NSNotificationCenter defaultCenter] removeObserver:someObserver];
```

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

GridCalendar

Quartz Composer WWDC 2005 TextEdit

Sketch-112

TextEditPlus
WhackedTV

Declared In
NSNotification.h

removeObserver:name:object:

Removes matching entries from the receiver's dispatch table.

```
- (void)removeObserver:(id)notificationObserver name:(NSString *)notificationName
    object:(id)notificationSender
```

Parameters

notificationObserver

Observer to remove from the dispatch table. Specify an observer to remove only entries for this observer. Must not be `nil`, or message will have no effect.

notificationName

Name of the notification to remove from dispatch table. Specify a notification name to remove only entries that specify this notification name. When `nil`, the receiver does not use notification names as criteria for removal.

notificationSender

Sender to remove from the dispatch table. Specify a notification sender to remove only entries that specify this sender. When `nil`, the receiver does not use notification senders as criteria for removal.

Discussion
Be sure to invoke this method (or `removeObserver:` (page 1043)) before the observer object or any object specified in `addObserver:selector:name:object:` (page 1041) is deallocated.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ImageMapExample
People
QTAudioExtractionPanel
QTKitMovieShuffler
VideoViewer

Declared In
NSNotification.h

NSNotificationQueue Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSNotificationQueue.h
Companion guide	Notification Programming Topics for Cocoa
Related sample code	Link Snoop

Overview

NSNotificationQueue objects (or simply notification queues) act as buffers for notification centers (instances of NSNotificationCenter). Whereas a notification center distributes notifications when posted, notifications placed into the queue can be delayed until the end of the current pass through the run loop or until the run loop is idle. Duplicate notifications can also be coalesced so that only one notification is sent although multiple notifications are posted. A notification queue maintains notifications (instances of NSNotification) generally in a first in first out (FIFO) order. When a notification rises to the front of the queue, the queue posts it to the notification center, which in turn dispatches the notification to all objects registered as observers.

Every thread has a default notification queue, which is associated with the default notification center for the task. You can create your own notification queues and have multiple queues per center and thread.

Tasks

Creating Notification Queues

`- initWithNotificationCenter:` (page 1048)
Initializes and returns a notification queue for the specified notification center.

Getting the Default Queue

`+ defaultQueue` (page 1046)
Returns the default notification queue for the current thread.

Managing Notifications

`- enqueueNotification:postingStyle:` (page 1047)
Adds a notification to the notification queue with a specified posting style.

`- enqueueNotification:postingStyle:coalesceMask:forModes:` (page 1047)
Adds a notification to the notification queue with a specified posting style, criteria for coalescing, and runloop mode.

`- dequeueNotificationsMatching:coalesceMask:` (page 1046)
Removes all notifications from the queue that match a provided notification using provided matching criteria.

Class Methods

defaultQueue

Returns the default notification queue for the current thread.

```
+ (NSNotificationQueue *)defaultQueue
```

Return Value
Returns the default notification queue for the current thread. This notification queue uses the default notification center.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Link Snoop

Declared In
`NSNotificationQueue.h`

Instance Methods

dequeueNotificationsMatching:coalesceMask:

Removes all notifications from the queue that match a provided notification using provided matching criteria.

```
- (void)dequeueNotificationsMatching:(NSNotification *)notification
    coalesceMask:(NSUInteger)coalesceMask
```

Parameters

notification

The notification used for matching notifications to remove from the notification queue.

coalesceMask

A mask indicating what criteria to use when matching attributes of *notification* to attributes of notifications in the queue. The mask is created by combining any of the constants `NSNotificationNoCoalescing`, `NSNotificationCoalescingOnName`, and `NSNotificationCoalescingOnSender`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSNotificationQueue.h`

enqueueNotification:postingStyle:

Adds a notification to the notification queue with a specified posting style.

```
- (void)enqueueNotification:(NSNotification *)notification
    postingStyle:(NSPostingStyle)postingStyle
```

Parameters

notification

The notification to add to the queue.

postingStyle

The posting style for the notification. The posting style indicates when the notification queue should post the notification to its notification center.

Discussion
Notifications added with this method are posted using the runloop mode `NSDefaultRunLoopMode` and coalescing criteria that will coalesce only notifications that match both the notification's name and object.

This method invokes `enqueueNotification:postingStyle:coalesceMask:forModes:` (page 1047).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Link Snoop

Declared In
`NSNotificationQueue.h`

enqueueNotification:postingStyle:coalesceMask:forModes:

Adds a notification to the notification queue with a specified posting style, criteria for coalescing, and runloop mode.

```
- (void)enqueueNotification:(NSNotification *)notification
    postingStyle:(NSPostingStyle)postingStyle coalesceMask:(NSUInteger)coalesceMask
     forModes:(NSArray *)modes
```

Parameters

notification

The notification to add to the queue.

postingStyle

The posting style for the notification. The posting style indicates when the notification queue should post the notification to its notification center.

coalesceMask

A mask indicating what criteria to use when matching attributes of *notification* to attributes of notifications in the queue. The mask is created by combining any of the constants `NSNotificationNoCoalescing`, `NSNotificationCoalescingOnName`, and `NSNotificationCoalescingOnSender`.

modes

The list of modes the notification may be posted in. The notification queue will only post the notification to its notification center if the run loops is in one of the modes provided in the array. May be `nil`, in which case it defaults to `NSDefaultRunLoopMode`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSNotificationQueue.h`

initWithNotificationCenter:

Initializes and returns a notification queue for the specified notification center.

```
- (id)initWithNotificationCenter:(NSNotificationCenter *)notificationCenter
```

Parameters

notificationCenter

The notification center used by the new notification queue.

Return Value
The newly initialized notification queue.

Discussion
This is the designated initializer for the NSNotificationQueue class.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSNotificationQueue.h`

Constants

NSNotificationCoalescing

These constants specify how notifications are coalesced.

```
typedef enum {
    NSNotificationNoCoalescing = 0,
    NSNotificationCoalescingOnName = 1,
    NSNotificationCoalescingOnSender = 2
} NSNotificationCoalescing;
```

Constants

`NSNotificationNoCoalescing`

Do not coalesce notifications in the queue.

Available in Mac OS X v10.0 and later.

Declared in `NSNotificationQueue.h`.

`NSNotificationCoalescingOnName`

Coalesce notifications with the same name.

Available in Mac OS X v10.0 and later.

Declared in `NSNotificationQueue.h`.

`NSNotificationCoalescingOnSender`

Coalesce notifications with the same object.

Available in Mac OS X v10.0 and later.

Declared in `NSNotificationQueue.h`.

Discussion

These constants are used in the third argument of `enqueueNotification:postingStyle:coalesceMask:forModes:` (page 1047). You can OR them together to specify more than one.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSNotificationQueue.h`

NSPostingStyle

These constants specify when notifications are posted.

```
typedef enum {
    NSPostWhenIdle = 1,
    NSPostASAP = 2,
    NSPostNow = 3
} NSPostingStyle;
```

Constants

`NSPostASAP`

The notification is posted at the end of the current notification callout or timer.

Available in Mac OS X v10.0 and later.

Declared in `NSNotificationQueue.h`.

`NSPostWhenIdle`

The notification is posted when the run loop is idle.

Available in Mac OS X v10.0 and later.

Declared in `NSNotificationQueue.h`.

`NSPostNow`

The notification is posted immediately after coalescing.

Available in Mac OS X v10.0 and later.

Declared in `NSNotificationQueue.h`.

Discussion

These constants are used in both `enqueueNotification:postingStyle:` (page 1047) and `enqueueNotification:postingStyle:coalesceMask:forModes:` (page 1047).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSNotificationQueue.h`

NSNull Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSNull.h
Companion guide	Number and Value Programming Topics for Cocoa
Related sample code	Apply Firmware Password MyPhoto QTQuartzPlayer SimpleCalendar

Overview

The `NSNull` class defines a singleton object used to represent null values in collection objects (which don't allow `nil` values).

Adopted Protocols

NSCoding

- `- encodeWithCoder:` (page 2034)
- `- initWithCoder:` (page 2034)

NSCopying

- `- copyWithZone:` (page 2042)

Tasks

Obtaining an Instance

`+ null` (page 1052)
Returns the singleton instance of `NSNull`.

Class Methods

null

Returns the singleton instance of `NSNull`.

```
+ (NSNull *)null
```

Return Value
The singleton instance of `NSNull`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Apply Firmware Password
MyPhoto
QTQuartzPlayer

Declared In
`NSNull.h`

NSNumber Class Reference

Inherits from	NSValue : NSObject
Conforms to	NSCoding (NSValue) NSCopying (NSValue) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSValue.h Foundation/NSDecimalNumber.h
Companion guides	Number and Value Programming Topics for Cocoa Property List Programming Guide
Related sample code	Dicey QTCoreVideo301 Quartz Composer WWDC 2005 TextEdit SimpleScriptingObjects TextEditPlus

Overview

`NSNumber` is a subclass of `NSValue` that offers a value as any C scalar (numeric) type. It defines a set of methods specifically for setting and accessing the value as a signed or unsigned `char`, `short int`, `int`, `long int`, `long long int`, `float`, or `double` or as a `BOOL`. (Note that number objects do not necessarily preserve the type they are created with.) It also defines a `compare:` (page 1064) method to determine the ordering of two `NSNumber` objects.

Creating a Subclass of NSNumber

As with any class cluster, if you create a subclass of `NSNumber`, you have to override the primitive methods of its superclass, `NSValue`. Furthermore, there is a restricted set of return values that your implementation of the `NSValue` method `objCType` can return, in order to take advantage of the abstract implementations of the non-primitive methods. The valid return values are "`c`", "`C`", "`s`", "`S`", "`i`", "`I`", "`l`", "`L`", "`q`", "`Q`", "`f`", and "`d`".

Tasks

Creating an NSNumber Object

`+ numberWithBool:` (page 1057)
Creates and returns an `NSNumber` object containing a given value, treating it as a `BOOL`.

`+ numberWithChar:` (page 1057)
Creates and returns an `NSNumber` object containing a given value, treating it as a signed `char`.

`+ numberWithDouble:` (page 1057)
Creates and returns an `NSNumber` object containing a given value, treating it as a `double`.

`+ numberWithFloat:` (page 1058)
Creates and returns an `NSNumber` object containing a given value, treating it as a `float`.

`+ numberWithInt:` (page 1058)
Creates and returns an `NSNumber` object containing a given value, treating it as a signed `int`.

`+ numberWithInteger:` (page 1059)
Creates and returns an `NSNumber` object containing a given value, treating it as an `NSInteger`.

`+ numberWithLong:` (page 1059)
Creates and returns an `NSNumber` object containing a given value, treating it as a signed `long`.

`+ numberWithLongLong:` (page 1060)
Creates and returns an `NSNumber` object containing a given value, treating it as a signed `long long`.

`+ numberWithShort:` (page 1060)
Creates and returns an `NSNumber` object containing *`value`*, treating it as a signed `short`.

`+ numberWithUnsignedChar:` (page 1061)
Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned char`.

`+ numberWithUnsignedInt:` (page 1061)
Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned int`.

`+ numberWithUnsignedInteger:` (page 1062)
Creates and returns an `NSNumber` object containing a given value, treating it as an `NSUInteger`.

`+ numberWithUnsignedLong:` (page 1062)
Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned long`.

`+ numberWithUnsignedLongLong:` (page 1063)
Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned long long`.

`+ numberWithUnsignedShort:` (page 1063)
Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned short`.

Initializing an NSNumber Object

`- initWithBool:` (page 1067)
Returns an `NSNumber` object initialized to contain a given value, treated as a `BOOL`.

`- initWithChar:` (page 1068)
Returns an `NSNumber` object initialized to contain a given value, treated as a signed `char`.

- `initWithDouble:` (page 1068)
  Returns an `NSNumber` object initialized to contain *`value`*, treated as a `double`.
- `initWithFloat:` (page 1068)
  Returns an `NSNumber` object initialized to contain a given value, treated as a `float`.
- `initWithInt:` (page 1069)
  Returns an `NSNumber` object initialized to contain a given value, treated as a signed `int`.
- `initWithInteger:` (page 1069)
  Returns an `NSNumber` object initialized to contain a given value, treated as an `NSInteger`.
- `initWithLong:` (page 1069)
  Returns an `NSNumber` object initialized to contain a given value, treated as a signed `long`.
- `initWithLongLong:` (page 1070)
  Returns an `NSNumber` object initialized to contain *`value`*, treated as a signed `long long`.
- `initWithShort:` (page 1070)
  Returns an `NSNumber` object initialized to contain a given value, treated as a signed `short`.
- `initWithUnsignedChar:` (page 1070)
  Returns an `NSNumber` object initialized to contain a given value, treated as an `unsigned char`.
- `initWithUnsignedInt:` (page 1071)
  Returns an `NSNumber` object initialized to contain a given value, treated as an `unsigned int`.
- `initWithUnsignedInteger:` (page 1071)
  Returns an `NSNumber` object initialized to contain a given value, treated as an `NSUInteger`.
- `initWithUnsignedLong:` (page 1072)
  Returns an `NSNumber` object initialized to contain a given value, treated as an `unsigned long`.
- `initWithUnsignedLongLong:` (page 1072)
  Returns an `NSNumber` object initialized to contain a given value, treated as an `unsigned long long`.
- `initWithUnsignedShort:` (page 1072)
  Returns an `NSNumber` object initialized to contain a given value, treated as an `unsigned short`.

Accessing Numeric Values

- `boolValue` (page 1064)
  Returns the receiver's value as a `BOOL`.
- `charValue` (page 1064)
  Returns the receiver's value as a `char`.
- `decimalValue` (page 1065)
  Returns the receiver's value, expressed as an `NSDecimal` structure.
- `doubleValue` (page 1066)
  Returns the receiver's value as a `double`.
- `floatValue` (page 1067)
  Returns the receiver's value as a `float`.
- `intValue` (page 1073)
  Returns the receiver's value as an `int`.
- `integerValue` (page 1073)
  Returns the receiver's value as an `NSInteger`.

- `longLongValue` (page 1074)
  Returns the receiver's value as a `long long`.
- `longValue` (page 1074)
  Returns the receiver's value as a `long`.
- `shortValue` (page 1075)
  Returns the receiver's value as a `short`.
- `unsignedCharValue` (page 1076)
  Returns the receiver's value as an unsigned `char`.
- `unsignedIntegerValue` (page 1076)
  Returns the receiver's value as an `NSUInteger`.
- `unsignedIntValue` (page 1076)
  Returns the receiver's value as an unsigned `int`.
- `unsignedLongLongValue` (page 1077)
  Returns the receiver's value as an unsigned `long long`.
- `unsignedLongValue` (page 1077)
  Returns the receiver's value as an unsigned `long`.
- `unsignedShortValue` (page 1077)
  Returns the receiver's value as an unsigned `short`.

Retrieving String Representations

- `descriptionWithLocale:` (page 1065)
  Returns a string that represents the contents of the receiver for a given locale.
- `stringValue` (page 1075)
  Returns the receiver's value as a human-readable string.

Comparing NSNumber Objects

- `compare:` (page 1064)
  Returns an `NSComparisonResult` value that indicates whether the receiver is greater than, equal to, or less than a given number.
- `isEqualToNumber:` (page 1073)
  Returns a Boolean value that indicates whether the receiver and a given number are equal.

Accessing Type Information

- `objCType` (page 1075)
  Returns a C string containing the Objective-C type of the data contained in the receiver.

Class Methods

numberWithBool:

Creates and returns an `NSNumber` object containing a given value, treating it as a `BOOL`.

```
+ (NSNumber *)numberWithBool:(BOOL)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as a `BOOL`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

EnhancedAudioBurn

GridCalendar

Quartz Composer WWDC 2005 TextEdit

SMARTQuery

TextEditPlus

Declared In

`NSValue.h`

numberWithChar:

Creates and returns an `NSNumber` object containing a given value, treating it as a signed `char`.

```
+ (NSNumber *)numberWithChar:(char)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as a signed `char`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSValue.h`

numberWithDouble:

Creates and returns an `NSNumber` object containing a given value, treating it as a `double`.

```
+ (NSNumber *)numberWithDouble:(double)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as a `double`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CIAnnotation

CocoaSOAP

SimpleScriptingObjects

TemperatureTester

TrackBall

Declared In

`NSValue.h`

numberWithFloat:

Creates and returns an `NSNumber` object containing a given value, treating it as a `float`.

```
+ (NSNumber *)numberWithFloat:(float)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as a `float`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CIAnnotation

Quartz Composer WWDC 2005 TextEdit

SampleScannerApp

SpeedometerView

TextEditPlus

Declared In

`NSValue.h`

numberWithInt:

Creates and returns an `NSNumber` object containing a given value, treating it as a signed `int`.

```
+ (NSNumber *)numberWithInt:(int)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as a signed `int`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Dicey

QTCoreVideo301

Quartz Composer WWDC 2005 TextEdit

StickiesExample

TextEditPlus

Declared In

`NSValue.h`

numberWithInteger:

Creates and returns an `NSNumber` object containing a given value, treating it as an `NSInteger`.

```
+ (NSNumber *)numberWithInteger:(NSInteger)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as an `NSInteger`.

Availability

Available in Mac OS X v10.5 and later.

Related Sample Code

AutomatorHandsOn

Core Data HTML Store

CustomAtomicStoreSubclass

Mountains

Declared In

`NSValue.h`

numberWithLong:

Creates and returns an `NSNumber` object containing a given value, treating it as a signed `long`.

```
+ (NSNumber *)numberWithLong:(long)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as a signed `long`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AttachAScript

CocoaSpeechSynthesisExample

QTAudioExtractionPanel

QTKitPlayer

QTMetadataEditor

Declared In

`NSValue.h`

numberWithLongLong:

Creates and returns an `NSNumber` object containing a given value, treating it as a signed `long long`.

```
+ (NSNumber *)numberWithLongLong:(long long)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as a signed `long long`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTKitMovieShuffler

Declared In

`NSValue.h`

numberWithShort:

Creates and returns an `NSNumber` object containing *value*, treating it as a signed `short`.

```
+ (NSNumber *)numberWithShort:(short)value
```

Parameters

value

The value for the new number.

Return Value
An `NSNumber` object containing *`value`*, treating it as a signed `short`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CocoaSpeechSynthesisExample
Core Data HTML Store
CoreRecipes
FunkyOverlayWindow
SampleScannerApp

Declared In
`NSValue.h`

numberWithUnsignedChar:

Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned char`.

```
+ (NSNumber *)numberWithUnsignedChar:(unsigned char)value
```

Parameters

`value`
The value for the new number.

Return Value
An `NSNumber` object containing *`value`*, treating it as an `unsigned char`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSValue.h`

numberWithUnsignedInt:

Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned int`.

```
+ (NSNumber *)numberWithUnsignedInt:(unsigned int)value
```

Parameters

`value`
The value for the new number.

Return Value
An `NSNumber` object containing *`value`*, treating it as an `unsigned int`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
EnhancedAudioBurn

OpenGLCaptureToMovie
Quartz Composer QCTV
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
NSValue.h

numberWithUnsignedInteger:

Creates and returns an NSNumber object containing a given value, treating it as an NSUInteger.

```
+ (NSNumber *)numberWithUnsignedInteger:(NSUInteger)value
```

Parameters

value

The value for the new number.

Return Value
An NSNumber object containing *value*, treating it as an NSUInteger.

Availability
Available in Mac OS X v10.5 and later.

Declared In
NSValue.h

numberWithUnsignedLong:

Creates and returns an NSNumber object containing a given value, treating it as an unsigned long.

```
+ (NSNumber *)numberWithUnsignedLong:(unsigned long)value
```

Parameters

value

The value for the new number.

Return Value
An NSNumber object containing *value*, treating it as an unsigned long.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Apply Firmware Password
QTMetadataEditor
QTRecorder
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
NSValue.h

numberWithUnsignedLongLong:

Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned long long`.

```
+ (NSNumber *)numberWithUnsignedLongLong:(unsigned long long)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as an `unsigned long long`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

EnhancedAudioBurn

Declared In

`NSValue.h`

numberWithUnsignedShort:

Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned short`.

```
+ (NSNumber *)numberWithUnsignedShort:(unsigned short)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as an `unsigned short`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AudioBurn

EnhancedDataBurn

QTMetadataEditor

Verification

Declared In

`NSValue.h`

Instance Methods

boolValue

Returns the receiver's value as a `BOOL`.

```
- (BOOL)boolValue
```

Return Value
The receiver's value as a `BOOL`, converting it as necessary.

Special Considerations

Prior to Mac OS X v10.3, the value returned isn't guaranteed to be one of `YES` or `NO`. A `0` value always means `NO` or false, but any nonzero value should be interpreted as `YES` or true.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CoreRecipes

Declared In
`NSValue.h`

charValue

Returns the receiver's value as a `char`.

```
- (char)charValue
```

Return Value
The receiver's value as a `char`, converting it as necessary.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSValue.h`

compare:

Returns an `NSComparisonResult` value that indicates whether the receiver is greater than, equal to, or less than a given number.

```
- (NSComparisonResult)compare:(NSNumber *)aNumber
```

Parameters

aNumber

The number with which to compare the receiver.

This value must not be `nil`. If the value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

Return Value

`NSOrderedAscending` if the value of *aNumber* is greater than the receiver's, `NSOrderedSame` if they're equal, and `NSOrderedDescending` if the value of *aNumber* is less than the receiver's.

Discussion

The `compare:` method follows the standard C rules for type conversion. For example, if you compare an `NSNumber` object that has an integer value with an `NSNumber` object that has a floating point value, the integer value is converted to a floating-point value for comparison.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSValue.h`

decimalValue

Returns the receiver's value, expressed as an `NSDecimal` structure.

```
- (NSDecimal)decimalValue
```

Return Value

The receiver's value, expressed as an `NSDecimal` structure. The value returned isn't guaranteed to be exact for `float` and `double` values.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSDecimalNumber.h`

descriptionWithLocale:

Returns a string that represents the contents of the receiver for a given locale.

```
- (NSString *)descriptionWithLocale:(id)aLocale
```

Parameters

aLocale

An object containing locale information with which to format the description. Use `nil` if you don't want the description formatted.

Return Value

A string that represents the contents of the receiver formatted using the locale information in *locale*.

Discussion
For example, if you have an `NSNumber` object that has the integer value 522, sending it the `descriptionWithLocale:` message returns the string "522".

To obtain the string representation, this method invokes `NSString`'s `initWithFormat:locale:` (page 1574) method, supplying the format based on the type the `NSNumber` object was created with:

Data Type	Format Specification
`char`	%i
`double`	%0.16g
`float`	%0.7g
`int`	%i
`long`	%li
`long long`	%lli
`short`	%hi
`unsigned char`	%u
`unsigned int`	%u
`unsigned long`	%lu
`unsigned long long`	%llu
`unsigned short`	%hu

Availability
Available in Mac OS X v10.0 and later.

See Also
`- stringValue` (page 1075)

Declared In
`NSValue.h`

doubleValue

Returns the receiver's value as a `double`.

```
- (double)doubleValue
```

Return Value
The receiver's value as a `double`, converting it as necessary.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CocoaSOAP
QTKitMovieShuffler
Quartz Composer QCTV
SimpleScriptingObjects
SimpleScriptingProperties

Declared In
`NSValue.h`

floatValue

Returns the receiver's value as a `float`.

```
- (float)floatValue
```

Return Value
The receiver's value as a `float`, converting it as necessary.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CIAnnotation
MyPhoto
Quartz Composer WWDC 2005 TextEdit
TextEditPlus
WebKitPluginWithJavaScript

Declared In
`NSValue.h`

initWithBool:

Returns an `NSNumber` object initialized to contain a given value, treated as a `BOOL`.

```
- (id)initWithBool:(BOOL)value
```

Parameters
value
The value for the new number.

Return Value
An `NSNumber` object containing *value*, treating it as a `BOOL`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSValue.h`

initWithChar:

Returns an `NSNumber` object initialized to contain a given value, treated as a signed `char`.

```
- (id)initWithChar:(char)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as a signed `char`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSValue.h`

initWithDouble:

Returns an `NSNumber` object initialized to contain *value*, treated as a `double`.

```
- (id)initWithDouble:(double)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as a `double`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSValue.h`

initWithFloat:

Returns an `NSNumber` object initialized to contain a given value, treated as a `float`.

```
- (id)initWithFloat:(float)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as a `float`.

Availability

Available in Mac OS X v10.0 and later.

Declared In
NSValue.h

initWithInt:

Returns an NSNumber object initialized to contain a given value, treated as a signed int.

```
- (id)initWithInt:(int)value
```

Parameters

value

The value for the new number.

Return Value
An NSNumber object containing *value*, treating it as a signed int.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSValue.h

initWithInteger:

Returns an NSNumber object initialized to contain a given value, treated as an NSInteger.

```
- (id)initWithInteger:(NSInteger)value
```

Parameters

value

The value for the new number.

Return Value
An NSNumber object containing *value*, treating it as an NSInteger.

Availability
Available in Mac OS X v10.5 and later.

Declared In
NSValue.h

initWithLong:

Returns an NSNumber object initialized to contain a given value, treated as a signed long.

```
- (id)initWithLong:(long)value
```

Parameters

value

The value for the new number.

Return Value
An NSNumber object containing *value*, treating it as a signed long.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSValue.h`

initWithLongLong:

Returns an `NSNumber` object initialized to contain *`value`*, treated as a signed `long long`.

```
- (id)initWithLongLong:(long long)value
```

Parameters

`value`
The value for the new number.

Return Value
An `NSNumber` object containing *`value`*, treating it as a signed `long long`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSValue.h`

initWithShort:

Returns an `NSNumber` object initialized to contain a given value, treated as a signed `short`.

```
- (id)initWithShort:(short)value
```

Parameters

`value`
The value for the new number.

Return Value
An `NSNumber` object containing *`value`*, treating it as a signed `short`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSValue.h`

initWithUnsignedChar:

Returns an `NSNumber` object initialized to contain a given value, treated as an `unsigned char`.

```
- (id)initWithUnsignedChar:(unsigned char)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as an `unsigned char`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSValue.h`

initWithUnsignedInt:

Returns an `NSNumber` object initialized to contain a given value, treated as an `unsigned int`.

```
- (id)initWithUnsignedInt:(unsigned int)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as an `unsigned int`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSValue.h`

initWithUnsignedInteger:

Returns an `NSNumber` object initialized to contain a given value, treated as an `NSUInteger`.

```
- (id)initWithUnsignedInteger:(NSUInteger)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as an `NSUInteger`.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSValue.h`

initWithUnsignedLong:

Returns an `NSNumber` object initialized to contain a given value, treated as an `unsigned long`.

```
- (id)initWithUnsignedLong:(unsigned long)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as an `unsigned long`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSValue.h`

initWithUnsignedLongLong:

Returns an `NSNumber` object initialized to contain a given value, treated as an `unsigned long long`.

```
- (id)initWithUnsignedLongLong:(unsigned long long)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as an `unsigned long long`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSValue.h`

initWithUnsignedShort:

Returns an `NSNumber` object initialized to contain a given value, treated as an `unsigned short`.

```
- (id)initWithUnsignedShort:(unsigned short)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as an `unsigned short`.

Availability

Available in Mac OS X v10.0 and later.

Declared In
NSValue.h

integerValue

Returns the receiver's value as an NSInteger.

```
- (NSInteger)integerValue
```

Return Value
The receiver's value as an NSInteger, converting it as necessary.

Availability
Available in Mac OS X v10.5 and later.

Related Sample Code
Mountains
QTCoreVideo301

Declared In
NSValue.h

intValue

Returns the receiver's value as an int.

```
- (int)intValue
```

Return Value
The receiver's value as an int, converting it as necessary.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ABPresence
Dicey
EnhancedAudioBurn
QTCoreVideo301
WebKitPluginWithJavaScript

Declared In
NSValue.h

isEqualToNumber:

Returns a Boolean value that indicates whether the receiver and a given number are equal.

```
- (BOOL)isEqualToNumber:(NSNumber *)aNumber
```

Parameters

aNumber

The number with which to compare the receiver.

Return Value

YES if the receiver and *aNumber* are equal, otherwise NOr

Discussion

Two NSNumber objects are considered equal if they have the same id values or if they have equivalent values (as determined by the compare: (page 1064) method).

This method is more efficient than compare: (page 1064) if you know the two objects are numbers.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSValue.h

longLongValue

Returns the receiver's value as a long long.

```
- (long long)longLongValue
```

Return Value

The receiver's value as a long long, converting it as necessary.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSValue.h

longValue

Returns the receiver's value as a long.

```
- (long)longValue
```

Return Value

The receiver's value as a long, converting it as necessary.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CocoaSpeechSynthesisExample
CustomAtomicStoreSubclass
QTKitMovieShuffler
Sketch-112
WhackedTV

Declared In
NSValue.h

objCType

Returns a C string containing the Objective-C type of the data contained in the receiver.

```
- (const char *)objCType
```

Return Value
A C string containing the Objective-C type of the data contained in the receiver, as encoded by the `@encode()` compiler directive.

Special Considerations
The returned type does not necessarily match the method the receiver was created with.

shortValue

Returns the receiver's value as a `short`.

```
- (short)shortValue
```

Return Value
The receiver's value as a `short`, converting it as necessary.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CocoaSpeechSynthesisExample
CoreRecipes

Declared In
NSValue.h

stringValue

Returns the receiver's value as a human-readable string.

```
- (NSString *)stringValue
```

Return Value
The receiver's value as a human-readable string, created by invoking `descriptionWithLocale:` (page 1065) where locale is `nil`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AlbumToSlideshow
Audio Unit Effect Templates

Calculator
VideoHardwareInfo

Declared In
`NSValue.h`

unsignedCharValue

Returns the receiver's value as an unsigned `char`.

```
- (unsigned char)unsignedCharValue
```

Return Value
The receiver's value as an unsigned `char`, converting it as necessary.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSValue.h`

unsignedIntegerValue

Returns the receiver's value as an `NSUInteger`.

```
- (NSUInteger)unsignedIntegerValue
```

Return Value
The receiver's value as an `NSUInteger`, converting it as necessary.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSValue.h`

unsignedIntValue

Returns the receiver's value as an unsigned `int`.

```
- (unsigned int)unsignedIntValue
```

Return Value
The receiver's value as an unsigned `int`, converting it as necessary.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Dicey
OpenGLCaptureToMovie
Quartz Composer WWDC 2005 TextEdit

TextEditPlus
WhackedTV

Declared In
NSValue.h

unsignedLongLongValue

Returns the receiver's value as an unsigned `long long`.

```
- (unsigned long long)unsignedLongLongValue
```

Return Value
The receiver's value as an unsigned `long long`, converting it as necessary.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSValue.h

unsignedLongValue

Returns the receiver's value as an unsigned `long`.

```
- (unsigned long)unsignedLongValue
```

Return Value
The receiver's value as an unsigned `long`, converting it as necessary.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTRecorder
Quartz Composer WWDC 2005 TextEdit
SampleScannerApp
TextEditPlus

Declared In
NSValue.h

unsignedShortValue

Returns the receiver's value as an unsigned `short`.

```
- (unsigned short)unsignedShortValue
```

Return Value
The receiver's value as an unsigned `short`, converting it as necessary.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSValue.h`

NSNumberFormatter Class Reference

Inherits from	NSFormatter : NSObject
Conforms to	NSCoding (NSFormatter) NSCopying (NSFormatter) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSNumberFormatter.h
Companion guide	Data Formatting Programming Guide for Cocoa
Related sample code	CoreRecipes Grady Mountains NumberInput_IMKit_Sample Quartz Composer QCTV

Overview

Instances of `NSNumberFormatter` format the textual representation of cells that contain `NSNumber` objects and convert textual representations of numeric values into `NSNumber` objects. The representation encompasses integers, floats, and doubles; floats and doubles can be formatted to a specified decimal position. `NSNumberFormatter` objects can also impose ranges on the numeric values cells can accept.

Many new methods were added to `NSNumberFormatter` for Mac OS X v10.4 with the intent of making the class interface more like that of `CFNumberFormatter`, the Core Foundation service on which the class is based. The behavior of an `NSNumberFormatter` object can conform either to the range of behaviors existing prior to Mac OS X v10.4 or to the range of behavior since that release. (Methods added for and since Mac OS X v10.4 are indicated by a method's availability statement.) You can determine the current formatter behavior with the `formatterBehavior` (page 1093) method and you can set the formatter behavior with the `setFormatterBehavior:` (page 1115) method.

iPhone OS Note: iPhone OS supports only the modern 10.4+ behavior. 10.0-style methods and format strings are not available on iPhone OS.

Important: The pre-Mac OS X v10.4 methods of `NSNumberFormatter` are not compatible with the methods added for Mac OS X v10.4. An `NSNumberFormatter` object should not invoke methods in these different behavior groups indiscriminately. Use the old-style methods if you have configured the number-formatter behavior to be `NSNumberFormatterBehavior10_0`. Use the new methods instead of the older-style ones if you have configured the number-formatter behavior to be `NSNumberFormatterBehavior10_4`.

Nomenclature note: `NSNumberFormatter` provides several methods (such as `setMaximumFractionDigits:` (page 1120)) that allow you to manage the number of **fraction digits** allowed as input by an instance: "fraction digits" are the numbers after the decimal separator (in English locales typically referred to as the "decimal point").

Tasks

Configuring Formatter Behavior and Style

- `setFormatterBehavior:` (page 1115)
  Sets the formatter behavior of the receiver.
- `formatterBehavior` (page 1093)
  Returns an `NSNumberFormatterBehavior` constant that indicates the formatter behavior of the receiver.
+ `setDefaultFormatterBehavior:` (page 1088)
  Sets the default formatter behavior for new instances of `NSNumberFormatter`.
+ `defaultFormatterBehavior` (page 1087)
  Returns an `NSNumberFormatterBehavior` constant that indicates default formatter behavior for new instances of `NSNumberFormatter`.
- `setNumberStyle:` (page 1126)
  Sets the number style used by the receiver.
- `numberStyle` (page 1105)
  Returns the number-formatter style of the receiver.
- `setGeneratesDecimalNumbers:` (page 1116)
  Controls whether the receiver creates instances of `NSDecimalNumber` when it converts strings to number objects.
- `generatesDecimalNumbers` (page 1094)
  Returns a Boolean value that indicates whether the receiver creates instances of `NSDecimalNumber` when it converts strings to number objects.

Converting Between Numbers and Strings

- `getObjectValue:forString:range:error:` (page 1094)
  Returns by reference a cell-content object after creating it from a range of characters in a given string.

- `numberFromString:` (page 1104)
  Returns an `NSNumber` object created by parsing a given string.
- `stringFromNumber:` (page 1136)
  Returns a string containing the formatted value of the provided number object.

Managing Localization of Numbers

- `setLocalizesFormat:` (page 1119)
  Sets whether the dollar sign character (`$`), decimal separator character (`.`), and thousand separator character (`,`) are converted to appropriately localized characters as specified by the user's localization preference.
- `localizesFormat` (page 1098)
  Returns a Boolean value that indicates whether the receiver localizes formats.
- `setLocale:` (page 1118)
  Sets the locale of the receiver.
- `locale` (page 1097)
  Returns the locale of the receiver.

Configuring Rounding Behavior

- `setRoundingBehavior:` (page 1130)
  Sets the rounding behavior used by the receiver.
- `roundingBehavior` (page 1108)
  Returns an `NSDecimalNumberHandler` object indicating the rounding behavior of the receiver.
- `setRoundingIncrement:` (page 1131)
  Sets the rounding increment used by the receiver.
- `roundingIncrement` (page 1108)
  Returns the rounding increment used by the receiver.
- `setRoundingMode:` (page 1131)
  Sets the rounding mode used by the receiver.
- `roundingMode` (page 1109)
  Returns the rounding mode used by the receiver.

Configuring Numeric Formats

- `setFormat:` (page 1114)
  Sets the receiver's format.
- `formatWidth` (page 1094)
  Returns the format width of the receiver.
- `setNegativeFormat:` (page 1124)
  Sets the format the receiver uses to display negative values.
- `negativeFormat` (page 1102)
  Returns the format used by the receiver to display negative numbers.

- `setPositiveFormat:` (page 1129)
  Sets the format the receiver uses to display positive values.
- `positiveFormat` (page 1107)
  Returns the format used by the receiver to display positive numbers.
- `setFormatWidth:` (page 1115)
  Sets the format width used by the receiver.
- `format` (page 1093)
  Returns the format used by the receiver.
- `setMultiplier:` (page 1123)
  Sets the multiplier of the receiver.
- `multiplier` (page 1102)
  Returns the multiplier used by the receiver as an `NSNumber` object.

Configuring Numeric Symbols

- `percentSymbol` (page 1106)
  Returns the string that the receiver uses to represent the percent symbol.
- `setPercentSymbol:` (page 1128)
  Sets the string used by the receiver to represent the percent symbol.
- `perMillSymbol` (page 1106)
  Returns the string that the receiver uses for the per-thousands symbol.
- `setPerMillSymbol:` (page 1128)
  Sets the string used by the receiver to represent the per-mill (per-thousand) symbol.
- `minusSign` (page 1101)
  Returns the string the receiver uses to represent the minus sign.
- `setMinusSign:` (page 1123)
  Sets the string used by the receiver for the minus sign.
- `plusSign` (page 1106)
  Returns the string the receiver uses for the plus sign.
- `setPlusSign:` (page 1128)
  Sets the string used by the receiver to represent the plus sign.
- `exponentSymbol` (page 1092)
  Returns the string the receiver uses as an exponent symbol.
- `setExponentSymbol:` (page 1114)
  Sets the string used by the receiver to represent the exponent symbol.
- `zeroSymbol` (page 1141)
  Returns the string the receiver uses as the symbol to show the value zero.
- `setZeroSymbol:` (page 1136)
  Sets the string the receiver uses as the symbol to show the value zero.
- `nilSymbol` (page 1104)
  Returns the string the receiver uses to represent a `nil` value.
- `setNilSymbol:` (page 1125)
  Sets the string the receiver uses to represent `nil` values.

- `notANumberSymbol` (page 1104)
  Returns the symbol the receiver uses to represent NaN (“not a number”) when it converts values.
- `setNotANumberSymbol:` (page 1126)
  Sets the string the receiver uses to represent NaN (“not a number”).
- `negativeInfinitySymbol` (page 1102)
  Returns the symbol the receiver uses to represent negative infinity.
- `setNegativeInfinitySymbol:` (page 1124)
  Sets the string used by the receiver for the negative infinity symbol.
- `positiveInfinitySymbol` (page 1107)
  Returns the string the receiver uses for the positive infinity symbol.
- `setPositiveInfinitySymbol:` (page 1129)
  Sets the string used by the receiver for the positive infinity symbol.

Configuring the Format of Currency

- `setCurrencySymbol:` (page 1113)
  Sets the string used by the receiver as a local currency symbol.
- `currencySymbol` (page 1092)
  Returns the receiver’s local currency symbol.
- `setCurrencyCode:` (page 1112)
  Sets the receiver’s currency code.
- `currencyCode` (page 1091)
  Returns the receiver’s currency code as a string.
- `setInternationalCurrencySymbol:` (page 1118)
  Sets the string used by the receiver for the international currency symbol.
- `internationalCurrencySymbol` (page 1096)
  Returns the international currency symbol used by the receiver.
- `setCurrencyGroupingSeparator:` (page 1112)
  Sets the currency grouping separator for the receiver.
- `currencyGroupingSeparator` (page 1091)
  Returns the currency grouping separator for the receiver.

Configuring Numeric Prefixes and Suffixes

- `setPositivePrefix:` (page 1129)
  Sets the string the receiver uses as the prefix for positive values.
- `positivePrefix` (page 1107)
  Returns the string the receiver uses as the prefix for positive values.
- `setPositiveSuffix:` (page 1130)
  Sets the string the receiver uses as the suffix for positive values.
- `positiveSuffix` (page 1108)
  Returns the string the receiver uses as the suffix for positive values.

- `setNegativePrefix:` (page 1125)
  Sets the string the receiver uses as a prefix for negative values.
- `negativePrefix` (page 1103)
  Returns the string the receiver inserts as a prefix to negative values.
- `setNegativeSuffix:` (page 1125)
  Sets the string the receiver uses as a suffix for negative values.
- `negativeSuffix` (page 1103)
  Returns the string the receiver adds as a suffix to negative values.

Configuring the Display of Numeric Values

- `setTextAttributesForNegativeValues:` (page 1132)
  Sets the text attributes to be used in displaying negative values .
- `textAttributesForNegativeValues` (page 1137)
  Returns a dictionary containing the text attributes that have been set for negative values.
- `setTextAttributesForPositiveValues:` (page 1134)
  Sets the text attributes to be used in displaying positive values.
- `textAttributesForPositiveValues` (page 1139)
  Returns a dictionary containing the text attributes that have been set for positive values.
- `setAttributedStringForZero:` (page 1111)
  Sets the attributed string that the receiver uses to display zero values.
- `attributedStringForZero` (page 1090)
  Returns the attributed string used to display zero values.
- `setTextAttributesForZero:` (page 1134)
  Sets the text attributes used to display a zero value.
- `textAttributesForZero` (page 1139)
  Returns a dictionary containing the text attributes used to display a value of zero.
- `setAttributedStringForNil:` (page 1110)
  Sets the attributed string the receiver uses to display `nil` values.
- `attributedStringForNil` (page 1089)
  Returns the attributed string used to display `nil` values.
- `setTextAttributesForNil:` (page 1133)
  Sets the text attributes used to display the `nil` symbol.
- `textAttributesForNil` (page 1138)
  Returns a dictionary containing the text attributes used to display the `nil` symbol.
- `setAttributedStringForNotANumber:` (page 1111)
  Sets the attributed string the receiver uses to display “not a number” values.
- `attributedStringForNotANumber` (page 1090)
  Returns the attributed string used to display “not a number” values.
- `setTextAttributesForNotANumber:` (page 1133)
  Sets the text attributes used to display the NaN ("not a number") string.
- `textAttributesForNotANumber` (page 1138)
  Returns a dictionary containing the text attributes used to display the NaN ("not a number") symbol.

- `setTextAttributesForPositiveInfinity:` (page 1134)
  Sets the text attributes used to display the positive infinity symbol.
- `textAttributesForPositiveInfinity` (page 1138)
  Returns a dictionary containing the text attributes used to display the positive infinity symbol.
- `setTextAttributesForNegativeInfinity:` (page 1132)
  Sets the text attributes used to display the negative infinity symbol.
- `textAttributesForNegativeInfinity` (page 1137)
  Returns a dictionary containing the text attributes used to display the negative infinity string.

Configuring Separators and Grouping Size

- `setGroupingSeparator:` (page 1116)
  Specifies the string used by the receiver for a grouping separator.
- `groupingSeparator` (page 1095)
  Returns a string containing the receiver's grouping separator.
- `setUsesGroupingSeparator:` (page 1135)
  Controls whether the receiver displays the grouping separator.
- `usesGroupingSeparator` (page 1140)
  Returns a Boolean value that indicates whether the receiver uses the grouping separator.
- `setThousandSeparator:` (page 1135)
  Sets the character the receiver uses as a thousand separator.
- `thousandSeparator` (page 1139)
  Returns a string containing the character the receiver uses to represent thousand separators.
- `setHasThousandSeparators:` (page 1117)
  Sets whether the receiver uses thousand separators.
- `hasThousandSeparators` (page 1096)
  Returns a Boolean value that indicates whether the receiver's format includes thousand separators.
- `setDecimalSeparator:` (page 1113)
  Sets the character the receiver uses as a decimal separator.
- `decimalSeparator` (page 1092)
  Returns a string containing the character the receiver uses to represent decimal separators.
- `setAlwaysShowsDecimalSeparator:` (page 1110)
  Controls whether the receiver always shows the decimal separator, even for integer numbers.
- `alwaysShowsDecimalSeparator` (page 1089)
  Returns a Boolean value that indicates whether the receiver always shows a decimal separator, even if the number is an integer.
- `setCurrencyDecimalSeparator:` (page 1112)
  Sets the string used by the receiver as a decimal separator.
- `currencyDecimalSeparator` (page 1091)
  Returns the receiver's currency decimal separator as a string.
- `setGroupingSize:` (page 1117)
  Sets the grouping size of the receiver.
- `groupingSize` (page 1095)
  Returns the receiver's primary grouping size.

- `setSecondaryGroupingSize:` (page 1131)
  Sets the secondary grouping size of the receiver.
- `secondaryGroupingSize` (page 1109)
  Returns the size of secondary groupings for the receiver.

Managing the Padding of Numbers

- `setPaddingCharacter:` (page 1127)
  Sets the string that the receiver uses to pad numbers in the formatted string representation.
- `paddingCharacter` (page 1105)
  Returns a string containing the padding character for the receiver.
- `setPaddingPosition:` (page 1127)
  Sets the padding position used by the receiver.
- `paddingPosition` (page 1105)
  Returns the padding position of the receiver.

Managing Input Attributes

- `setAllowsFloats:` (page 1110)
  Sets whether the receiver allows as input floating-point values (that is, values that include the period character [.]).
- `allowsFloats` (page 1088)
  Returns a Boolean value that indicates whether the receiver allows floating-point values as input.
- `setMinimum:` (page 1121)
  Sets the lowest number the receiver allows as input.
- `minimum` (page 1100)
  Returns the lowest number allowed as input by the receiver.
- `setMaximum:` (page 1119)
  Sets the highest number the receiver allows as input.
- `maximum` (page 1098)
  Returns the highest number allowed as input by the receiver.
- `setMinimumIntegerDigits:` (page 1122)
  Sets the minimum number of integer digits allowed as input by the receiver.
- `minimumIntegerDigits` (page 1101)
  Returns the minimum number of integer digits allowed as input by the receiver.
- `setMinimumFractionDigits:` (page 1122)
  Sets the minimum number of digits after the decimal separator allowed as input by the receiver.
- `minimumFractionDigits` (page 1100)
  Returns the minimum number of digits after the decimal separator allowed as input by the receiver.
- `setMaximumIntegerDigits:` (page 1120)
  Sets the maximum number of integer digits allowed as input by the receiver.
- `maximumIntegerDigits` (page 1099)
  Returns the maximum number of integer digits allowed as input by the receiver.

- `setMaximumFractionDigits:` (page 1120)
  Sets the maximum number of digits after the decimal separator allowed as input by the receiver.
- `maximumFractionDigits` (page 1099)
  Returns the maximum number of digits after the decimal separator allowed as input by the receiver.

Configuring Significant Digits

- `setUsesSignificantDigits:` (page 1136)
  Sets whether the receiver uses significant digits.
- `usesSignificantDigits` (page 1140)
  Returns a Boolean value that indicates whether the receiver uses significant digits.
- `setMinimumSignificantDigits:` (page 1123)
  Sets the minimum number of significant digits for the receiver.
- `minimumSignificantDigits` (page 1101)
  Returns the minimum number of significant digits for the receiver.
- `setMaximumSignificantDigits:` (page 1121)
  Sets the maximum number of significant digits for the receiver.
- `maximumSignificantDigits` (page 1099)
  Returns the maximum number of significant digits for the receiver.

Managing Leniency Behavior

- `setLenient:` (page 1118)
  Sets whether the receiver will use heuristics to guess at the number which is intended by a string.
- `isLenient` (page 1097)
  Returns a Boolean value that indicates whether the receiver uses heuristics to guess at the number which is intended by a string.

Managing the Validation of Partial Numeric Strings

- `setPartialStringValidationEnabled:` (page 1127)
  Sets whether partial string validation is enabled for the receiver.
- `isPartialStringValidationEnabled` (page 1097)
  Returns a Boolean value that indicates whether partial string validation is enabled.

Class Methods

defaultFormatterBehavior

Returns an `NSNumberFormatterBehavior` constant that indicates default formatter behavior for new instances of `NSNumberFormatter`.

```
+ (NSNumberFormatterBehavior)defaultFormatterBehavior
```

Return Value
An `NSNumberFormatterBehavior` constant that indicates default formatter behavior for new instances of `NSNumberFormatter`.

Availability
Available in Mac OS X v10.4 and later.

See Also
`+ setDefaultFormatterBehavior:` (page 1088)

Declared In
`NSNumberFormatter.h`

setDefaultFormatterBehavior:

Sets the default formatter behavior for new instances of `NSNumberFormatter`.

```
+ (void)setDefaultFormatterBehavior:(NSNumberFormatterBehavior)behavior
```

Parameters
behavior
An `NSNumberFormatterBehavior` constant that indicates the revision of the class providing the default behavior.

Availability
Available in Mac OS X v10.4 and later.

See Also
`+ defaultFormatterBehavior` (page 1087)

Related Sample Code
NumberInput_IMKit_Sample

Declared In
`NSNumberFormatter.h`

Instance Methods

allowsFloats

Returns a Boolean value that indicates whether the receiver allows floating-point values as input.

```
- (BOOL)allowsFloats
```

Return Value
`YES` if the receiver allows as input floating-point values (that is, values that include the period character [`.`]), otherwise `NO`.

Discussion
When this method returns `NO`, only integer values can be provided as input. The default is `YES`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setAllowsFloats:` (page 1110)

Declared In
`NSNumberFormatter.h`

alwaysShowsDecimalSeparator

Returns a Boolean value that indicates whether the receiver always shows a decimal separator, even if the number is an integer.

```
- (BOOL)alwaysShowsDecimalSeparator
```

Return Value
`YES` if the receiver always shows a decimal separator, even if the number is an integer, otherwise `NO`.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setAlwaysShowsDecimalSeparator:` (page 1110)

Declared In
`NSNumberFormatter.h`

attributedStringForNil

Returns the attributed string used to display `nil` values.

```
- (NSAttributedString *)attributedStringForNil
```

Return Value
The attributed string used to display `nil` values.

Discussion
By default `nil` values are displayed as an empty string.

Special Considerations

This method is for use with formatters using `NSNumberFormatterBehavior10_0` behavior.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setAttributedStringForNil:` (page 1110)

Declared In
NSNumberFormatter.h

attributedStringForNotANumber

Returns the attributed string used to display “not a number” values.

```
- (NSAttributedString *)attributedStringForNotANumber
```

Return Value
The attributed string used to display “not a number” values.

Discussion
By default “not a number” values are displayed as the string “NaN”.

Special Considerations

This method is for use with formatters using `NSNumberFormatterBehavior10_0` behavior.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setAttributedStringForNotANumber:` (page 1111)

Declared In
NSNumberFormatter.h

attributedStringForZero

Returns the attributed string used to display zero values.

```
- (NSAttributedString *)attributedStringForZero
```

Return Value
The attributed string used to display zero values.

Discussion
By default zero values are displayed according to the format specified for positive values; for more discussion of this subject see *Data Formatting Programming Guide for Cocoa*.

Special Considerations

This method is for use with formatters using `NSNumberFormatterBehavior10_0` behavior.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setAttributedStringForZero:` (page 1111)

Declared In
NSNumberFormatter.h

currencyCode

Returns the receiver's currency code as a string.

```
- (NSString *)currencyCode
```

Return Value
The receiver's currency code as a string.

Discussion
A currency code is a three-letter code that is, in most cases, composed of a country's two-character Internet country code plus an extra character to denote the currency unit. For example, the currency code for the Australian dollar is "AUD". Currency codes are based on the ISO 4217 standard.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setCurrencyCode:` (page 1112)

Declared In
`NSNumberFormatter.h`

currencyDecimalSeparator

Returns the receiver's currency decimal separator as a string.

```
- (NSString *)currencyDecimalSeparator
```

Return Value
The receiver's currency decimal separator as a string.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `currencyDecimalSeparator` (page 1091)

Declared In
`NSNumberFormatter.h`

currencyGroupingSeparator

Returns the currency grouping separator for the receiver.

```
- (NSString *)currencyGroupingSeparator
```

Return Value
The currency grouping separator for the receiver.

Availability
Available in Mac OS X v10.5 and later.

Declared In
NSNumberFormatter.h

currencySymbol

Returns the receiver's local currency symbol.

```
- (NSString *)currencySymbol
```

Discussion
A country typically has a local currency symbol and an international currency symbol. The local symbol is used within the country, while the international currency symbol is used in international contexts to specify that country's currency unambiguously. The local currency symbol is often represented by a Unicode code point.

Availability
Available in Mac OS X v10.4 and later.

See Also
- internationalCurrencySymbol (page 1096)
- setCurrencySymbol: (page 1113)

Declared In
NSNumberFormatter.h

decimalSeparator

Returns a string containing the character the receiver uses to represent decimal separators.

```
- (NSString *)decimalSeparator
```

Return Value
A string containing the character the receiver uses to represent decimal separators.

Discussion
The return value doesn't indicate whether decimal separators are enabled.

Availability
Available in Mac OS X v10.0 and later.

See Also
- setDecimalSeparator: (page 1113)

Declared In
NSNumberFormatter.h

exponentSymbol

Returns the string the receiver uses as an exponent symbol.

```
- (NSString *)exponentSymbol
```

Return Value
The string the receiver uses as an exponent symbol.

Discussion
The exponent symbol is the “E” or “e” in the scientific notation of numbers, as in 1.0e+56.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setExponentSymbol:` (page 1114)

Declared In
`NSNumberFormatter.h`

format

Returns the format used by the receiver.

```
- (NSString *)format
```

Return Value
The format used by the receiver.

Special Considerations

This method is for use with formatters using `NSNumberFormatterBehavior10_0` behavior.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setFormat:` (page 1114)

Declared In
`NSNumberFormatter.h`

formatterBehavior

Returns an `NSNumberFormatterBehavior` constant that indicates the formatter behavior of the receiver.

```
- (NSNumberFormatterBehavior)formatterBehavior
```

Return Value
An `NSNumberFormatterBehavior` constant that indicates the formatter behavior of the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setFormatterBehavior:` (page 1115)

Declared In
`NSNumberFormatter.h`

formatWidth

Returns the format width of the receiver.

```
- (NSUInteger)formatWidth
```

Discussion
The format width is the number of characters of a formatted number within a string that is either left justified or right justified based on the value returned from `paddingPosition` (page 1105).

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setFormatWidth:` (page 1115)

Declared In
`NSNumberFormatter.h`

generatesDecimalNumbers

Returns a Boolean value that indicates whether the receiver creates instances of `NSDecimalNumber` when it converts strings to number objects.

```
- (BOOL)generatesDecimalNumbers
```

Return Value
`YES` if the receiver creates instances of `NSDecimalNumber` when it converts strings to number objects, `NO` if it creates instance of `NSNumber`.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setGeneratesDecimalNumbers:` (page 1116)

Declared In
`NSNumberFormatter.h`

getObjectValue:forString:range:error:

Returns by reference a cell-content object after creating it from a range of characters in a given string.

```
- (BOOL)getObjectValue:(out id *)anObject forString:(NSString *)aString range:(inout
    NSRange *)rangep error:(out NSError **)error
```

Parameters

anObject
On return, contains an instance of `NSDecimalNumber` or `NSNumber` based on the current value of `generatesDecimalNumbers` (page 1094). The default is to return `NSDecimalNumber` instances

aString
A string object with the range of characters specified in *rangep* that is used to create *anObject*.

rangep

A range of characters in *aString*. On return, contains the actual range of characters used to create the object.

error

If an error occurs, upon return contains an `NSError` object that explains the reason why the conversion failed. If you pass in `nil` for *error* you are indicating that you are not interested in error information.

Return Value
`YES` if the conversion from string to cell-content object was successful, otherwise `NO`.

Discussion
If there is an error, the delegate (if any) of the control object managing the cell can then respond to the failure in the `NSControl` delegation method `control:didFailToFormatString:errorDescription:`.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `numberFromString:` (page 1104)
- `stringFromNumber:` (page 1136)

Declared In
`NSNumberFormatter.h`

groupingSeparator

Returns a string containing the receiver's grouping separator.

```
- (NSString *)groupingSeparator
```

Return Value
A string containing the receiver's grouping separator.

Discussion
For example, the grouping separator used in the United States is the comma ("10,000") whereas in France it is the period ("10.000").

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setGroupingSeparator:` (page 1116)

Declared In
`NSNumberFormatter.h`

groupingSize

Returns the receiver's primary grouping size.

```
- (NSUInteger)groupingSize
```

Return Value
The receiver's primary grouping size.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setGroupingSize:` (page 1117)

Declared In
`NSNumberFormatter.h`

hasThousandSeparators

Returns a Boolean value that indicates whether the receiver's format includes thousand separators.

```
- (BOOL)hasThousandSeparators
```

Return Value
`YES` if the receiver's format includes thousand separators, otherwise `NO`.

Discussion
The default is `NO`.

Special Considerations

This method is for use with formatters using `NSNumberFormatterBehavior10_0` behavior.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `setHasThousandSeparators:` (page 1117)

Declared In
`NSNumberFormatter.h`

internationalCurrencySymbol

Returns the international currency symbol used by the receiver.

```
- (NSString *)internationalCurrencySymbol
```

Discussion
A country typically has a local currency symbol and an international currency symbol. The local symbol is used within the country, while the international currency symbol is used in international contexts to specify that country's currency unambiguously. The international currency symbol is often represented by a Unicode code point.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `currencySymbol` (page 1092)

- setInternationalCurrencySymbol: (page 1118)

Declared In
NSNumberFormatter.h

isLenient

Returns a Boolean value that indicates whether the receiver uses heuristics to guess at the number which is intended by a string.

```
- (BOOL)isLenient
```

Return Value
YES if the receiver uses heuristics to guess at the number which is intended by the string; otherwise NO.

Availability
Available in Mac OS X v10.5 and later.

See Also
- setLenient: (page 1118)

Declared In
NSNumberFormatter.h

isPartialStringValidationEnabled

Returns a Boolean value that indicates whether partial string validation is enabled.

```
- (BOOL)isPartialStringValidationEnabled
```

Return Value
YES if partial string validation is enabled, otherwise NO.

Availability
Available in Mac OS X v10.5 and later.

See Also
- setPartialStringValidationEnabled: (page 1127)

Declared In
NSNumberFormatter.h

locale

Returns the locale of the receiver.

```
- (NSLocale *)locale
```

Return Value
The locale of the receiver.

Discussion
A number formatter's locale specifies default localization attributes, such as ISO country and language codes, currency code, calendar, system of measurement, and decimal separator.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setLocale:` (page 1118)

Declared In
`NSNumberFormatter.h`

localizesFormat

Returns a Boolean value that indicates whether the receiver localizes formats.

```
- (BOOL)localizesFormat
```

Return Value
`YES` if the receiver localizes formats, otherwise `NO`.

Discussion
The default is `NO`.

Special Considerations

This method is for use with formatters using `NSNumberFormatterBehavior10_0` behavior.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `setLocalizesFormat:` (page 1119)

Declared In
`NSNumberFormatter.h`

maximum

Returns the highest number allowed as input by the receiver.

```
- (NSNumber *)maximum
```

Return Value
The highest number allowed as input by the receiver or `nil`, meaning no limit.

Discussion
For versions prior to Mac OS X v10.4 (and number-formatter behavior set to `NSNumberFormatterBehavior10_0`) this method returns an `NSDecimalNumber` object.

Availability
Available in Mac OS X v10.4 and later.

Version returning NSDecimalNumber available prior to Mac OS X v10.4.

See Also
- `- setMaximum:` (page 1119)
- `+ setDefaultFormatterBehavior:` (page 1088)
- `- formatterBehavior` (page 1093)
- `- setFormatterBehavior:` (page 1115)

Declared In
`NSNumberFormatter.h`

maximumFractionDigits

Returns the maximum number of digits after the decimal separator allowed as input by the receiver.

```
- (NSUInteger)maximumFractionDigits
```

Return Value
The maximum number of digits after the decimal separator allowed as input by the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `- setMaximumFractionDigits:` (page 1120)

Declared In
`NSNumberFormatter.h`

maximumIntegerDigits

Returns the maximum number of integer digits allowed as input by the receiver.

```
- (NSUInteger)maximumIntegerDigits
```

Return Value
The maximum number of integer digits allowed as input by the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `- setMaximumIntegerDigits:` (page 1120)

Declared In
`NSNumberFormatter.h`

maximumSignificantDigits

Returns the maximum number of significant digits for the receiver.

```
- (NSUInteger)maximumSignificantDigits
```

Return Value
The maximum number of significant digits for the receiver.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `setMaximumSignificantDigits:` (page 1121)
- `minimumSignificantDigits` (page 1101)
- `usesSignificantDigits` (page 1140)

Declared In
`NSNumberFormatter.h`

minimum

Returns the lowest number allowed as input by the receiver.

```
- (NSNumber *)minimum
```

Return Value
The lowest number allowed as input by the receiver or `nil`, meaning no limit.

Discussion
For versions prior to Mac OS X v10.4 (and number-formatter behavior set to `NSNumberFormatterBehavior10_0`) this method returns an `NSDecimalNumber` object.

Availability
Available in Mac OS X v10.4 and later. Version returning NSDecimalNumber available prior to Mac OS X v10.4.

See Also
- `setMinimum:` (page 1121)
- `setDefaultFormatterBehavior:` (page 1088)
- `formatterBehavior` (page 1093)
- `setFormatterBehavior:` (page 1115)

Declared In
`NSNumberFormatter.h`

minimumFractionDigits

Returns the minimum number of digits after the decimal separator allowed as input by the receiver.

```
- (NSUInteger)minimumFractionDigits
```

Return Value
The minimum number of digits after the decimal separator allowed as input by the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setMinimumFractionDigits:` (page 1122)

Declared In
`NSNumberFormatter.h`

minimumIntegerDigits

Returns the minimum number of integer digits allowed as input by the receiver.

```
- (NSUInteger)minimumIntegerDigits
```

Return Value
The minimum number of integer digits allowed as input by the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setMinimumIntegerDigits:` (page 1122)

Declared In
`NSNumberFormatter.h`

minimumSignificantDigits

Returns the minimum number of significant digits for the receiver.

```
- (NSUInteger)minimumSignificantDigits
```

Return Value
The minimum number of significant digits for the receiver.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `setMinimumSignificantDigits:` (page 1123)
- `maximumSignificantDigits` (page 1099)
- `usesSignificantDigits` (page 1140)

Declared In
`NSNumberFormatter.h`

minusSign

Returns the string the receiver uses to represent the minus sign.

```
- (NSString *)minusSign
```

Return Value
The string that represents the receiver's minus sign.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setMinusSign:` (page 1123)

Declared In
`NSNumberFormatter.h`

multiplier

Returns the multiplier used by the receiver as an `NSNumber` object.

```
- (NSNumber *)multiplier
```

Discussion
A multiplier is a factor used in conversions between numbers and strings (that is, numbers as stored and numbers as displayed). When the input value is a string, the multiplier is used to divide, and when the input value is a number, the multiplier is used to multiply. These operations allow the formatted values to be different from the values that a program manipulates internally.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setMultiplier:` (page 1123)

Declared In
`NSNumberFormatter.h`

negativeFormat

Returns the format used by the receiver to display negative numbers.

```
- (NSString *)negativeFormat
```

Availability
Available in Mac OS X v10.0 and later.

See Also
- `setNegativeFormat:` (page 1124)
- `setFormat:` (page 1114)

Declared In
`NSNumberFormatter.h`

negativeInfinitySymbol

Returns the symbol the receiver uses to represent negative infinity.

```
- (NSString *)negativeInfinitySymbol
```

Return Value
The symbol the receiver uses to represent negative infinity.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setNegativeInfinitySymbol:` (page 1124)

Declared In
`NSNumberFormatter.h`

negativePrefix

Returns the string the receiver inserts as a prefix to negative values.

```
- (NSString *)negativePrefix
```

Return Value
The string the receiver inserts as a prefix to negative values.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `negativeSuffix` (page 1103)
- `setNegativePrefix:` (page 1125)

Declared In
`NSNumberFormatter.h`

negativeSuffix

Returns the string the receiver adds as a suffix to negative values.

```
- (NSString *)negativeSuffix
```

Return Value
The string the receiver adds as a suffix to negative values.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `negativePrefix` (page 1103)
- `setNegativeSuffix:` (page 1125)

Declared In
`NSNumberFormatter.h`

nilSymbol

Returns the string the receiver uses to represent a `nil` value.

```
- (NSString *)nilSymbol
```

Return Value
The string the receiver uses to represent a `nil` value.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setNilSymbol:` (page 1125)

Declared In
`NSNumberFormatter.h`

notANumberSymbol

Returns the symbol the receiver uses to represent NaN (“not a number”) when it converts values.

```
- (NSString *)notANumberSymbol
```

Return Value
The symbol the receiver uses to represent NaN (“not a number”) when it converts values.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setNotANumberSymbol:` (page 1126)

Declared In
`NSNumberFormatter.h`

numberFromString:

Returns an `NSNumber` object created by parsing a given string.

```
- (NSNumber *)numberFromString:(NSString *)string
```

Parameters
string
An `NSString` object that is parsed to generate the returned number object.

Return Value
An `NSNumber` object created by parsing *string* using the receiver’s format.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- stringFromNumber:` (page 1136)

Related Sample Code
NumberInput_IMKit_Sample

Declared In
NSNumberFormatter.h

numberStyle

Returns the number-formatter style of the receiver.

```
- (NSNumberFormatterStyle)numberStyle
```

Return Value
An NSNumberFormatterStyle constant that indicates the number-formatter style of the receiver.

Discussion
Styles are essentially predetermined sets of values for certain properties. Examples of number-formatter styles are those used for decimal values, percentage values, and currency.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setNumberStyle: (page 1126)

Declared In
NSNumberFormatter.h

paddingCharacter

Returns a string containing the padding character for the receiver.

```
- (NSString *)paddingCharacter
```

Availability
Available in Mac OS X v10.4 and later.

See Also
- setPaddingCharacter: (page 1127)

Declared In
NSNumberFormatter.h

paddingPosition

Returns the padding position of the receiver.

```
- (NSNumberFormatterPadPosition)paddingPosition
```

Discussion
The returned constant indicates whether the padding is before or after the number's prefix or suffix.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setPaddingPosition:` (page 1127)

Declared In
`NSNumberFormatter.h`

percentSymbol

Returns the string that the receiver uses to represent the percent symbol.

```
- (NSString *)percentSymbol
```

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setPercentSymbol:` (page 1128)

Declared In
`NSNumberFormatter.h`

perMillSymbol

Returns the string that the receiver uses for the per-thousands symbol.

```
- (NSString *)perMillSymbol
```

Return Value
The string that the receiver uses for the per-thousands symbol.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setPerMillSymbol:` (page 1128)

Declared In
`NSNumberFormatter.h`

plusSign

Returns the string the receiver uses for the plus sign.

```
- (NSString *)plusSign
```

Return Value
The string the receiver uses for the plus sign.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setPlusSign:` (page 1128)

Declared In
`NSNumberFormatter.h`

positiveFormat

Returns the format used by the receiver to display positive numbers.

```
- (NSString *)positiveFormat
```

Availability
Available in Mac OS X v10.0 and later.

See Also
- `setPositiveFormat:` (page 1129)
- `setFormat:` (page 1114)

Declared In
`NSNumberFormatter.h`

positiveInfinitySymbol

Returns the string the receiver uses for the positive infinity symbol.

```
- (NSString *)positiveInfinitySymbol
```

Return Value
The string the receiver uses for the positive infinity symbol.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setPositiveInfinitySymbol:` (page 1129)

Declared In
`NSNumberFormatter.h`

positivePrefix

Returns the string the receiver uses as the prefix for positive values.

```
- (NSString *)positivePrefix
```

Return Value
The string the receiver uses as the prefix for positive values.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setPositivePrefix:` (page 1129)

Declared In
`NSNumberFormatter.h`

positiveSuffix

Returns the string the receiver uses as the suffix for positive values.

```
- (NSString *)positiveSuffix
```

Return Value
The string the receiver uses as the suffix for positive values.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setPositiveSuffix:` (page 1130)

Declared In
`NSNumberFormatter.h`

roundingBehavior

Returns an `NSDecimalNumberHandler` object indicating the rounding behavior of the receiver.

```
- (NSDecimalNumberHandler *)roundingBehavior
```

Return Value
An `NSDecimalNumberHandler` object indicating the rounding behavior of the receiver.

Special Considerations
This method is for use with formatters using `NSNumberFormatterBehavior10_0` behavior.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `setRoundingBehavior:` (page 1130)

Declared In
`NSNumberFormatter.h`

roundingIncrement

Returns the rounding increment used by the receiver.

```
- (NSNumber *)roundingIncrement
```

Return Value
The rounding increment used by the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setRoundingIncrement: (page 1131)

Declared In
NSNumberFormatter.h

roundingMode

Returns the rounding mode used by the receiver.

```
- (NSNumberFormatterRoundingMode)roundingMode
```

Return Value
The rounding mode used by the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setRoundingMode: (page 1131)

Declared In
NSNumberFormatter.h

secondaryGroupingSize

Returns the size of secondary groupings for the receiver.

```
- (NSUInteger)secondaryGroupingSize
```

Return Value
The size of secondary groupings for the receiver.

Discussion
Some locales allow the specification of another grouping size for larger numbers. For example, some locales may represent a number such as 61, 242, 378.46 (as in the United States) as 6,12,42,378.46. In this case, the secondary grouping size (covering the groups of digits furthest from the decimal point) is 2.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setSecondaryGroupingSize: (page 1131)

Declared In
NSNumberFormatter.h

setAllowsFloats:

Sets whether the receiver allows as input floating-point values (that is, values that include the period character [.]).

```
- (void)setAllowsFloats:(BOOL)flag
```

Parameters

flag

YES if the receiver allows floating-point values, NO otherwise.

Discussion
By default, floating point values are allowed as input.

Availability
Available in Mac OS X v10.0 and later.

See Also
- allowsFloats (page 1088)

Related Sample Code
Quartz Composer QCTV

Declared In
NSNumberFormatter.h

setAlwaysShowsDecimalSeparator:

Controls whether the receiver always shows the decimal separator, even for integer numbers.

```
- (void)setAlwaysShowsDecimalSeparator:(BOOL)flag
```

Parameters

flag

YES if the receiver should always show the decimal separator, NO otherwise.

Availability
Available in Mac OS X v10.4 and later.

See Also
- alwaysShowsDecimalSeparator (page 1089)

Declared In
NSNumberFormatter.h

setAttributedStringForNil:

Sets the attributed string the receiver uses to display nil values.

```
- (void)setAttributedStringForNil:(NSAttributedString *)newAttributedString
```

Parameters

newAttributedString

An NSAttributedString object that the receiver uses to display nil values.

Special Considerations

This method is for use with formatters using `NSNumberFormatterBehavior10_0` behavior.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- attributedStringForNil` (page 1089)

Declared In
`NSNumberFormatter.h`

setAttributedStringForNotANumber:

Sets the attributed string the receiver uses to display "not a number" values.

```
- (void)setAttributedStringForNotANumber:(NSAttributedString *)newAttributedString
```

Parameters

newAttributedString
An `NSAttributedString` object that the receiver uses to display NaN values.

Special Considerations

This method is for use with formatters using `NSNumberFormatterBehavior10_0` behavior.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- attributedStringForNotANumber` (page 1090)

Declared In
`NSNumberFormatter.h`

setAttributedStringForZero:

Sets the attributed string that the receiver uses to display zero values.

```
- (void)setAttributedStringForZero:(NSAttributedString *)newAttributedString
```

Parameters

newAttributedString
An `NSAttributedString` object that the receiver uses to display zero values.

Special Considerations

This method is for use with formatters using `NSNumberFormatterBehavior10_0` behavior.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- attributedStringForZero` (page 1090)

Declared In
NSNumberFormatter.h

setCurrencyCode:

Sets the receiver's currency code.

```
- (void)setCurrencyCode:(NSString *)string
```

Parameters

string
A string specifying the receiver's new currency code.

Discussion
A currency code is a three-letter code that is, in most cases, composed of a country's two-character Internet country code plus an extra character to denote the currency unit. For example, the currency code for the Australian dollar is "AUD". Currency codes are based on the ISO 4217 standard.

Availability
Available in Mac OS X v10.4 and later.

See Also
- currencyCode (page 1091)

Declared In
NSNumberFormatter.h

setCurrencyDecimalSeparator:

Sets the string used by the receiver as a decimal separator.

```
- (void)setCurrencyDecimalSeparator:(NSString *)string
```

Parameters

string
The string to use as the currency decimal separator.

Availability
Available in Mac OS X v10.4 and later.

See Also
- currencyDecimalSeparator (page 1091)

Declared In
NSNumberFormatter.h

setCurrencyGroupingSeparator:

Sets the currency grouping separator for the receiver.

```
- (NSString *)setCurrencyGroupingSeparator:(NSString *)string
```

Parameters

string

The currency grouping separator for the receiver.

Availability

Available in Mac OS X v10.5 and later.

Declared In

NSNumberFormatter.h

setCurrencySymbol:

Sets the string used by the receiver as a local currency symbol.

```
- (void)setCurrencySymbol:(NSString *)string
```

Parameters

string

A string that represents a local currency symbol.

Discussion

The local symbol is used within the country, while the international currency symbol is used in international contexts to specify that country's currency unambiguously. The local currency symbol is often represented by a Unicode code point.

Availability

Available in Mac OS X v10.4 and later.

See Also

- `currencySymbol` (page 1092)
- `setInternationalCurrencySymbol:` (page 1118)

Declared In

NSNumberFormatter.h

setDecimalSeparator:

Sets the character the receiver uses as a decimal separator.

```
- (void)setDecimalSeparator:(NSString *)newSeparator
```

Parameters

newSeparator

The string that specifies the decimal-separator character to use. If *newSeparator* contains multiple characters, only the first one is used.

Discussion

If you don't have decimal separators enabled through another means (such as `setFormat:` (page 1114)), using this method enables them.

Availability

Available in Mac OS X v10.0 and later.

See Also
- `decimalSeparator` (page 1092)
- `formatterBehavior` (page 1093)

Declared In
`NSNumberFormatter.h`

setExponentSymbol:

Sets the string used by the receiver to represent the exponent symbol.

```
- (void)setExponentSymbol:(NSString *)string
```

Parameters

string

A string that represents an exponent symbol.

Discussion
The exponent symbol is the “E” or “e” in the scientific notation of numbers, as in 1.0e+56.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `exponentSymbol` (page 1092)

Declared In
`NSNumberFormatter.h`

setFormat:

Sets the receiver’s format.

```
- (void)setFormat:(NSString *)aFormat
```

Parameters

aFormat

A string that can consist of one, two, or three parts separated by “;”. The first part of the string represents the positive format, the second part of the string represents the zero value, and the last part of the string represents the negative format. If the string has just two parts, the first one becomes the positive format, and the second one becomes the negative format. If the string has just one part, it becomes the positive format, and default formats are provided for zero and negative values based on the positive format. For more discussion of this subject, see *Data Formatting Programming Guide for Cocoa*.

Discussion
The following code excerpt shows the three different approaches for setting an `NSNumberFormatter` object’s format using `setFormat:`:

```
NSNumberFormatter *numberFormatter =
    [[[NSNumberFormatter alloc] init] autorelease];

// specify just positive format
[numberFormatter setFormat:@"$#,##0.00"];
```

```
// specify positive and negative formats
[numberFormatter setFormat:@"$#,##0.00;($#,##0.00)"];

// specify positive, zero, and negative formats
[numberFormatter setFormat:@"$#,###.00;0.00;($#,##0.00)"];
```

Special Considerations

This method is for use with formatters using `NSNumberFormatterBehavior10_0` behavior.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- format` (page 1093)

Declared In
`NSNumberFormatter.h`

setFormatterBehavior:

Sets the formatter behavior of the receiver.

```
- (void)setFormatterBehavior:(NSNumberFormatterBehavior)behavior
```

Parameters

behavior

An `NSNumberFormatterBehavior` constant that indicates the revision of the `NSNumberFormatter` class providing the current behavior.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- formatterBehavior` (page 1093)

Related Sample Code
TrackBall

Declared In
`NSNumberFormatter.h`

setFormatWidth:

Sets the format width used by the receiver.

```
- (void)setFormatWidth:(NSUInteger)number
```

Parameters

number

An integer that specifies the format width.

Discussion
The format width is the number of characters of a formatted number within a string that is either left justified or right justified based on the value returned from `paddingPosition` (page 1105).

Availability
Available in Mac OS X v10.4 and later.

See Also
- `formatWidth` (page 1094)

Declared In
`NSNumberFormatter.h`

setGeneratesDecimalNumbers:

Controls whether the receiver creates instances of `NSDecimalNumber` when it converts strings to number objects.

```
- (void)setGeneratesDecimalNumbers:(BOOL)flag
```

Parameters

flag

`YES` if the receiver should generate `NSDecimalNumber` instances, `NO` if it should generate `NSNumber` instances.

Discussion
The default is `YES`.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `generatesDecimalNumbers` (page 1094)

Declared In
`NSNumberFormatter.h`

setGroupingSeparator:

Specifies the string used by the receiver for a grouping separator.

```
- (void)setGroupingSeparator:(NSString *)string
```

Parameters

string

A string that specifies the grouping separator to use.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `groupingSeparator` (page 1095)

Declared In
NSNumberFormatter.h

setGroupingSize:

Sets the grouping size of the receiver.

```
- (void)setGroupingSize:(NSUInteger)numDigits
```

Parameters

numDigits
An integer that specifies the grouping size.

Availability
Available in Mac OS X v10.4 and later.

See Also
- groupingSize (page 1095)

Declared In
NSNumberFormatter.h

setHasThousandSeparators:

Sets whether the receiver uses thousand separators.

```
- (void)setHasThousandSeparators:(BOOL)flag
```

Parameters

flag
When *flag* is NO, thousand separators are disabled for both positive and negative formats (even if you've set them through another means, such as setFormat: (page 1114)). When *flag* is YES, thousand separators are used.

Discussion
In addition to using this method to add thousand separators to your format, you can also use it to disable thousand separators if you've set them using another method. The default is NO (though you in effect change this setting to YES when you set thousand separators through any means, such as setFormat: (page 1114)).

Special Considerations

This method is for use with formatters using NSNumberFormatterBehavior10_0 behavior.

Availability
Available in Mac OS X v10.0 and later.

See Also
- hasThousandSeparators (page 1096)

Declared In
NSNumberFormatter.h

setInternationalCurrencySymbol:

Sets the string used by the receiver for the international currency symbol.

```
- (void)setInternationalCurrencySymbol:(NSString *)string
```

Parameters

string

A string that represents an international currency symbol.

Discussion

The local symbol is used within the country, while the international currency symbol is used in international contexts to specify that country's currency unambiguously. The local currency symbol is often represented by a Unicode code point.

Availability

Available in Mac OS X v10.4 and later.

See Also

- internationalCurrencySymbol (page 1096)

Declared In

NSNumberFormatter.h

setLenient:

Sets whether the receiver will use heuristics to guess at the number which is intended by a string.

```
- (void)setLenient:(BOOL)b
```

Parameters

b

YES if the receiver will use heuristics to guess at the number which is intended by the string; otherwise NO.

Discussion

If the formatter is set to be lenient, as with any guessing it may get the result number wrong (that is, a number other than that which was intended).

Availability

Available in Mac OS X v10.5 and later.

See Also

- isLenient (page 1097)

Declared In

NSNumberFormatter.h

setLocale:

Sets the locale of the receiver.

```
- (void)setLocale:(NSLocale *)theLocale
```

Parameters

theLocale

An `NSLocale` object representing the new locale of the receiver.

Discussion

The locale determines the default values for many formatter attributes, such as ISO country and language codes, currency code, calendar, system of measurement, and decimal separator.

Availability

Available in Mac OS X v10.4 and later.

See Also

- `locale` (page 1097)

Declared In

`NSNumberFormatter.h`

setLocalizesFormat:

Sets whether the dollar sign character (`$`), decimal separator character (`.`), and thousand separator character (`,`) are converted to appropriately localized characters as specified by the user's localization preference.

```
- (void)setLocalizesFormat:(BOOL)flag
```

Parameters

flag

`YES` if these characters are converted to the localized equivalents, `NO` otherwise.

Discussion

While the currency-symbol part of this feature may be useful in certain types of applications, it's probably more likely that you would tie a particular application to a particular currency (that is, that you would "hard-code" the currency symbol and separators instead of having them dynamically change based on the user's configuration). The reason for this, of course, is that `NSNumberFormatter` doesn't perform currency conversions, it just formats numeric data. You wouldn't want one user interpreting the value `"56324"` as US currency and another user who's accessing the same data interpreting it as Japanese currency, simply based on each user's localization preferences.

Special Considerations

This method is for use with formatters using `NSNumberFormatterBehavior10_0` behavior.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `localizesFormat` (page 1098)

Declared In

`NSNumberFormatter.h`

setMaximum:

Sets the highest number the receiver allows as input.

```
- (void)setMaximum:(NSNumber *)aMaximum
```

Parameters

aMaximum

A number object that specifies a maximum input value.

Discussion

If *aMaximum* is `nil`, checking for the maximum value is disabled. For versions prior to Mac OS X v10.4 (and number-formatter behavior set to `NSNumberFormatterBehavior10_0`) this method requires an `NSDecimalNumber` argument.

Availability

Available in Mac OS X v10.4 and later. Version requiring NSDecimalNumber argument available prior to Mac OS X v10.4.

See Also

- `maximum` (page 1098)
+ `setDefaultFormatterBehavior:` (page 1088)
- `formatterBehavior` (page 1093)
- `setFormatterBehavior:` (page 1115)

Related Sample Code

Quartz Composer QCTV

Declared In

`NSNumberFormatter.h`

setMaximumFractionDigits:

Sets the maximum number of digits after the decimal separator allowed as input by the receiver.

```
- (void)setMaximumFractionDigits:(NSUInteger)number
```

Parameters

number

The maximum number of digits after the decimal separator allowed as input.

Availability

Available in Mac OS X v10.4 and later.

See Also

- `maximumFractionDigits` (page 1099)

Related Sample Code

TrackBall

Declared In

`NSNumberFormatter.h`

setMaximumIntegerDigits:

Sets the maximum number of integer digits allowed as input by the receiver.

```
- (void)setMaximumIntegerDigits:(NSUInteger)number
```

Parameters

number

The maximum number of integer digits allowed as input.

Availability
Available in Mac OS X v10.4 and later.

See Also
- minimumIntegerDigits (page 1101)

Declared In
NSNumberFormatter.h

setMaximumSignificantDigits:

Sets the maximum number of significant digits for the receiver.

```
- (void)setMaximumSignificantDigits:(NSUInteger)number
```

Parameters

number

The maximum number of significant digits for the receiver.

Availability
Available in Mac OS X v10.5 and later.

See Also
- maximumSignificantDigits (page 1099)
- setMinimumSignificantDigits: (page 1123)
- usesSignificantDigits (page 1140)

Declared In
NSNumberFormatter.h

setMinimum:

Sets the lowest number the receiver allows as input.

```
- (void)setMinimum:(NSNumber *)aMinimum
```

Parameters

aMinimum

A number object that specifies a minimum input value.

Discussion
If *aMinimum* is nil, checking for the minimum value is disabled. For versions prior to Mac OS X v10.4 (and number-formatter behavior set to NSNumberFormatterBehavior10_0) this method requires an NSDecimalNumber argument.

Availability
Available in Mac OS X v10.4 and later. Version requiring NSDecimalNumber argument available prior to Mac OS X v10.4.

See Also
- `- minimum` (page 1100)
- `+ setDefaultFormatterBehavior:` (page 1088)
- `- formatterBehavior` (page 1093)
- `- setFormatterBehavior:` (page 1115)

Related Sample Code
Quartz Composer QCTV

Declared In
`NSNumberFormatter.h`

setMinimumFractionDigits:

Sets the minimum number of digits after the decimal separator allowed as input by the receiver.

```
- (void)setMinimumFractionDigits:(NSUInteger)number
```

Parameters

number
The minimum number of digits after the decimal separator allowed as input.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `- minimumFractionDigits` (page 1100)

Related Sample Code
TrackBall

Declared In
`NSNumberFormatter.h`

setMinimumIntegerDigits:

Sets the minimum number of integer digits allowed as input by the receiver.

```
- (void)setMinimumIntegerDigits:(NSUInteger)number
```

Parameters

number
The minimum number of integer digits allowed as input.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `- minimumIntegerDigits` (page 1101)

Related Sample Code
TrackBall

Declared In
NSNumberFormatter.h

setMinimumSignificantDigits:

Sets the minimum number of significant digits for the receiver.

```
- (void)setMinimumSignificantDigits:(NSUInteger)number
```

Parameters

number
The minimum number of significant digits for the receiver.

Availability
Available in Mac OS X v10.5 and later.

See Also
- minimumSignificantDigits (page 1101)
- setMaximumSignificantDigits: (page 1121)
- usesSignificantDigits (page 1140)

Declared In
NSNumberFormatter.h

setMinusSign:

Sets the string used by the receiver for the minus sign.

```
- (void)setMinusSign:(NSString *)string
```

Parameters

string
A string that represents a minus sign.

Availability
Available in Mac OS X v10.4 and later.

See Also
- minusSign (page 1101)

Declared In
NSNumberFormatter.h

setMultiplier:

Sets the multiplier of the receiver.

```
- (void)setMultiplier:(NSNumber *)number
```

Parameters

number
A number object that represents a multiplier.

Discussion
A multiplier is a factor used in conversions between numbers and strings (that is, numbers as stored and numbers as displayed). When the input value is a string, the multiplier is used to divide, and when the input value is a number, the multiplier is used to multiply. These operations allow the formatted values to be different from the values that a program manipulates internally.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `multiplier` (page 1102)

Declared In
`NSNumberFormatter.h`

setNegativeFormat:

Sets the format the receiver uses to display negative values.

```
- (void)setNegativeFormat:(NSString *)aFormat
```

Parameters

aFormat
A string that specifies the format for negative values.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `negativeFormat` (page 1102)
- `setFormat:` (page 1114)

Declared In
`NSNumberFormatter.h`

setNegativeInfinitySymbol:

Sets the string used by the receiver for the negative infinity symbol.

```
- (void)setNegativeInfinitySymbol:(NSString *)string
```

Parameters

string
A string that represents a negative infinity symbol.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `negativeInfinitySymbol` (page 1102)

Declared In
`NSNumberFormatter.h`

setNegativePrefix:

Sets the string the receiver uses as a prefix for negative values.

```
- (void)setNegativePrefix:(NSString *)string
```

Parameters

string

A string to use as the prefix for negative values.

Availability
Available in Mac OS X v10.4 and later.

See Also
- negativePrefix (page 1103)

Declared In
NSNumberFormatter.h

setNegativeSuffix:

Sets the string the receiver uses as a suffix for negative values.

```
- (void)setNegativeSuffix:(NSString *)string
```

Parameters

string

A string to use as the suffix for negative values.

Availability
Available in Mac OS X v10.4 and later.

See Also
- negativeSuffix (page 1103)

Declared In
NSNumberFormatter.h

setNilSymbol:

Sets the string the receiver uses to represent `nil` values.

```
- (void)setNilSymbol:(NSString *)string
```

Parameters

string

A string that represents a `nil` value.

Availability
Available in Mac OS X v10.4 and later.

See Also
- nilSymbol (page 1104)

Declared In
NSNumberFormatter.h

setNotANumberSymbol:

Sets the string the receiver uses to represent NaN (“not a number”).

```
- (void)setNotANumberSymbol:(NSString *)string
```

Parameters

string

A string that represents a NaN symbol.

Availability
Available in Mac OS X v10.4 and later.

See Also
- notANumberSymbol (page 1104)

Declared In
NSNumberFormatter.h

setNumberStyle:

Sets the number style used by the receiver.

```
- (void)setNumberStyle:(NSNumberFormatterStyle)style
```

Parameters

style

An NSNumberFormatterStyle constant that specifies a formatter style.

Discussion
Styles are essentially predetermined sets of values for certain properties. Examples of number-formatter styles are those used for decimal values, percentage values, and currency.

Availability
Available in Mac OS X v10.4 and later.

See Also
- numberStyle (page 1105)

Related Sample Code
Grady
Mountains
NumberInput_IMKit_Sample
TrackBall

Declared In
NSNumberFormatter.h

setPaddingCharacter:

Sets the string that the receiver uses to pad numbers in the formatted string representation.

```
- (void)setPaddingCharacter:(NSString *)string
```

Parameters

string

A string containing a padding character (or characters).

Availability

Available in Mac OS X v10.4 and later.

See Also

- paddingCharacter (page 1105)

Declared In

NSNumberFormatter.h

setPaddingPosition:

Sets the padding position used by the receiver.

```
- (void)setPaddingPosition:(NSNumberFormatterPadPosition)position
```

Parameters

position

An NSNumberFormatterPadPosition constant that indicates a padding position (before or after prefix or suffix).

Availability

Available in Mac OS X v10.4 and later.

See Also

- paddingPosition (page 1105)

Declared In

NSNumberFormatter.h

setPartialStringValidationEnabled:

Sets whether partial string validation is enabled for the receiver.

```
- (void)setPartialStringValidationEnabled:(BOOL)b
```

Parameters

b

YES if partial string validation is enabled, otherwise NO.

Availability

Available in Mac OS X v10.5 and later.

See Also

- isPartialStringValidationEnabled (page 1097)

Declared In
NSNumberFormatter.h

setPercentSymbol:

Sets the string used by the receiver to represent the percent symbol.

```
- (void)setPercentSymbol:(NSString *)string
```

Parameters

string
A string that represents a percent symbol.

Availability
Available in Mac OS X v10.4 and later.

See Also
- percentSymbol (page 1106)

Declared In
NSNumberFormatter.h

setPerMillSymbol:

Sets the string used by the receiver to represent the per-mill (per-thousand) symbol.

```
- (void)setPerMillSymbol:(NSString *)string
```

Parameters

string
A string that represents a per-mill symbol.

Availability
Available in Mac OS X v10.4 and later.

See Also
- perMillSymbol (page 1106)

Declared In
NSNumberFormatter.h

setPlusSign:

Sets the string used by the receiver to represent the plus sign.

```
- (void)setPlusSign:(NSString *)string
```

Parameters

string
A string that represents a plus sign.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `plusSign` (page 1106)

Declared In
`NSNumberFormatter.h`

setPositiveFormat:

Sets the format the receiver uses to display positive values.

```
- (void)setPositiveFormat:(NSString *)aFormat
```

Parameters

aFormat
A string that specifies the format for positive values.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `positiveFormat` (page 1107)
- `setFormat:` (page 1114)

Declared In
`NSNumberFormatter.h`

setPositiveInfinitySymbol:

Sets the string used by the receiver for the positive infinity symbol.

```
- (void)setPositiveInfinitySymbol:(NSString *)string
```

Parameters

string
A string that represents a positive infinity symbol.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `positiveInfinitySymbol` (page 1107)

Declared In
`NSNumberFormatter.h`

setPositivePrefix:

Sets the string the receiver uses as the prefix for positive values.

```
- (void)setPositivePrefix:(NSString *)string
```

Parameters

string

A string to use as the prefix for positive values.

Availability

Available in Mac OS X v10.4 and later.

See Also

- positivePrefix (page 1107)

Declared In

NSNumberFormatter.h

setPositiveSuffix:

Sets the string the receiver uses as the suffix for positive values.

```
- (void)setPositiveSuffix:(NSString *)string
```

Parameters

string

A string to use as the suffix for positive values.

Availability

Available in Mac OS X v10.4 and later.

See Also

- positiveSuffix (page 1108)

Declared In

NSNumberFormatter.h

setRoundingBehavior:

Sets the rounding behavior used by the receiver.

```
- (void)setRoundingBehavior:(NSDecimalNumberHandler *)newRoundingBehavior
```

Parameters

newRoundingBehavior

An NSDecimalNumberHandler object representing a rounding behavior.

Special Considerations

This method is for use with formatters using NSNumberFormatterBehavior10_0 behavior.

Availability

Available in Mac OS X v10.0 and later.

See Also

- roundingBehavior (page 1108)

Declared In

NSNumberFormatter.h

setRoundingIncrement:

Sets the rounding increment used by the receiver.

```
- (void)setRoundingIncrement:(NSNumber *)number
```

Parameters

number

A number object specifying a rounding increment.

Availability
Available in Mac OS X v10.4 and later.

See Also
- roundingIncrement (page 1108)

Declared In
NSNumberFormatter.h

setRoundingMode:

Sets the rounding mode used by the receiver.

```
- (void)setRoundingMode:(NSNumberFormatterRoundingMode)mode
```

Parameters

mode

An NSNumberFormatterRoundingMode constant that indicates a rounding mode.

Availability
Available in Mac OS X v10.4 and later.

See Also
- roundingMode (page 1109)

Declared In
NSNumberFormatter.h

setSecondaryGroupingSize:

Sets the secondary grouping size of the receiver.

```
- (void)setSecondaryGroupingSize:(NSUInteger)number
```

Parameters

number

An integer that specifies the size of secondary groupings.

Discussion
Some locales allow the specification of another grouping size for larger numbers. For example, some locales may represent a number such as 61, 242, 378.46 (as in the United States) as 6,12,42,378.46. In this case, the secondary grouping size (covering the groups of digits furthest from the decimal point) is 2.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `secondaryGroupingSize` (page 1109)

Declared In
`NSNumberFormatter.h`

setTextAttributesForNegativeInfinity:

Sets the text attributes used to display the negative infinity symbol.

```
- (void)setTextAttributesForNegativeInfinity:(NSDictionary *)newAttributes
```

Parameters

newAttributes
A dictionary containing text attributes for the display of the negative infinity symbol.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `textAttributesForNegativeInfinity` (page 1137)
- `setNegativeInfinitySymbol:` (page 1124)

Declared In
`NSNumberFormatter.h`

setTextAttributesForNegativeValues:

Sets the text attributes to be used in displaying negative values .

```
- (void)setTextAttributesForNegativeValues:(NSDictionary *)newAttributes
```

Parameters

newAttributes
A dictionary containing properties for the display of negative values.

Discussion
For example, this code excerpt causes negative values to be displayed in red:

```
NSNumberFormatter *numberFormatter =
    [[[NSNumberFormatter alloc] init] autorelease];
NSMutableDictionary *newAttrs = [NSMutableDictionary dictionary];

[numberFormatter setFormat:@"$#,##0.00;($#,##0.00)"];
[newAttrs setObject:[NSColor redColor] forKey:@"NSColor"];
[numberFormatter setTextAttributesForNegativeValues:newAttrs];
[[textField cell] setFormatter:numberFormatter];
```

An even simpler way to cause negative values to be displayed in red is to include the constant `[Red]` in your format string, as shown in this example:

```
[numberFormatter setFormat:@"$#,##0.00;[Red]($#,##0.00)"];
```

When you set a value's text attributes to use color, the color appears only when the value's cell doesn't have input focus. When the cell has input focus, the value is displayed in standard black.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `textAttributesForNegativeValues` (page 1137)

Declared In
`NSNumberFormatter.h`

setTextAttributesForNil:

Sets the text attributes used to display the `nil` symbol.

```
- (void)setTextAttributesForNil:(NSDictionary *)newAttributes
```

Parameters

newAttributes
A dictionary containing text attributes for the display of the `nil` symbol.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `textAttributesForNil` (page 1138)
- `nilSymbol` (page 1104)

Declared In
`NSNumberFormatter.h`

setTextAttributesForNotANumber:

Sets the text attributes used to display the NaN ("not a number") string.

```
- (void)setTextAttributesForNotANumber:(NSDictionary *)newAttributes
```

Parameters

newAttributes
A dictionary containing text attributes for the display of the NaN symbol.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setTextAttributesForNotANumber:` (page 1133)
- `notANumberSymbol` (page 1104)

Declared In
`NSNumberFormatter.h`

setTextAttributesForPositiveInfinity:

Sets the text attributes used to display the positive infinity symbol.

```
- (void)setTextAttributesForPositiveInfinity:(NSDictionary *)newAttributes
```

Parameters

newAttributes

A dictionary containing text attributes for the display of the positive infinity symbol.

Availability

Available in Mac OS X v10.4 and later.

See Also

- positiveInfinitySymbol (page 1107)
- textAttributesForPositiveInfinity (page 1138)

Declared In

NSNumberFormatter.h

setTextAttributesForPositiveValues:

Sets the text attributes to be used in displaying positive values.

```
- (void)setTextAttributesForPositiveValues:(NSDictionary *)newAttributes
```

Parameters

newAttributes

A dictionary containing text attributes for the display of positive values.

Discussion

See setTextAttributesForNegativeValues: (page 1132) for an example of how a related method might be used.

Availability

Available in Mac OS X v10.0 and later.

See Also

- textAttributesForPositiveValues (page 1139)

Declared In

NSNumberFormatter.h

setTextAttributesForZero:

Sets the text attributes used to display a zero value.

```
- (void)setTextAttributesForZero:(NSDictionary *)newAttributes
```

Parameters

newAttributes

A dictionary containing text attributes for the display of zero values.

Availability

Available in Mac OS X v10.4 and later.

See Also
- `textAttributesForZero` (page 1139)

Declared In
`NSNumberFormatter.h`

setThousandSeparator:

Sets the character the receiver uses as a thousand separator.

```
- (void)setThousandSeparator:(NSString *)newSeparator
```

Parameters

newSeparator

A string that specifies the thousand-separator character to use. If *newSeparator* contains multiple characters, only the first one is used.

Discussion
If you don't have thousand separators enabled through any other means (such as `setFormat:` (page 1114)), using this method enables them.

Special Considerations

This method is for use with formatters using `NSNumberFormatterBehavior10_0` behavior.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `thousandSeparator` (page 1139)

Declared In
`NSNumberFormatter.h`

setUsesGroupingSeparator:

Controls whether the receiver displays the grouping separator.

```
- (void)setUsesGroupingSeparator:(BOOL)flag
```

Parameters

flag

`YES` if the receiver should display the grouping separator, `NO` otherwise.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `usesGroupingSeparator` (page 1140)

Declared In
`NSNumberFormatter.h`

setUsesSignificantDigits:

Sets whether the receiver uses significant digits.

```
- (void)setUsesSignificantDigits:(BOOL)b
```

Parameters

b

`YES` if the receiver uses significant digits, otherwise `NO`.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `usesSignificantDigits` (page 1140)
- `setMaximumSignificantDigits:` (page 1121)
- `setMinimumSignificantDigits:` (page 1123)

Declared In

`NSNumberFormatter.h`

setZeroSymbol:

Sets the string the receiver uses as the symbol to show the value zero.

```
- (void)setZeroSymbol:(NSString *)string
```

Parameters

string

The string the receiver uses as the symbol to show the value zero.

Discussion

By default this is `0`; you might want to set it to, for example, " - ", similar to the way that a spreadsheet might when a column is defined as accounting.

Special Considerations

On Mac OS X v10.4, this method works correctly for 10_0-style number formatters but does not work correctly for 10_4-style number formatters. You can work around the problem by subclassing and overriding the methods that convert between strings and numbers to look for the zero cases first and provide different behavior, invoking `super` when not zero.

Availability

Available in Mac OS X v10.4 and later.

See Also

- `zeroSymbol` (page 1141)

Declared In

`NSNumberFormatter.h`

stringFromNumber:

Returns a string containing the formatted value of the provided number object.

```
- (NSString *)stringFromNumber:(NSNumber *)number
```

Parameters

number

An `NSNumber` object that is parsed to create the returned string object.

Return Value

A string containing the formatted value of *number* using the receiver's current settings.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- numberFromString:` (page 1104)

Related Sample Code

Mountains

NumberInput_IMKit_Sample

Declared In

`NSNumberFormatter.h`

textAttributesForNegativeInfinity

Returns a dictionary containing the text attributes used to display the negative infinity string.

```
- (NSDictionary *)textAttributesForNegativeInfinity
```

Return Value

A dictionary containing the text attributes used to display the negative infinity string.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- setTextAttributesForNegativeInfinity:` (page 1132)

Declared In

`NSNumberFormatter.h`

textAttributesForNegativeValues

Returns a dictionary containing the text attributes that have been set for negative values.

```
- (NSDictionary *)textAttributesForNegativeValues
```

Return Value

A dictionary containing the text attributes that have been set for negative values.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- setTextAttributesForNegativeValues:` (page 1132)

Declared In
NSNumberFormatter.h

textAttributesForNil

Returns a dictionary containing the text attributes used to display the `nil` symbol.

```
- (NSDictionary *)textAttributesForNil
```

Return Value
A dictionary containing the text attributes used to display the `nil` symbol.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setTextAttributesForNil:` (page 1133)

Declared In
NSNumberFormatter.h

textAttributesForNotANumber

Returns a dictionary containing the text attributes used to display the NaN ("not a number") symbol.

```
- (NSDictionary *)textAttributesForNotANumber
```

Return Value
A dictionary containing the text attributes used to display the NaN ("not a number") symbol.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setTextAttributesForNotANumber:` (page 1133)
- `notANumberSymbol` (page 1104)

Declared In
NSNumberFormatter.h

textAttributesForPositiveInfinity

Returns a dictionary containing the text attributes used to display the positive infinity symbol.

```
- (NSDictionary *)textAttributesForPositiveInfinity
```

Return Value
A dictionary containing the text attributes used to display the positive infinity symbol.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setTextAttributesForPositiveInfinity:` (page 1134)
- `positiveInfinitySymbol` (page 1107)

Declared In
`NSNumberFormatter.h`

textAttributesForPositiveValues

Returns a dictionary containing the text attributes that have been set for positive values.

```
- (NSDictionary *)textAttributesForPositiveValues
```

Return Value
A dictionary containing the text attributes that have been set for positive values.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `setTextAttributesForPositiveValues:` (page 1134)

Declared In
`NSNumberFormatter.h`

textAttributesForZero

Returns a dictionary containing the text attributes used to display a value of zero.

```
- (NSDictionary *)textAttributesForZero
```

Return Value
A dictionary containing the text attributes used to display a value of zero.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setTextAttributesForZero:` (page 1134)

Declared In
`NSNumberFormatter.h`

thousandSeparator

Returns a string containing the character the receiver uses to represent thousand separators.

```
- (NSString *)thousandSeparator
```

Return Value
A string containing the character the receiver uses to represent thousand separators.

Discussion
By default this is the comma character (,). Note that the return value doesn't indicate whether thousand separators are enabled.

Special Considerations

This method is for use with formatters using `NSNumberFormatterBehavior10_0` behavior.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setThousandSeparator:` (page 1135)

Declared In
`NSNumberFormatter.h`

usesGroupingSeparator

Returns a Boolean value that indicates whether the receiver uses the grouping separator.

```
- (BOOL)usesGroupingSeparator
```

Return Value
`YES` if the receiver uses the grouping separator, otherwise `NO`.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setUsesGroupingSeparator:` (page 1135)

Declared In
`NSNumberFormatter.h`

usesSignificantDigits

Returns a Boolean value that indicates whether the receiver uses significant digits.

```
- (BOOL)usesSignificantDigits
```

Return Value
`YES` if the receiver uses significant digits, otherwise `NO`.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- setUsesSignificantDigits:` (page 1136)
`- maximumSignificantDigits` (page 1099)
`- minimumSignificantDigits` (page 1101)

Declared In
`NSNumberFormatter.h`

zeroSymbol

Returns the string the receiver uses as the symbol to show the value zero.

```
- (NSString *)zeroSymbol
```

Return Value
The string the receiver uses as the symbol to show the value zero.

Discussion
For a discussion of how this is used, see `setZeroSymbol:` (page 1136).

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setZeroSymbol:` (page 1136)

Declared In
`NSNumberFormatter.h`

Constants

NSNumberFormatterStyle

These constants specify predefined number format styles.

```
typedef enum {
    NSNumberFormatterNoStyle = kCFNumberFormatterNoStyle,
    NSNumberFormatterDecimalStyle = kCFNumberFormatterDecimalStyle,
    NSNumberFormatterCurrencyStyle = kCFNumberFormatterCurrencyStyle,
    NSNumberFormatterPercentStyle = kCFNumberFormatterPercentStyle,
    NSNumberFormatterScientificStyle = kCFNumberFormatterScientificStyle,
    NSNumberFormatterSpellOutStyle = kCFNumberFormatterSpellOutStyle
} NSNumberFormatterStyle;
```

Constants

`NSNumberFormatterNoStyle`
Specifies no style.
Available in Mac OS X v10.4 and later.
Declared in `NSNumberFormatter.h`.

`NSNumberFormatterDecimalStyle`
Specifies a decimal style format.
Available in Mac OS X v10.4 and later.
Declared in `NSNumberFormatter.h`.

`NSNumberFormatterCurrencyStyle`
Specifies a currency style format.
Available in Mac OS X v10.4 and later.
Declared in `NSNumberFormatter.h`.

`NSNumberFormatterPercentStyle`
Specifies a percent style format.

Available in Mac OS X v10.4 and later.

Declared in `NSNumberFormatter.h`.

`NSNumberFormatterScientificStyle`
Specifies a scientific style format.

Available in Mac OS X v10.4 and later.

Declared in `NSNumberFormatter.h`.

`NSNumberFormatterSpellOutStyle`
Specifies a spell-out format; for example, “23” becomes “twenty-three”.

Available in Mac OS X v10.4 and later.

Declared in `NSNumberFormatter.h`.

Discussion
These constants are used by the `numberStyle` (page 1105) and `setNumberStyle:` (page 1126) methods.

Declared In
`NSNumberFormatter.h`

NSNumberFormatterBehavior

These constants specify the behavior of a number formatter.

```
typedef enum {
    NSNumberFormatterBehaviorDefault = 0,
    NSNumberFormatterBehavior10_0 = 1000,
    NSNumberFormatterBehavior10_4 = 1040,
} NSNumberFormatterBehavior;
```

Constants
`NSNumberFormatterBehaviorDefault`
The number-formatter behavior set as the default for new instances. You can set the default formatter behavior with the class method `setDefaultFormatterBehavior:` (page 1088).

Available in Mac OS X v10.4 and later.

Declared in `NSNumberFormatter.h`.

`NSNumberFormatterBehavior10_0`
The number-formatter behavior as it existed prior to Mac OS X v10.4.

Available in Mac OS X v10.4 and later.

Declared in `NSNumberFormatter.h`.

`NSNumberFormatterBehavior10_4`
The number-formatter behavior since Mac OS X v10.4.

Available in Mac OS X v10.4 and later.

Declared in `NSNumberFormatter.h`.

Discussion
These constants are returned by the `defaultFormatterBehavior` (page 1087) class method and the `formatterBehavior` (page 1093) instance methods; you set them with the `setDefaultFormatterBehavior:` (page 1088) class method and the `setFormatterBehavior:` (page 1115) instance method.

Declared In
NSNumberFormatter.h

NSNumberFormatterPadPosition

These constants are used to specify how numbers should be padded.

```
typedef enum {
    NSNumberFormatterPadBeforePrefix = kCFNumberFormatterPadBeforePrefix,
    NSNumberFormatterPadAfterPrefix = kCFNumberFormatterPadAfterPrefix,
    NSNumberFormatterPadBeforeSuffix = kCFNumberFormatterPadBeforeSuffix,
    NSNumberFormatterPadAfterSuffix = kCFNumberFormatterPadAfterSuffix
} NSNumberFormatterPadPosition;
```

Constants

`NSNumberFormatterPadBeforePrefix`
Specifies that the padding should occur before the prefix.
Available in Mac OS X v10.4 and later.
Declared in `NSNumberFormatter.h`.

`NSNumberFormatterPadAfterPrefix`
Specifies that the padding should occur after the prefix.
Available in Mac OS X v10.4 and later.
Declared in `NSNumberFormatter.h`.

`NSNumberFormatterPadBeforeSuffix`
Specifies that the padding should occur before the suffix.
Available in Mac OS X v10.4 and later.
Declared in `NSNumberFormatter.h`.

`NSNumberFormatterPadAfterSuffix`
Specifies that the padding should occur after the suffix.
Available in Mac OS X v10.4 and later.
Declared in `NSNumberFormatter.h`.

Discussion
These constants are used by the `paddingPosition` (page 1105) and `setPaddingPosition:` (page 1127) methods.

Declared In
NSNumberFormatter.h

NSNumberFormatterRoundingMode

These constants are used to specify how numbers should be rounded.

```
typedef enum {
    NSNumberFormatterRoundCeiling = kCFNumberFormatterRoundCeiling,
    NSNumberFormatterRoundFloor = kCFNumberFormatterRoundFloor,
    NSNumberFormatterRoundDown = kCFNumberFormatterRoundDown,
    NSNumberFormatterRoundUp = kCFNumberFormatterRoundUp,
    NSNumberFormatterRoundHalfEven = kCFNumberFormatterRoundHalfEven,
    NSNumberFormatterRoundHalfDown = kCFNumberFormatterRoundHalfDown,
    NSNumberFormatterRoundHalfUp = kCFNumberFormatterRoundHalfUp
} NSNumberFormatterRoundingMode;
```

Constants

`NSNumberFormatterRoundCeiling`
Round up to next larger number with the proper number of digits after the decimal separator.
Available in Mac OS X v10.4 and later.
Declared in `NSNumberFormatter.h`.

`NSNumberFormatterRoundFloor`
Round down to next smaller number with the proper number of digits after the decimal separator.
Available in Mac OS X v10.4 and later.
Declared in `NSNumberFormatter.h`.

`NSNumberFormatterRoundDown`
Round down to next smaller number with the proper number of digits after the decimal separator.
Available in Mac OS X v10.4 and later.
Declared in `NSNumberFormatter.h`.

`NSNumberFormatterRoundHalfEven`
Round the last digit, when followed by a 5, toward an even digit (.25 -> .2, .35 -> .4)
Available in Mac OS X v10.4 and later.
Declared in `NSNumberFormatter.h`.

`NSNumberFormatterRoundUp`
Round up to next larger number with the proper number of digits after the decimal separator.
Available in Mac OS X v10.4 and later.
Declared in `NSNumberFormatter.h`.

`NSNumberFormatterRoundHalfDown`
Round down when a 5 follows putative last digit.
Available in Mac OS X v10.4 and later.
Declared in `NSNumberFormatter.h`.

`NSNumberFormatterRoundHalfUp`
Round up when a 5 follows putative last digit.
Available in Mac OS X v10.4 and later.
Declared in `NSNumberFormatter.h`.

Declared In

`NSNumberFormatter.h`

These constants are used by the `roundingMode` (page 1109)and `setRoundingMode:` (page 1131) methods.

NSObject Class Reference

Inherits from	none (NSObject is a root class)
Conforms to	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSObject.h Foundation/NSArchiver.h Foundation/NSClassDescription.h Foundation/NSConnection.h Foundation/NSKeyedArchiver.h Foundation/NSObjectScripting.h Foundation/NSPortCoder.h Foundation/NSRunLoop.h Foundation/NSScriptClassDescription.h Foundation/NSThread.h
Companion guide	Cocoa Fundamentals Guide
Related sample code	CoreRecipes Dicey ImageClient Quartz Composer WWDC 2005 TextEdit StickiesExample

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.

Selectors

`NSObject` has some special methods that take advantage of the Objective-C runtime system. For example, you can ask a class or instance if it responds to a message before sending it a message. You can also ask for a method implementation and invoke it using one of the `perform...` methods, or as a function. The advantage of obtaining a method's implementation and calling it as a function is that you can invoke the implementation multiple times within a loop, or similar C construct, without the overhead of Objective-C messaging.

These and other NSObject methods take a selector of type SEL as an argument. For efficiency, full ASCII names are not used to represent methods in compiled code. Instead the compiler uses a unique identifier to represent a method at runtime called a **selector**. A selector for a method name is obtained using the @selector() directive:

```
SEL method = @selector(isEqual:);
```

The instanceMethodForSelector: (page 1159) class method and the methodForSelector: (page 1181) instance method return a method implementation of type IMP. IMP is defined as a pointer to a function that returns an id and takes a variable number of arguments (in addition to the two "hidden" arguments—self and _cmd—that are passed to every method implementation):

```
typedef id (*IMP)(id, SEL, ...);
```

This definition serves as a prototype for the function pointer returned by these methods. It's sufficient for methods that return an object and take object arguments. However, if the selector takes different argument types or returns anything but an id, its function counterpart will be inadequately prototyped. Lacking a prototype, the compiler will promote floats to doubles and chars to ints, which the implementation won't expect. It will therefore behave differently (and erroneously) when performed as a method.

To remedy this situation, it's necessary to provide your own prototype. In the example below, the declaration of the test variable serves to prototype the implementation of the isEqual: method. test is defined as a pointer to a function that returns a BOOL and takes an id argument (in addition to the two "hidden" arguments). The value returned by methodForSelector: (page 1181) is then similarly cast to be a pointer to this same function type:

```
BOOL (*test)(id, SEL, id);
test = (BOOL (*)(id, SEL, id))[target methodForSelector:@selector(isEqual:)];

while ( !test(target, @selector(isEqual:), someObject) ) {
    ...
}
```

In some cases, it might be clearer to define a type (similar to IMP) that can be used both for declaring the variable and for casting the function pointer methodForSelector: (page 1181) returns. The example below defines the EqualIMP type for just this purpose:

```
typedef BOOL (*EqualIMP)(id, SEL, id);
EqualIMP test;
test = (EqualIMP)[target methodForSelector:@selector(isEqual:)];

while ( !test(target, @selector(isEqual:), someObject) ) {
    ...
}
```

Either way, it's important to cast the return value of methodForSelector: (page 1181) to the appropriate function type. It's not sufficient to simply call the function returned by methodForSelector: and cast the result of that call to the desired type. Doing so can result in errors.

See "How Messaging Works" in *The Objective-C 2.0 Programming Language* for more information.

Adopted Protocols

NSObject
- - autorelease (page 2099)
- - class (page 2100)
- - conformsToProtocol: (page 2100)
- - description (page 2100)
- - hash (page 2101)
- - isEqual: (page 2101)
- - isKindOfClass: (page 2102)
- - isMemberOfClass: (page 2103)
- - isProxy (page 2104)
- - performSelector: (page 2104)
- - performSelector:withObject: (page 2105)
- - performSelector:withObject:withObject: (page 2105)
- - release (page 2106)
- - respondsToSelector: (page 2107)
- - retain (page 2108)
- - retainCount (page 2109)
- - self (page 2109)
- - superclass (page 2110)
- - zone (page 2110)

Tasks

Initializing a Class

+ initialize (page 1158)
: Initializes the receiver before it's used (before it receives its first message).

+ load (page 1161)
: 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

+ new (page 1163)
: Allocates a new instance of the receiving class, sends it an init (page 1178) message, and returns the initialized object.

+ alloc (page 1152)
: Returns a new instance of the receiving class.

+ `allocWithZone:` (page 1152)
  Returns a new instance of the receiving class where memory for the new instance is allocated from a given zone.

- `init` (page 1178)
  Implemented by subclasses to initialize a new object (the receiver) immediately after memory for it has been allocated.

- `copy` (page 1172)
  Returns the object returned by `copyWithZone:` (page 2042), where the zone is `nil`.

+ `copyWithZone:` (page 1157)
  Returns the receiver.

- `mutableCopy` (page 1182)
  Returns the object returned by `mutableCopyWithZone:` (page 2094) where the zone is `nil`.

+ `mutableCopyWithZone:` (page 1162)
  Returns the receiver.

- `dealloc` (page 1174)
  Deallocates the memory occupied by the receiver.

- `finalize` (page 1176)
  The garbage collector invokes this method on the receiver before disposing of the memory it uses.

Identifying Classes

+ `class` (page 1155)
  Returns the class object.

+ `superclass` (page 1167)
  Returns the class object for the receiver's superclass.

+ `isSubclassOfClass:` (page 1161)
  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 1161)
  Returns a Boolean value that indicates whether instances of the receiver are capable of responding to a given selector.

Testing Protocol Conformance

+ `conformsToProtocol:` (page 1156)
  Returns a Boolean value that indicates whether the receiver conforms to a given protocol.

Obtaining Information About Methods

`- methodForSelector:` (page 1181)
Locates and returns the address of the receiver's implementation of a method so it can be called as a function.

`+ instanceMethodForSelector:` (page 1159)
Locates and returns the address of the implementation of the instance method identified by a given selector.

`+ instanceMethodSignatureForSelector:` (page 1160)
Returns an `NSMethodSignature` object that contains a description of the instance method identified by a given selector.

`- methodSignatureForSelector:` (page 1181)
Returns an `NSMethodSignature` object that contains a description of the method identified by a given selector.

Describing Objects

`+ description` (page 1157)
Returns a string that represents the contents of the receiving class.

Posing

`+ poseAsClass:` (page 1164) Deprecated in Mac OS X v10.5
Causes the receiving class to pose as a specified superclass.

Sending Messages

`- performSelector:withObject:afterDelay:` (page 1186)
Invokes a method of the receiver on the current thread using the default mode after a delay.

`- performSelector:withObject:afterDelay:inModes:` (page 1187)
Invokes a method of the receiver on the current thread using the specified modes after a delay.

`- performSelectorOnMainThread:withObject:waitUntilDone:` (page 1188)
Invokes a method of the receiver on the main thread using the default mode.

`- performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 1189)
Invokes a method of the receiver on the main thread using the specified modes.

`- performSelector:onThread:withObject:waitUntilDone:` (page 1183)
Invokes a method of the receiver on the specified thread using the default mode.

`- performSelector:onThread:withObject:waitUntilDone:modes:` (page 1184)
Invokes a method of the receiver on the specified thread using the specified modes.

`- performSelectorInBackground:withObject:` (page 1188)
Invokes a method of the receiver on a new background thread.

`+ cancelPreviousPerformRequestsWithTarget:` (page 1153)
Cancels perform requests previously registered with the `performSelector:withObject:afterDelay:` (page 1186) instance method.

`+ cancelPreviousPerformRequestsWithTarget:selector:object:` (page 1154)
Cancels perform requests previously registered with `performSelector:withObject:afterDelay:` (page 1186).

Forwarding Messages

`- forwardInvocation:` (page 1177)
Overridden by subclasses to forward messages to other objects.

Dynamically Resolving Methods

`+ resolveClassMethod:` (page 1165)
Dynamically provides an implementation for a given selector for a class method.

`+ resolveInstanceMethod:` (page 1165)
Dynamically provides an implementation for a given selector for an instance method.

Error Handling

`- doesNotRecognizeSelector:` (page 1175)
Handles messages the receiver doesn't recognize.

Archiving

`- awakeAfterUsingCoder:` (page 1169)
Overridden by subclasses to substitute another object in place of the object that was decoded and subsequently received this message.

`- classForArchiver` (page 1170)
Overridden by subclasses to substitute a class other than its own during archiving.

`- classForCoder` (page 1171)
Overridden by subclasses to substitute a class other than its own during coding.

`- classForKeyedArchiver` (page 1171)
Overridden by subclasses to substitute a new class for instances during keyed archiving.

`+ classFallbacksForKeyedArchiver` (page 1155)
Overridden to return the names of classes that can be used to decode objects if their class is unavailable.

`+ classForKeyedUnarchiver` (page 1156)
Overridden by subclasses to substitute a new class during keyed unarchiving.

`- classForPortCoder` (page 1171)
Overridden by subclasses to substitute a class other than its own for distribution encoding.

`- replacementObjectForArchiver:` (page 1190)
Overridden by subclasses to substitute another object for itself during archiving.

`- replacementObjectForCoder:` (page 1191)
Overridden by subclasses to substitute another object for itself during encoding.

- `replacementObjectForKeyedArchiver:` (page 1191)
  Overridden by subclasses to substitute another object for itself during keyed archiving.
- `replacementObjectForPortCoder:` (page 1192)
  Overridden by subclasses to substitute another object or a copy for itself during distribution encoding.
+ `setVersion:` (page 1166)
  Sets the receiver's version number.
+ `version` (page 1167)
  Returns the version number assigned to the class.

Working with Class Descriptions

- `attributeKeys` (page 1168)
  Returns an array of `NSString` objects containing the names of immutable values that instances of the receiver's class contain.
- `classDescription` (page 1170)
  Returns an object containing information about the attributes and relationships of the receiver's class.
- `inverseForRelationshipKey:` (page 1180)
  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 1194)
  Returns array containing the keys for the to-many relationship properties of the receiver.
- `toOneRelationshipKeys` (page 1195)
  Returns the keys for the to-one relationship properties of the receiver, if any.

Scripting

- `classCode` (page 1169)
  Returns the receiver's Apple event type code, as stored in the `NSScriptClassDescription` object for the object's class.
- `className` (page 1172)
  Returns a string containing the name of the class.
- `copyScriptingValue:forKey:withProperties:` (page 1173)
  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 1183)
  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 1193)
  Returns an `NSString`-keyed dictionary of the receiver's scriptable properties.
- `setScriptingProperties:` (page 1194)
  Given an `NSString`-keyed dictionary, sets one or more scriptable properties of the receiver.
- `scriptingValueForSpecifier:` (page 1193)
  Given an object specifier, returns the specified object or objects in the receiving container.

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`. The new instance is allocated from the default zone—use `allocWithZone:` (page 1152) to specify a particular zone.

An `init...` method must be used to complete the initialization process. For example:

```
TheClass *newObject = [[TheClass alloc] init];
```

Subclasses shouldn't override `alloc` to include initialization code. Instead, class-specific versions of `init...` methods should be implemented for that purpose. Class methods can also be implemented to combine allocation and initialization, similar to the `new` class method.

Special Considerations

If you are using managed memory (not garbage collection), this method retains the object before returning it. The returned object has a retain count of `1` and is *not* autoreleased. The invoker of this method is responsible for releasing the returned object, using either `release` (page 2106) or `autorelease` (page 2099).

Availability
Available in Mac OS X v10.0 and later.

See Also
`- init` (page 1178)

Related Sample Code
CoreRecipes
GLSLShowpiece
ImageClient
iSpend
QTCoreVideo301

Declared In
`NSObject.h`

allocWithZone:

Returns a new instance of the receiving class where memory for the new instance is allocated from a given zone.

```
+ (id)allocWithZone:(NSZone *)zone
```

Parameters

zone

The memory zone in which to create the new instance.

Return Value

A new instance of the receiver, where memory for the new instance is allocated from *zone*.

Discussion

The `isa` instance variable of the new instance is initialized to a data structure that describes the class; memory for its other instance variables is set to 0. If *zone* is `nil`, the new instance will be allocated from the default zone (as returned by `NSDefaultMallocZone`).

An `init...` method must be used to complete the initialization process. For example:

```
TheClass *newObject = [[TheClass allocWithZone:someZone] init];
```

Subclasses shouldn't override `allocWithZone:` to include any initialization code. Instead, class-specific versions of `init...` methods should be implemented for that purpose.

When one object creates another, it's sometimes a good idea to make sure they're both allocated from the same region of memory. The `zone` (page 2110) method (declared in the `NSObject` protocol) can be used for this purpose; it returns the zone where the receiver is located. For example:

```
id myCompanion = [[TheClass allocWithZone:[self zone]] init];
```

Special Considerations

If you are using managed memory (not garbage collection), this method retains the object before returning it. The returned object has a retain count of `1` and is *not* autoreleased. The invoker of this method is responsible for releasing the returned object, using either `release` (page 2106) or `autorelease` (page 2099).

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ alloc` (page 1152)

`- init` (page 1178)

Related Sample Code

MenuItemView

QTCoreVideo201

Quartz Composer WWDC 2005 TextEdit

Sketch-112

TextEditPlus

Declared In

`NSObject.h`

cancelPreviousPerformRequestsWithTarget:

Cancels perform requests previously registered with the `performSelector:withObject:afterDelay:` (page 1186) instance method.

```
+ (void)cancelPreviousPerformRequestsWithTarget:(id)aTarget
```

Parameters

aTarget

The target for requests previously registered with the `performSelector:withObject:afterDelay:` (page 1186) 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 Mac OS X v10.2 and later.

Declared In

`NSRunLoop.h`

cancelPreviousPerformRequestsWithTarget:selector:object:

Cancels perform requests previously registered with `performSelector:withObject:afterDelay:` (page 1186).

```
+ (void)cancelPreviousPerformRequestsWithTarget:(id)aTarget selector:(SEL)aSelector
    object:(id)anArgument
```

Parameters

aTarget

The target for requests previously registered with the `performSelector:withObject:afterDelay:` (page 1186) instance method

aSelector

The selector for requests previously registered with the `performSelector:withObject:afterDelay:` (page 1186) instance method.

See "Selectors" (page 1145) for a description of the `SEL` type.

anArgument

The argument for requests previously registered with the `performSelector:withObject:afterDelay:` (page 1186) instance method. Argument equality is determined using `isEqual:` (page 2101), 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 Mac 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:` (page 2102) method (declared in the `NSObject` protocol):

```
BOOL test = [self isKindOfClass:[SomeClass class]];
```

Availability
Available in Mac OS X v10.0 and later.

See Also
`class` (page 2100) (`NSObject` protocol)

Related Sample Code
NewsReader
OpenGLCaptureToMovie
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus

Declared In
`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`.

Developers who introduce a new class can use this method to provided 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 Mac 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 Mac 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 Mac OS X v10.0 and later.

See Also
`+ conformsToProtocol:` (page 1156)

Declared In
`NSObject.h`

copyWithZone:

Returns the receiver.

```
+ (id)copyWithZone:(NSZone *)zone
```

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 Mac OS X v10.0 and later.

See Also
`- copy` (page 1172)

Related Sample Code
AlbumToSlideshow
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`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 Mac OS X v10.0 and later.

See Also
`description` (page 2100) (`NSObject` protocol)

Related Sample Code
iSpend
QTKitMovieShuffler
QTRecorder
SimpleCalendar
StickiesExample

Declared In
`NSObject.h`

initialize

Initializes the receiver before it's used (before it receives its first message).

```
+ (void)initialize
```

Discussion
The runtime sends `initialize` to each class in a program exactly one time just before the class, or any class that inherits from it, is sent its first message from within the program. (Thus the method may never be invoked if the class is not used.) The runtime sends the `initialize` message to classes in a thread-safe manner. Superclasses receive this message before their subclasses.

For example, if the first message your program sends is this:

```
[NSApplication new]
```

the runtime system sends these three `initialize` messages:

```
[NSObject initialize];
[NSResponder initialize];
[NSApplication initialize];
```

because `NSApplication` is a subclass of `NSResponder` and `NSResponder` is a subclass of `NSObject`. All the `initialize` messages precede the `new` (page 1163) message.

If your program later begins to use the `NSText` class,

```
[NSText instancesRespondToSelector:someSelector]
```

the runtime system invokes these additional `initialize` messages:

```
[NSView initialize];
[NSText initialize];
```

because NSText inherits from NSObject, NSResponder, and NSView. The instancesRespondToSelector: (page 1161) message is sent only after all these classes are initialized. Note that the initialize messages to NSObject and NSResponder aren't repeated.

You implement initialize to provide class-specific initialization as needed. Since the runtime sends appropriate initialize messages automatically, you should typically not send initialize to super in your implementation.

If a particular class does not implement initialize, the initialize method of its superclass is invoked twice, once for the superclass and once for the non-implementing subclass. If you want to make sure that your class performs class-specific initializations only once, implement initialize as in the following example:

```
@implementation MyClass
+ (void)initialize
{
    if ( self == [MyClass class] ) {
        /* put initialization code here */
    }
}
```

Loading a subclasses of MyClass that does not implement its own initialize method will cause MyClass's implementation to be invoked. The test clause (if (self == [MyClass class])) ensures that the initialization code has no effect if initialize is invoked when a subclass is loaded.

Special Considerations

initialize it 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 1161) methods.

Availability
Available in Mac OS X v10.0 and later.

See Also
- init (page 1178)
+ load (page 1161)
class (page 2100) (NSObject protocol)

Related Sample Code
CoreRecipes
Dicey
iSpend
NewsReader
Reducer

Declared In
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:` (page 2107) method to check before passing the selector to `methodForSelector:`.

See "Selectors" (page 1145) for a description of the `SEL` type.

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 1181) instance method to the class instead.

See "Selectors" (page 1145) for a description of the `IMP` type, and how to invoke the returned method implementation.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`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.

See "Selectors" (page 1145) for a description of the `SEL` type.

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 Mac OS X v10.0 and later.

See Also
`- methodSignatureForSelector:` (page 1181)

Declared In
`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. See "Selectors" (page 1145) for a description of the `SEL` type.

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:` (page 2107).

Availability

Available in Mac OS X v10.0 and later.

See Also

`- forwardInvocation:` (page 1177)

Declared In

`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 Mac OS X v10.2 and later.

Declared In

`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.

On Mac OS X v10.5, 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 `+load` method, you can therefore safely message other unrelated classes from the same image, but any `+load` methods on those classes may not have run yet.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ initialize` (page 1158)

Related Sample Code
CIAnnotation
Core Data HTML Store
CustomAtomicStoreSubclass
LSMSmartCategorizer
TextLinks

Declared In
`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 Mac OS X v10.0 and later.

Declared In
`NSObject.h`

new

Allocates a new instance of the receiving class, sends it an `init` (page 1178) 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 1152) and `init` (page 1178). Like `alloc` (page 1152), it initializes the `isa` instance variable of the new object so it points to the class data structure. It then invokes the `init` (page 1178) method to complete the initialization process.

Unlike `alloc` (page 1152), `new` (page 1163) is sometimes re-implemented in subclasses to invoke a class-specific initialization method. If the `init...` method includes arguments, they're typically reflected in a `new...` method as well. For example:

```
+ newMyClassWithTag:(int)tag data:(struct info *)data
{
    return [[self alloc] initWithTag:tag data:data];
}
```

However, there's little point in implementing a `new...` method if it's simply a shorthand for `alloc` (page 1152) and `init...`, as shown above. Often `new...` methods will do more than just allocation and initialization. In some classes, they manage a set of instances, returning the one with the requested properties if it already exists, allocating and initializing a new instance only if necessary. For example:

```
+ newMyClassWithTag:(int)tag data:(struct info *)data
{
    MyClass *theInstance;

    if ( theInstance = findTheObjectWithTheTag(tag) )
        return [theInstance retain];
    return [[self alloc] initWithTag:tag data:data];
}
```

Although it's appropriate to define new `new...` methods in this way, the `alloc` (page 1152) and `allocWithZone:` (page 1152) methods should never be augmented to include initialization code.

Special Considerations

If you are using managed memory (not garbage collection), this method retains the object before returning it. The returned object has a retain count of `1` and is *not* autoreleased. The invoker of this method is responsible for releasing the returned object, using either `release` (page 2106) or `autorelease` (page 2099).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Fiendishthngs
LSMSmartCategorizer
NURBSSurfaceVertexProg
Quartz Composer QCTV
SurfaceVertexProgram

Declared In

`NSObject.h`

poseAsClass:

Causes the receiving class to pose as a specified superclass. (Deprecated in Mac 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 Mac OS X v10.5. The `poseAsClass:` method is not available in 64-bit applications on Mac OS X v10.5.

Availability

Available in Mac OS X v10.0.
Deprecated in Mac OS X v10.5.

Declared In

`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 provides an implementation for a given selector. See `resolveInstanceMethod:` (page 1165) for further discussion.

Availability

Available in Mac OS X v10.5 and later.

See Also

`+ resolveInstanceMethod:` (page 1165)

Declared In

`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 1165) 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 (see The Runtime System in *The Objective-C 2.0 Programming Language*) is invoked. If `respondsToSelector:` (page 2107) or `instancesRespondToSelector:` (page 1161) is invoked, the dynamic method resolver is given the opportunity to provide an `IMP` for the given selector first.

Availability
Available in Mac OS X v10.5 and later.

See Also
`+ resolveClassMethod:` (page 1165)

Declared In
`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 Mac OS X v10.0 and later.

See Also
`+ version` (page 1167)

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 Mac OS X v10.0 and later.

See Also
`+ class` (page 1155)
`superclass` (page 2110) (`NSObject` protocol)

Declared In
`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 Mac OS X v10.0 and later.

See Also
`+ setVersion:` (page 1166)
`versionForClassName:` (page 295) (`NSCoder`)

Related Sample Code
CoreRecipes
Fiendishthngs
PrefsPane

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 Mac OS X v10.0 and later.

See Also
`- classDescription` (page 1170)
`- inverseForRelationshipKey:` (page 1180)
`- toManyRelationshipKeys` (page 1194)
`- toOneRelationshipKeys` (page 1195)

Related Sample Code
Core Data HTML Store
CoreRecipes
StickiesExample

Declared In
`NSClassDescription.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

This method can be used 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. To prevent the accidental use of the receiver after its replacement has been returned, you should invoke the receiver's `release` method to release the object immediately.

This method is invoked by `NSCoder`. `NSObject`'s implementation simply returns `self`.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- classForCoder` (page 1171)
`- replacementObjectForCoder:` (page 1191)
`initWithCoder:` (page 2034) (`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 Mac OS X v10.0 and later.

Related Sample Code

Video Hardware Info

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 Mac OS X v10.0 and later.

See Also
- attributeKeys (page 1168)
- inverseForRelationshipKey: (page 1180)
- toManyRelationshipKeys (page 1194)
- toOneRelationshipKeys (page 1195)

Related Sample Code
SimpleScriptingObjects

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 1171). Override classForCoder (page 1171) to add general coding behavior.

Availability
Available in Mac OS X v10.0 and later.

See Also
- replacementObjectForArchiver: (page 1190)

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 Mac OS X v10.0 and later.

See Also
- awakeAfterUsingCoder: (page 1169)
- replacementObjectForCoder: (page 1191)

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 Mac OS X v10.2 and later.

See Also
- replacementObjectForKeyedArchiver: (page 1191)

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 1171) to add general coding behavior. This method is invoked by `NSPortCoder`. `NSObject`'s implementation returns the class returned by `classForCoder` (page 1171).

Availability
Available in Mac OS X v10.0 and later.

See Also
`- replacementObjectForPortCoder:` (page 1192)

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 Mac OS X v10.0 and later.

Related Sample Code
Sketch-112
StickiesExample

Declared In
`NSScriptClassDescription.h`

copy

Returns the object returned by `copyWithZone:` (page 2042), where the zone is `nil`.

```
- (id)copy
```

Return Value
The object returned by the `NSCopying` protocol method `copyWithZone:` (page 2042), where the zone is `nil`.

Discussion
This is a convenience method for classes that adopt the `NSCopying` protocol. An exception is raised if there is no implementation for `copyWithZone:` (page 2042).

`NSObject` does not itself support the `NSCopying` protocol. Subclasses must support the protocol and implement the `copyWithZone:` (page 2042) method. A subclass version of the `copyWithZone:` (page 2042) method should send the message to `super` first, to incorporate its implementation, unless the subclass descends directly from `NSObject`.

Special Considerations

If you are using managed memory (not garbage collection), this method retains the new object before returning it. The invoker of the method, however, is responsible for releasing the returned object.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AttachAScript
Dicey
iSpend
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`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:` (page 2118). 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 Mac 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 never send a `dealloc` message directly. Instead, an object's `dealloc` method is invoked indirectly through the `release` (page 2106) `NSObject` protocol method (if the `release` message results in the receiver's retain count becoming `0`). See *Memory Management Programming Guide for Cocoa* for more details on the use of these methods.

Subclasses must implement their own versions of `dealloc` to allow the release of any additional memory consumed by the object—such as dynamically allocated storage for data or object instance variables owned by the deallocated object. After performing the class-specific deallocation, the subclass method should incorporate superclass versions of `dealloc` through a message to `super`:

```
- (void)dealloc {
    [companion release];
    NSZoneFree(private, [self zone])
    [super dealloc];
}
```

Important: Note that when an application terminates, objects may not be sent a `dealloc` message since the process's memory is automatically cleared on exit—it is more efficient simply to allow the operating system to clean up resources than to invoke all the memory management methods. For this and other reasons, you should not manage scarce resources in `dealloc`—see Object Ownership and Disposal in *Memory Management Programming Guide for Cocoa* for more details.

Special Considerations

When garbage collection is enabled, the garbage collector sends `finalize` (page 1176) to the receiver instead of `dealloc`.

When garbage collection is enabled, this method is a no-op.

Availability
Available in Mac OS X v10.0 and later.

See Also
`autorelease` (page 2099) (`NSObject` protocol)
`release` (page 2106) (`NSObject` protocol)
`- finalize` (page 1176)

Related Sample Code
ImageClient
iSpend
QTCoreVideo301
Sketch-112
StickiesExample

Declared In
`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.
See "Selectors" (page 1145) for a description of the `SEL` type.

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 1172) or `init` (page 1178) 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:` (page 2107) 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` (page 2307) 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 Mac OS X v10.0 and later.

See Also
- `forwardInvocation:` (page 1177)

Declared In
`NSObject.h`

finalize

The garbage collector invokes this method on the receiver before disposing of the memory it uses.

```
- (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`.

Note: Garbage collection is not available for use in Mac OS X before version 10.5.

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 Mac OS X v10.4 and later.

See Also
- `dealloc` (page 1174)

Declared In
`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 1181) in addition to `forwardInvocation:`. The mechanism for forwarding messages uses information obtained from `methodSignatureForSelector:` (page 1181) to create the `NSInvocation` object to be forwarded. Your overriding method must provide an appropriate method signature for the given selector, either by preformulating 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
        [self doesNotRecognizeSelector:aSelector];
}
```

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 1175) 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 Mac OS X v10.0 and later.

Declared In
`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
The initialized receiver.

Discussion
An `init` message is generally coupled with an `alloc` (page 1152) or `allocWithZone:` (page 1152) 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`.

Subclass implementations of this method should initialize and return the new object. If it can't be initialized, they should release the object and return `nil`. In some cases, an `init` method might release the new object and return a substitute. Programs should therefore always use the object returned by `init`, and not necessarily the one returned by `alloc` (page 1152) or `allocWithZone:` (page 1152), in subsequent code.

Every class must guarantee that the `init` method either returns a fully functional instance of the class or raises an exception. Subclasses should override the `init` method to add class-specific initialization code. Subclass versions of `init` need to incorporate the initialization code for the classes they inherit from, through a message to `super`:

```
- (id)init
{
    if ((self = [super init])) {
        /* class-specific initialization goes here */
    }
    return self;
}
```

Note that the message to `super` precedes the initialization code added in the method. This sequencing ensures that initialization proceeds in the order of inheritance.

Subclasses often define `init...` methods with additional arguments to allow specific values to be set. The more arguments a method has, the more freedom it gives you to determine the character of initialized objects. Classes often have a set of `init...` methods, each with a different number of arguments. For example:

```
- (id)init;
- (id)initWithTag:(int)tag;
- (id)initWithTag:(int)tag data:(struct info *)data;
```

The convention is that at least one of these methods, usually the one with the most arguments, includes a message to `super` to incorporate the initialization of classes higher up the hierarchy. This method is called the *designated initializer* for the class. The other `init...` methods defined in the class directly or indirectly invoke the designated initializer through messages to `self`. In this way, all `init...` methods are chained together. For example:

```
- (id)init
{
    return [self initWithTag:-1];
}

- (id)initWithTag:(int)tag
{
    return [self initWithTag:tag data:NULL];
}

- (id)initWithTag:(int)tag data:(struct info *)data
{
    if ((self = [super init. . .])) {
        /* class-specific initialization goes here */
    }
    return self;
}
```

In this example, the `initWithTag:data:` method is the designated initializer for the class.

If a subclass does any initialization of its own, it must define its own designated initializer. This method should begin by sending a message to `super` to invoke the designated initializer of its superclass. Suppose, for example, that the three methods illustrated above are defined in the B class. The C class, a subclass of B, might have this designated initializer:

```
- (id)initWithTag:(int)tag data:(struct info *)data object:anObject
{
    if ((self = [super initWithTag:tag data:data])) {
        /* class-specific initialization goes here */
    }
    return self;
}
```

If inherited `init...` methods are to successfully initialize instances of the subclass, they must all be made to (directly or indirectly) invoke the new designated initializer. To accomplish this, the subclass is obliged to cover (override) only the designated initializer of the superclass. For example, in addition to its designated initializer, the C class would also implement this method:

```
- (id)initWithTag:(int)tag data:(struct info *)data
{
    return [self initWithTag:tag data:data object:nil];
}
```

This code ensures that all three methods inherited from the B class also work for instances of the C class.

Often the designated initializer of the subclass overrides the designated initializer of the superclass. If so, the subclass need only implement the one `init...` method.

These conventions maintain a direct chain of `init...` links and ensure that the `new` method and all inherited `init...` methods return usable, initialized objects. They also prevent the possibility of an infinite loop wherein a subclass method sends a message (to `super`) to perform a superclass method, which in turn sends a message (to `self`) to perform the subclass method.

This `init` method is the designated initializer for the `NSObject` class. Subclasses that do their own initialization should override it, as described above.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ImageClient
Quartz Composer WWDC 2005 TextEdit
Quartz EB
StickiesExample
TextEditPlus

Declared In
`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 Mac OS X v10.0 and later.

See Also
- `attributeKeys` (page 1168)
- `classDescription` (page 1170)
- `toManyRelationshipKeys` (page 1194)
- `toOneRelationshipKeys` (page 1195)

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:` (page 2107) 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.

See "Selectors" (page 1145) for a description of the `IMP` and `SEL` types, and how to invoke the returned method implementation.

Availability
Available in Mac OS X v10.0 and later.

See Also
+ `instanceMethodForSelector:` (page 1159)

Declared In
`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.

See "Selectors" (page 1145) for a description of the SEL type.

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 Mac OS X v10.0 and later.

See Also

+ instanceMethodSignatureForSelector: (page 1160)

- forwardInvocation: (page 1177)

Declared In

NSObject.h

mutableCopy

Returns the object returned by mutableCopyWithZone: (page 2094) where the zone is nil.

```
- (id)mutableCopy
```

Return Value

The object returned by the NSMutableCopying protocol method mutableCopyWithZone: (page 2094), 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: (page 2094).

Special Considerations

If you are using managed memory (not garbage collection), this method retains the new object before returning it. The invoker of the method, however, is responsible for releasing the returned object.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CoreRecipes

iSpend

NewsReader

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In
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:` (page 2118). 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 Mac 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 *)thr 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.

See "Selectors" (page 1145) for a description of the `SEL` type.

thr

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 1186) or `performSelector:withObject:afterDelay:inModes:` (page 1187) method.

This method retains the receiver and the *arg* parameter until after the selector is performed.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `performSelector:onThread:withObject:waitUntilDone:modes:` (page 1184)
- `performSelectorInBackground:withObject:` (page 1188)

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 *)thr 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.

See "Selectors" (page 1145) for a description of the `SEL` type.

thr

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.

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 1186) or `performSelector:withObject:afterDelay:inModes:` (page 1187) method instead.

This method retains the receiver and the *arg* parameter until after the selector is performed.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `performSelector:onThread:withObject:waitUntilDone:` (page 1183)
- `performSelectorInBackground:withObject:` (page 1188)

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.

See "Selectors" (page 1145) for a description of the `SEL` type.

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 1187) method instead. To ensure that the selector is performed on the main thread, use the `performSelectorOnMainThread:withObject:waitUntilDone:` (page 1188) or `performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 1189) method instead. To cancel a queued message, use the cancelPreviousPerformRequestsWithTarget: (page 1153) or cancelPreviousPerformRequestsWithTarget:selector:object: (page 1154) method.

This method retains the receiver and the *anArgument* parameter until after the selector is performed.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ cancelPreviousPerformRequestsWithTarget:selector:object:` (page 1154)
`- performSelectorOnMainThread:withObject:waitUntilDone:` (page 1188)
`- performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 1189)
`- performSelector:onThread:withObject:waitUntilDone:modes:` (page 1184)

Related Sample Code

IdentitySample

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.

See "Selectors" (page 1145) for a description of the `SEL` type.

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.

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 1187) method instead. To ensure that the selector is performed on the main thread, use the `performSelectorOnMainThread:withObject:waitUntilDone:` (page 1188) or `performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 1189) method instead. To cancel a queued message, use the cancelPreviousPerformRequestsWithTarget: (page 1153) or cancelPreviousPerformRequestsWithTarget:selector:object: (page 1154) method.

This method retains the receiver and the *anArgument* parameter until after the selector is performed.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `performSelector:withObject:afterDelay:` (page 1186)
- `performSelectorOnMainThread:withObject:waitUntilDone:` (page 1188)
- `performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 1189)
- `performSelector:onThread:withObject:waitUntilDone:modes:` (page 1184)

`addTimer:forMode:` (page 1333) (`NSRunLoop`)

`invalidate` (page 1660) (`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.

See "Selectors" (page 1145) for a description of the `SEL` type.

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*.

This method retains the receiver and the *arg* parameter until after the selector is performed.

Availability

Available in Mac OS X v10.5 and later.

See Also

- performSelector:onThread:withObject:waitUntilDone:modes: (page 1184)

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.

See "Selectors" (page 1145) for a description of the `SEL` type.

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 default 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 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 1186) or `performSelector:withObject:afterDelay:inModes:` (page 1187) method.

This method retains the receiver and the *arg* parameter until after the selector is performed.

Availability
Available in Mac OS X v10.2 and later.

See Also
- `performSelector:withObject:afterDelay:` (page 1186)
- `performSelector:withObject:afterDelay:inModes:` (page 1187)
- `performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 1189)
- `performSelector:onThread:withObject:waitUntilDone:modes:` (page 1184)

Related Sample Code
AudioDeviceNotify
CocoaDVDPlayer
ExtractMovieAudioToAIFF
HelpHook
JSheets

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.

See "Selectors" (page 1145) for a description of the `SEL` type.

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.

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.

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 1186) or `performSelector:withObject:afterDelay:inModes:` (page 1187) method.

This method retains the receiver and the *arg* parameter until after the selector is performed.

Availability

Available in Mac OS X v10.2 and later.

See Also

- `performSelector:withObject:afterDelay:` (page 1186)
- `performSelector:withObject:afterDelay:inModes:` (page 1187)
- `performSelectorOnMainThread:withObject:waitUntilDone:` (page 1188)
- `performSelector:onThread:withObject:waitUntilDone:modes:` (page 1184)

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 1191).

Availability

Available in Mac OS X v10.0 and later.

See Also

`- classForArchiver` (page 1170)

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 Mac OS X v10.0 and later.

See Also

`- classForCoder` (page 1171)

`- awakeAfterUsingCoder:` (page 1169)

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 Mac OS X v10.2 and later.

See Also

`- classForKeyedArchiver` (page 1171)

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 1191), enabling all objects to be distributed by proxy as the default. However, if `replacementObjectForCoder:` (page 1191) 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 {
    if ([encoder isByref])
        return [NSDistantObject proxyWithLocal:self
connection:[encoder connection]];
    else
        return self;
}
```

Availability

Available in Mac OS X v10.0 and later.

See Also

`- classForPortCoder` (page 1171)

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 Mac OS X v10.2 and later.

See Also
- setScriptingProperties: (page 1194)

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: (page 2123) 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 Mac 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 Mac OS X v10.2 and later.

See Also
- scriptingProperties (page 1193)

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 Mac OS X v10.0 and later.

See Also
- `attributeKeys` (page 1168)
- `classDescription` (page 1170)
- `inverseForRelationshipKey:` (page 1180)
- `toOneRelationshipKeys` (page 1195)

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 Mac OS X v10.0 and later.

See Also
- `attributeKeys` (page 1168)
- `classDescription` (page 1170)
- `toManyRelationshipKeys` (page 1194)
- `inverseForRelationshipKey:` (page 1180)

Declared In
`NSClassDescription.h`

NSOperation Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	Foundation/NSOperation.h
Companion guide	Threading Programming Guide
Related sample code	NSOperationSample

Overview

The `NSOperation` class manages the execution of a single encapsulated task. Operations are typically scheduled by adding them to an operation queue object (an instance of the `NSOperationQueue` class), although you can also execute them directly by explicitly invoking their `start` method.

Operation objects are single-shot objects, that is, they perform their task once. You cannot reuse the same `NSOperation` object to perform a task (or a slight variant of the task) multiple times in succession. Attempting to execute an operation that has already finished results in an exception.

When manually executing operations, you are responsible for making sure the object is ready to execute. Starting an operation that is not in the ready state generally results in an exception being thrown. If you use an operation queue to manage the execution, the `NSOperationQueue` object ensures that the operation is executed only when it is ready.

Concurrent Versus Non-Concurrent Operations

Operation objects can be designed for either concurrent or non-concurrent operation. In the context of an `NSOperation` object, the terms concurrent and non-concurrent do not necessarily refer to the side-by-side execution of threads. Instead, a non-concurrent operation is one that executes using the environment that is provided for it while a concurrent operation is responsible for setting up its own execution environment. To understand how this might work in your code, look at the `NSOperationQueue` object as an example. For a non-concurrent operation, an operation queue automatically creates a thread and calls the operation object's `start` method, the default implementation of which configures the thread environment and calls the operation object's `main` method to run your custom code. For a concurrent operation, the queue simply calls the object's `start` method on the current thread. The operation object is then responsible for setting up the appropriate execution environment, which could include starting a new thread.

If you always design your operations to execute on a thread, then creating non-concurrent operations is the simplest way to go. There are some situations though where you might want to create a concurrent operation instead, including the following:

- You want to create the thread yourself.
- You want to launch a separate task instead of a thread.
- Your operation's main method initiates an asynchronous call and exits. (In such a situation, the callback function or method would then pass control to the operation object to process the request. For example, you could use this technique to set up a timer and then use the methods of the operation object to do some work each time the timer fires.)

By default, operations are designated as non-concurrent. For information on how to create a concurrent operation object, see the subclassing notes for this class.

Operation Dependencies

You can configure an operation to depend on the completion of other operations by adding those operations as dependencies. An operation object that has dependencies does not execute until all of its dependent operation objects finish executing. Once the last dependent operation finishes, the operation object moves to the ready state.

If a dependent operation is unable to perform its task for some reason, it is the responsibility of your code to make that determination. Operation objects that are non-concurrent (that is, their `isConcurrent` method returns `NO`) automatically catch and suppress any exceptions thrown by the operation object's `main` method. Thus, an operation that generates an exception may appear to finish normally even if it did not. If you need to track errors in a dependent operation, you must build that capability into the `main` method of your operation objects explicitly.

KVO-Compliant Properties

The `NSOperation` class is key-value coding (KVC) and key-value observing (KVO) compliant for several of its properties. As needed, you can observe these properties to control other parts of your application. The properties you can observe include the following:

- `isCancelled` - read-only property
- `isConcurrent` - read-only property
- `isExecuting` - read-only property
- `isFinished` - read-only property
- `isReady` - read-only property
- `dependencies` - read-only property
- `queuePriority` - readable and writable property

Although you can attach observers to these properties, you should not use Cocoa bindings to bind them to elements of your application's user interface. Code associated with your user interface typically must execute only in your application's main thread. Because an operation may execute in any thread, any KVO notifications associated with that operation may similarly occur in any thread.

If you override any of the preceding properties, your implementations must maintain KVC and KVO compliance. If you define additional properties for your `NSOperation` objects, it is recommended that you make those properties KVC and KVO compliant as well. For information on how to support key-value coding, see *Key-Value Coding Programming Guide*. For information on how to support key-value observing, see *Key-Value Observing Programming Guide*.

Threading Considerations

The methods of the `NSOperation` class implement automatic synchronization on the current instance. It is therefore safe to use a single instance of the `NSOperation` object from multiple threads without creating additional locks to synchronize access to the object.

When you subclass `NSOperation`, the methods in your implementation should also be safe to call from multiple threads. For example, if the methods of your operation object access shared resources, they should take the appropriate locks to synchronize access to those resources. For more information about writing thread-safe code, see *Threading Programming Guide*.

Subclassing Notes

The `NSOperation` class does not do anything by default and must be subclassed to perform any desired tasks. How you create your subclass depends on whether your operation is designed to execute concurrently or non-concurrently with respect to the thread that started the operation.

Methods to Override

For non-concurrent operations, you typically implement only one method:

- `main`

In your `main` method, you implement the code needed to perform the given operation. The `NSOperation` class manages the changes in state for your operation automatically and reports the appropriate condition of your operation from its methods.

If you are creating a concurrent operation, you need to override the following methods:

- `start`
- `isConcurrent`
- `isExecuting`
- `isFinished`

In your `start` method, you must prepare the operation for execution, which includes preparing the runtime environment for your operation. (For example, if you wanted to create a thread yourself, you would do it here.) Once your runtime environment is established, you can call any methods or functions you want to subsequently start your operation. Your implementation of the `start` method should not invoke `super`.

When implementing a concurrent operation, your custom subclass is responsible for reporting some of the state information associated with running the operation. In particular, you must override the `isExecuting` and `isFinished` methods to report on the current execution state of your operation. These methods must return accurate values for the state of your operation at all times, including when your operation has been cancelled. Your overridden methods should be KVO compliant.

Responding to the Cancel Command

An operation is responsible for periodically calling its own `isCancelled` method and aborting execution if it ever returns `YES`. Because it is bad form to kill a thread outright, the `NSOperationQueue` object sends a `cancel` message to your operation object if it ever needs your object to stop executing. (Other entities can similarly call the `cancel` method on an executing operation to ask it to stop.) The need to cancel an operation can typically arise from a user request or in a situation where the application or system is shutting down. When detected, your operation should clean up its environment and exit as soon as possible.

If an operation is cancelled, it should still update its internal state variables to reflect the change in execution status. In particular, the object's `isFinished` method should return `YES` and its `isExecuting` method should return `NO`. It must do this even if the it was cancelled before it started executing.

Note: If you implement a custom operation object as a concurrent operation, the `start` method can still be called even if the operation has already been cancelled. Your startup code should be prepared to handle this situation and clean up appropriately.

Tasks

Initialization

- `init` (page 1203)
  Returns an initialized `NSOperation` object.

Executing the Operation

- `start` (page 1207)
  Begins the execution of the operation.
- `main` (page 1205)
  Performs the receiver's non-concurrent task.

Canceling Operations

- `cancel` (page 1202)
  Advises the operation object that it should stop executing its task.

Getting the Operation Status

- `isCancelled` (page 1203)
  Returns a Boolean value indicating whether the operation has been cancelled.
- `isExecuting` (page 1204)
  Returns a Boolean value indicating whether the operation is currently executing.
- `isFinished` (page 1204)
  Returns a Boolean value indicating whether the operation is done executing.
- `isConcurrent` (page 1204)
  Returns a Boolean value indicating whether the operation runs asynchronously.
- `isReady` (page 1205)
  Returns a Boolean value indicating whether the receiver's operation can be performed now.

Managing Dependencies

- `addDependency:` (page 1201)
  Makes the receiver dependent on the completion of the specified operation.
- `removeDependency:` (page 1206)
  Removes the receiver's dependence on the specified operation.
- `dependencies` (page 1202)
  Returns a new array object containing the operations on which the receiver is dependent.

Prioritizing Operations in an Operation Queue

- `queuePriority` (page 1206)
  Returns the priority of the operation in an operation queue.
- `setQueuePriority:` (page 1207)
  Sets the priority of the operation when used in an operation queue.

Instance Methods

addDependency:

Makes the receiver dependent on the completion of the specified operation.

```
- (void)addDependency:(NSOperation *)operation
```

Parameters

operation
  The operation on which the receiver is dependent. The same dependency should not be added more than once to the receiver, and the results of doing so are undefined.

Discussion
The receiver is not considered ready to execute until all of its dependent operations finish executing. If the receiver is already executing its task, adding dependencies is unlikely to have any practical effect. This method may change the `isReady` and `dependencies` properties of the receiver.

It is a programmer error to create any circular dependencies among a set of operations. Doing so can cause a deadlock among the operations and may freeze your program.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `removeDependency:` (page 1206)
- `dependencies` (page 1202)

Declared In
`NSOperation.h`

cancel

Advises the operation object that it should stop executing its task.

```
- (void)cancel
```

Discussion
This method does not force your operation code to stop. The code for your operation must invoke the `isCancelled` method periodically to determine whether the operation should be stopped. Once cancelled, an operation cannot be restarted.

If the operation is already finished executing, this method has no effect. Canceling an operation that is currently in an operation queue, but not yet executing, causes it to be removed from the queue (although not necessarily right away).

Availability
Available in Mac OS X v10.5 and later.

See Also
- `isCancelled` (page 1203)

Declared In
`NSOperation.h`

dependencies

Returns a new array object containing the operations on which the receiver is dependent.

```
- (NSArray *)dependencies
```

Return Value
A new array object containing the `NSOperation` objects.

Discussion
The receiver is not considered ready to execute until all of its dependent operations finish executing.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `addDependency:` (page 1201)
- `removeDependency:` (page 1206)

Declared In
`NSOperation.h`

init

Returns an initialized `NSOperation` object.

```
- (id)init
```

Return Value
The initialized `NSOperation` object.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSOperation.h`

isCancelled

Returns a Boolean value indicating whether the operation has been cancelled.

```
- (BOOL)isCancelled
```

Return Value
`YES` if the operation was explicitly cancelled by an invocation of the receiver's `cancel` method; otherwise, `NO`. This method may return `YES` even if the operation is currently executing.

Discussion
Canceling an operation does not actively stop the receiver's code from executing. An operation object is responsible for calling this method periodically and stopping itself if the method returns `YES`.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `cancel` (page 1202)

Related Sample Code
NSOperationSample

Declared In
`NSOperation.h`

isConcurrent

Returns a Boolean value indicating whether the operation runs asynchronously.

```
- (BOOL)isConcurrent
```

Return Value
`YES` if the operation is asynchronous; otherwise, `NO` if the operation runs synchronously on whatever thread started it. This method returns `NO` by default.

Discussion
If you are implementing a concurrent operation, you must override this method and return `YES` from your implementation. For more information about the differences between concurrent and non-concurrent operations, see "Concurrent Versus Non-Concurrent Operations" (page 1197).

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSOperation.h`

isExecuting

Returns a Boolean value indicating whether the operation is currently executing.

```
- (BOOL)isExecuting
```

Return Value
`YES` if the operation is executing; otherwise, `NO` if the operation has not been started or is already finished.

Discussion
If you are implementing a concurrent operation, you should override this method to return the execution state of your operation. Concurrent operations are also responsible for generating the appropriate KVO notifications whenever the execution state changes. For more information about manually generating KVO notifications, see *Key-Value Observing Programming Guide*.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSOperation.h`

isFinished

Returns a Boolean value indicating whether the operation is done executing.

```
- (BOOL)isFinished
```

Return Value
`YES` if the operation is no longer executing; otherwise, `NO`.

Discussion
If you are implementing a concurrent operation, you should override this method to return the finished state of your operation. Concurrent operations are also responsible for generating the appropriate KVO notifications whenever the finished state changes. For more information about manually generating KVO notifications, see *Key-Value Observing Programming Guide*.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSOperation.h`

isReady

Returns a Boolean value indicating whether the receiver's operation can be performed now.

```
- (BOOL)isReady
```

Return Value
YES if the operation can be performed now; otherwise, NO.

Discussion
Operations may not be ready due to dependencies on other operations or because of external conditions that might prevent needed data from being ready. The `NSOperation` class manages dependencies on other operations and reports the readiness of the receiver based on those dependencies.

Note: If the receiver is cancelled before it starts, operations that are dependent on the completion of the receiver will never become ready.

If your operation object has additional dependencies, you must override this method and return a value that accurately reflects the readiness of the receiver. Your custom implementation should invoke `super` and incorporate its return value into this method's return value. Your custom implementation must be KVO compliant.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- dependencies` (page 1202)

Declared In
`NSOperation.h`

main

Performs the receiver's non-concurrent task.

```
- (void)main
```

Discussion
The default implementation of this method does nothing. For non-concurrent operations, you must override this method in your `NSOperation` subclass to perform the desired task. In your implementation, do not invoke `super`.

If you are implementing a concurrent operation, you should override the `start` method instead. In your overridden `start` method, you can continue to call this method to do the actual work if separating the work from your starting logic is practical.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- start` (page 1207)

Declared In
`NSOperation.h`

queuePriority

Returns the priority of the operation in an operation queue.

```
- (NSOperationQueuePriority)queuePriority
```

Return Value
The relative priority of the operation. The returned value always corresponds to one of the predefined constants. (For a list of valid values, see "Operation Priorities" (page 1208).) If no priority is explicitly set, this method returns `NSOperationQueuePriorityNormal`.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- setQueuePriority:` (page 1207)

Declared In
`NSOperation.h`

removeDependency:

Removes the receiver's dependence on the specified operation.

```
- (void)removeDependency:(NSOperation *)operation
```

Parameters
operation
The dependent operation to be removed from the receiver.

Discussion
This method may change the `isReady` and `dependencies` properties of the receiver.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `addDependency:` (page 1201)
- `dependencies` (page 1202)

Declared In
`NSOperation.h`

setQueuePriority:

Sets the priority of the operation when used in an operation queue.

```
- (void)setQueuePriority:(NSOperationQueuePriority)priority
```

Parameters

priority

The relative priority of the operation. For a list of valid values, see "Operation Priorities" (page 1208).

Discussion
You should use priority values only as needed to classify the relative priority of non-dependent operations. Priority values should not be used to implement dependency management among different operation objects. If you need to establish dependencies between operations, use the `addDependency:` method instead.

If you attempt to specify a priority value that does not match one of the defined constants, this method automatically adjusts the value you specify towards the `NSOperationQueuePriorityNormal` priority, stopping at the first valid constant value. For example, if you specified the value -10, this method would adjust that value to match the `NSOperationQueuePriorityVeryLow` constant. Similarly, if you specified +10, this method would adjust the value to match the `NSOperationQueuePriorityVeryHigh` constant.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `queuePriority` (page 1206)
- `addDependency:` (page 1201)

Related Sample Code
NSOperationSample

Declared In
`NSOperation.h`

start

Begins the execution of the operation.

```
- (void)start
```

Discussion
The default implementation of this method configures the execution environment for a non-concurrent operation and invokes the receiver's main method. As part of the default configuration, this method performs several checks to ensure that the non-concurrent operation can actually run and generates appropriate KVO notifications for each change in the operation's state. If the receiver's operation has already been performed,

this method throws an `NSInvalidArgumentException` exception. If the operation has already been cancelled, this method simply returns without calling main. If the operation is to be performed on a separate thread, this method may return before the operation itself completes on the other thread.

Note: An operation may not be ready to execute if it is dependent on other operations that have not yet finished.

If you are implementing a concurrent operation, you must override this method to initiate your operation; however, your implementation must not call `super`. If you override this method, you must also override the `isExecuting` and `isFinished` methods to report when your operation begins executing and finishes. Your implementations for these methods must maintain KVO compliance for the associated properties by manually sending the appropriate value change messages. For more information about manually generating KVO notifications, see *Key-Value Observing Programming Guide*.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `main` (page 1205)
- `isReady` (page 1205)
- `dependencies` (page 1202)

Declared In
`NSOperation.h`

Constants

NSOperationQueuePriority

Describes the priority of an operation relative to other operations in an operation queue.

```
typedef NSInteger NSOperationQueuePriority;
```

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSOperation.h`

Operation Priorities

These constants let you prioritize the order in which operations execute.

```
enum {
    NSOperationQueuePriorityVeryLow = -8,
    NSOperationQueuePriorityLow = -4,
    NSOperationQueuePriorityNormal = 0,
    NSOperationQueuePriorityHigh = 4,
    NSOperationQueuePriorityVeryHigh = 8
};
```

Constants

`NSOperationQueuePriorityVeryLow`
Operations receive very low priority for execution.
Available in Mac OS X v10.5 and later.
Declared in `NSOperation.h`.

`NSOperationQueuePriorityLow`
Operations receive low priority for execution.
Available in Mac OS X v10.5 and later.
Declared in `NSOperation.h`.

`NSOperationQueuePriorityNormal`
Operations receive the normal priority for execution.
Available in Mac OS X v10.5 and later.
Declared in `NSOperation.h`.

`NSOperationQueuePriorityHigh`
Operations receive high priority for execution.
Available in Mac OS X v10.5 and later.
Declared in `NSOperation.h`.

`NSOperationQueuePriorityVeryHigh`
Operations receive very high priority for execution.
Available in Mac OS X v10.5 and later.
Declared in `NSOperation.h`.

Discussion

You can use these constants to specify the relative ordering of operations that are waiting to be started in an operation queue. You should always use these constants (and not the defined value) for determining priority.

Declared In

`NSOperation.h`

NSOperationQueue Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	Foundation/NSOperation.h
Companion guide	Threading Programming Guide
Related sample code	NSOperationSample

Overview

The `NSOperationQueue` class manages a set of `NSOperation` objects in a priority queue and regulates their execution. Operations remain in the queue until they are explicitly cancelled or finish executing. An application may create multiple operation queues, with each queue running up to its designated maximum number of operations.

A specific `NSOperation` object can be in only one operation queue at a time. Operations within a single queue coordinate their execution order using both priority levels and inter-operation object dependencies. Operation objects in different queues can coordinate their execution order using dependencies, which are not queue-specific.

Inter-operation dependencies provide an absolute execution order for operations. An operation object is not considered ready to execute until all of its dependent operations have finished executing. For operations that are ready to execute, the operation queue always executes the one with the highest priority relative to the other ready operations. For details on how to set priority levels and dependencies, see *NSOperation Class Reference*.

You should never manually start an operation while it is sitting in an operation queue. Once added, an operation stays in its queue until it finishes executing or is cancelled.

If the `isConcurrent` method of an operation returns `NO`, the operation queue automatically creates a new thread for that operation before running it. If the `isConcurrent` method returns `YES`, the operation object must create its own thread or otherwise configure its own runtime environment as part of its execution phase.

KVO-Compliant Properties

The `NSOperationQueue` class is key-value coding (KVC) and key-value observing (KVO) compliant. You can observe these properties as desired to control other parts of your application. The properties you can observe include the following:

- `operations` - read-only property
- `maxConcurrentOperationCount` - readable and writable property

For more information about key-value observing and how to attach observers to an object, see *Key-Value Observing Programming Guide*.

Threading Considerations

It is safe to use a single `NSOperationQueue` object from multiple threads without creating additional locks to synchronize access to that object.

Tasks

Managing Operations in the Queue

- `addOperation:` (page 1213)
  Adds the specified operation object to the receiver.
- `operations` (page 1214)
  Returns a new array containing the operations currently in the queue.
- `cancelAllOperations` (page 1213)
  Cancels all queued and executing operations.
- `waitUntilAllOperationsAreFinished` (page 1216)
  Blocks the current thread until all of the receiver's queued and executing operations finish executing.

Managing the Number of Running Operations

- `maxConcurrentOperationCount` (page 1214)
  Returns the maximum number of concurrent operations that the receiver can execute.
- `setMaxConcurrentOperationCount:` (page 1215)
  Sets the maximum number of concurrent operations that the receiver can execute.

Suspending Operations

- `setSuspended:` (page 1215)
  Modifies the execution of pending operations

- isSuspended (page 1214)
  Returns a Boolean value indicating whether the receiver is scheduling queued operations for execution.

Instance Methods

addOperation:

Adds the specified operation object to the receiver.

```
- (void)addOperation:(NSOperation *)operation
```

Parameters

operation
  The operation object to be added to the queue. In memory-managed applications, this object is retained by the operation queue.

Discussion
An operation object can be in at most one operation queue at a time and cannot be added if it is currently executing or finished. This method throws an `NSInvalidArgumentException` exception if any of these conditions is true.

Once added, the specified *operation* remains in the queue until it is executed or cancelled.

Availability
Available in Mac OS X v10.5 and later.

See Also
- cancel (page 1202) (NSOperation)
- isExecuting (page 1204) (NSOperation)

Declared In
NSOperation.h

cancelAllOperations

Cancels all queued and executing operations.

```
- (void)cancelAllOperations
```

Discussion
This method sends a `cancel` message to all operations currently in the queue or executing. Queued operations are cancelled before they begin executing. If an operation is already executing, it is up to that operation to recognize the cancellation and stop what it is doing.

Availability
Available in Mac OS X v10.5 and later.

See Also
cancel (page 1202) (NSOperation)

Declared In
NSOperation.h

isSuspended

Returns a Boolean value indicating whether the receiver is scheduling queued operations for execution.

```
- (BOOL)isSuspended
```

Return Value
NO if operations are being scheduled for execution; otherwise, YES.

Availability
Available in Mac OS X v10.5 and later.

See Also
- setSuspended: (page 1215)

Declared In
NSOperation.h

maxConcurrentOperationCount

Returns the maximum number of concurrent operations that the receiver can execute.

```
- (NSInteger)maxConcurrentOperationCount
```

Return Value
The maximum number of concurrent operations set explicitly on the receiver using the setMaxConcurrentOperationCount: method.

Availability
Available in Mac OS X v10.5 and later.

See Also
- setMaxConcurrentOperationCount: (page 1215)

Declared In
NSOperation.h

operations

Returns a new array containing the operations currently in the queue.

```
- (NSArray *)operations
```

Return Value
A new array object containing the NSOperation objects in the order in which they were added to the queue.

Availability
Available in Mac OS X v10.5 and later.

Declared In
NSOperation.h

setMaxConcurrentOperationCount:

Sets the maximum number of concurrent operations that the receiver can execute.

```
- (void)setMaxConcurrentOperationCount:(NSInteger)count
```

Parameters

count

The maximum number of concurrent operations. Specify the value `NSOperationQueueDefaultMaxConcurrentOperationCount` if you want the receiver to choose an appropriate value based on the number of available processors and other relevant factors.

Discussion
The specified value affects only the receiver and the operations in its queue. Other operation queue objects can also execute their maximum number of operations in parallel.

Reducing the number of concurrent operations does not affect any operations that are currently executing. If you specify the value `NSOperationQueueDefaultMaxConcurrentOperationCount` (which is recommended), the maximum number of operations can change dynamically based on system conditions.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `maxConcurrentOperationCount` (page 1214)

Declared In
NSOperation.h

setSuspended:

Modifies the execution of pending operations

```
- (void)setSuspended:(BOOL)suspend
```

Parameters

suspend

If `YES`, the queue stops scheduling queued operations for execution. If `NO`, the queue begins scheduling operations again.

Discussion
This method suspends or restarts the execution of queued operations only. It does not have any impact on the state of currently running operations. Running operations continue to run until their natural termination or until they are explicitly cancelled.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `isSuspended` (page 1214)

Declared In
NSOperation.h

waitUntilAllOperationsAreFinished

Blocks the current thread until all of the receiver's queued and executing operations finish executing.

```
- (void)waitUntilAllOperationsAreFinished
```

Discussion
When called, this method blocks the current thread and waits for the receiver's current and pending operations to finish executing. While the thread is blocked, the receiver continues to launch already queued operations and monitor those that are executing. During this time, the current thread cannot add operations to the queue, but other threads may. Once all of the pending operations are finished, this method returns.

Availability
Available in Mac OS X v10.5 and later.

Declared In
NSOperation.h

Constants

Concurrent Operation Constants

Indicates the number of supported concurrent operations.

```
enum {
    NSOperationQueueDefaultMaxConcurrentOperationCount = -1
};
```

Constants
`NSOperationQueueDefaultMaxConcurrentOperationCount`
The default maximum number of operations is determined dynamically by the `NSOperationQueue` object based on current system conditions.

Available in Mac OS X v10.5 and later.

Declared in `NSOperation.h`.

Declared In
NSOperation.h

NSOutputStream Class Reference

Inherits from	NSStream : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.3 and later.
Declared in	Foundation/NSStream.h
Companion guide	Stream Programming Guide for Cocoa
Related sample code	CocoaEcho CocoaHTTPServer CocoaSOAP

Overview

The `NSOutputStream` class is a subclass of `NSStream` that provides write-only stream functionality.

Subclassing Notes

The `NSOutputStream` is a concrete subclass of `NSStream` that lets you write data to a stream. Although `NSOutputStream` is probably sufficient for most situations requiring this capability, you can create a subclass of `NSOutputStream` if you want more specialized behavior (for example, you want to record statistics on the data in a stream).

Methods to Override

To create a subclass of `NSOutputStream` you may have to implement initializers for the type of stream data supported and suitably reimplement existing initializers. You must also provide complete implementations of the following methods:

- `write:maxLength:` (page 1222)

  From the current write pointer, take up to the number of bytes specified in the `maxLength:` parameter from the client-supplied buffer (first parameter) and put them onto the stream. The buffer must be of the size specified by the second parameter. To prepare for the next operation, offset the write pointer by the number of bytes written. Return a signed integer based on the outcome of the current operation:

  - If the write operation is successful, return the actual number of bytes put onto the stream.

- If there was an error writing to the stream, return -1.
- If the stream is of a fixed length and has reached its capacity, return zero.

■ `hasSpaceAvailable` (page 1220)

Return `YES` if the stream can currently accept more data, `NO` if it cannot. If you want to be semantically compatible with `NSOutputStream`, return `YES` if a write must be attempted to determine if space is available.

Tasks

Creating Streams

`+ outputStreamToMemory` (page 1220)
Creates and returns an initialized output stream that will write stream data to memory.

`+ outputStreamToBuffer:capacity:` (page 1218)
Creates and returns an initialized output stream that can write to a provided buffer.

`+ outputStreamToFileAtPath:append:` (page 1219)
Creates and returns an initialized output stream for writing to a specified file.

`- initToMemory` (page 1222)
Returns an initialized output stream that will write to memory.

`- initToBuffer:capacity:` (page 1220)
Returns an initialized output stream that can write to a provided buffer.

`- initToFileAtPath:append:` (page 1221)
Returns an initialized output stream for writing to a specified file.

Using Streams

`- hasSpaceAvailable` (page 1220)
Returns whether the receiver can be written to.

`- write:maxLength:` (page 1222)
Writes the contents of a provided data buffer to the receiver.

Class Methods

outputStreamToBuffer:capacity:

Creates and returns an initialized output stream that can write to a provided buffer.

```
+ (id)outputStreamToBuffer:(uint8_t *)buffer capacity:(NSUInteger)capacity
```

Parameters

buffer

The buffer the output stream will write to.

capacity

The size of the buffer in bytes.

Return Value

An initialized output stream that can write to *buffer*.

Discussion

The stream must be opened before it can be used.

When the number of bytes written to *buffer* has reached *capacity*, the stream's `streamStatus` (page 1504) will return `NSStreamStatusAtEnd`.

Availability

Available in Mac OS X v10.3 and later.

See Also

`+ outputStreamToMemory` (page 1220)

`+ outputStreamToFileAtPath:append:` (page 1219)

`- initToBuffer:capacity:` (page 1220)

Declared In

`NSStream.h`

outputStreamToFileAtPath:append:

Creates and returns an initialized output stream for writing to a specified file.

```
+ (id)outputStreamToFileAtPath:(NSString *)path append:(BOOL)shouldAppend
```

Parameters

path

The path to the file the output stream will write to.

shouldAppend

`YES` if newly written data should be appended to any existing file contents, `NO` otherwise.

Return Value

An initialized output stream that can write to *path*.

Discussion

The stream must be opened before it can be used.

Availability

Available in Mac OS X v10.3 and later.

See Also

`+ outputStreamToMemory` (page 1220)

`+ outputStreamToBuffer:capacity:` (page 1218)

`- initToFileAtPath:append:` (page 1221)

Declared In
NSStream.h

outputStreamToMemory

Creates and returns an initialized output stream that will write stream data to memory.

```
+ (id)outputStreamToMemory
```

Return Value
An initialized output stream that will write stream data to memory.

Discussion
The stream must be opened before it can be used.

You retrieve the contents of the memory stream by sending the message `propertyForKey:` (page 1501) to the receiver with an argument of `NSStreamDataWrittenToMemoryStreamKey`.

Availability
Available in Mac OS X v10.3 and later.

See Also
`+ outputStreamToBuffer:capacity:` (page 1218)
`+ outputStreamToFileAtPath:append:` (page 1219)
`- initToMemory` (page 1222)

Declared In
NSStream.h

Instance Methods

hasSpaceAvailable

Returns whether the receiver can be written to.

```
- (BOOL)hasSpaceAvailable
```

Return Value
`YES` if the receiver can be written to or if a write must be attempted in order to determine if space is available, `NO` otherwise.

Availability
Available in Mac OS X v10.3 and later.

Declared In
NSStream.h

initToBuffer:capacity:

Returns an initialized output stream that can write to a provided buffer.

```
- (id)initToBuffer:(uint8_t *)buffer capacity:(NSUInteger)capacity
```

Parameters

buffer

The buffer the output stream will write to.

capacity

The size of the buffer in bytes.

Return Value

An initialized output stream that can write to *buffer*.

Discussion

The stream must be opened before it can be used.

When the number of bytes written to *buffer* has reached *capacity*, the stream's `streamStatus` (page 1504) will return `NSStreamStatusAtEnd`.

Availability

Available in Mac OS X v10.3 and later.

See Also

- `initToMemory` (page 1222)
- `initToFileAtPath:append:` (page 1221)
+ `outputStreamToBuffer:capacity:` (page 1218)

Declared In

`NSStream.h`

initToFileAtPath:append:

Returns an initialized output stream for writing to a specified file.

```
- (id)initToFileAtPath:(NSString *)path append:(BOOL)shouldAppend
```

Parameters

path

The path to the file the output stream will write to.

shouldAppend

`YES` if newly written data should be appended to any existing file contents, `NO` otherwise.

Return Value

An initialized output stream that can write to *path*.

Discussion

The stream must be opened before it can be used.

Availability

Available in Mac OS X v10.3 and later.

See Also

- `initToMemory` (page 1222)
- `initToBuffer:capacity:` (page 1220)
+ `outputStreamToFileAtPath:append:` (page 1219)

Declared In
NSStream.h

initToMemory

Returns an initialized output stream that will write to memory.

```
- (id)initToMemory
```

Return Value
An initialized output stream that will write stream data to memory.

Discussion
The stream must be opened before it can be used.

The contents of the memory stream are retrieved by passing the constant `NSStreamDataWrittenToMemoryStreamKey` to `propertyForKey:` (page 1501).

Availability
Available in Mac OS X v10.3 and later.

See Also
- `initToBuffer:capacity:` (page 1220)
- `initToFileAtPath:append:` (page 1221)
+ `outputStreamToMemory` (page 1220)

Declared In
NSStream.h

write:maxLength:

Writes the contents of a provided data buffer to the receiver.

```
- (NSInteger)write:(const uint8_t *)buffer maxLength:(NSUInteger)length
```

Parameters
buffer
The data to write.

length
The length of the data buffer, in bytes.

Return Value
The number of bytes actually written, or `-1` if an error occurs. More information about the error can be obtained with `streamError` (page 1504). If the receiver is a fixed-length stream and has reached its capacity, `0` is returned.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CocoaEcho

Declared In
`NSStream.h`

NSPipe Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSFileHandle.h
Companion guide	Interacting with the Operating System
Related sample code	Moriarity

Overview

`NSPipe` objects provide an object-oriented interface for accessing pipes. An `NSPipe` object represents both ends of a pipe and enables communication through the pipe. A pipe is a one-way communications channel between related processes; one process writes data, while the other process reads that data. The data that passes through the pipe is buffered; the size of the buffer is determined by the underlying operating system. `NSPipe` is an abstract class, the public interface of a class cluster.

Tasks

Creating an NSPipe Object

`- init` (page 1227)
Returns an initialized `NSPipe` object.

`+ pipe` (page 1226)
Returns an `NSPipe` object.

Getting the File Handles for a Pipe

`- fileHandleForReading` (page 1226)
Returns the receiver's read file handle.

`- fileHandleForWriting` (page 1227)
Returns the receiver's write file handle.

Class Methods

pipe

Returns an `NSPipe` object.

```
+ (id)pipe
```

Return Value
An initialized `NSPipe` object. Returns `nil` if the method encounters errors while attempting to create the pipe or the `NSFileHandle` objects that serve as endpoints of the pipe.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Moriarity

Declared In
`NSFileHandle.h`

Instance Methods

fileHandleForReading

Returns the receiver's read file handle.

```
- (NSFileHandle *)fileHandleForReading
```

Return Value
The receiver's read file handle.The descriptor represented by this object is deleted, and the object itself is automatically deallocated when the receiver is deallocated.

Discussion
You use the returned file handle to read from the pipe using `NSFileHandle`'s read methods—`availableData` (page 610), `readDataToEndOfFile` (page 614), and `readDataOfLength:` (page 613).

You don't need to send `closeFile` (page 611) to this object or explicitly release the object after you have finished using it.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSFileHandle.h`

fileHandleForWriting

Returns the receiver's write file handle.

```
- (NSFileHandle *)fileHandleForWriting
```

Return Value
The receiver's write file handle. This object is automatically deallocated when the receiver is deallocated.

Discussion
You use the returned file handle to write to the pipe using `NSFileHandle`'s `writeData:` (page 619) method. When you are finished writing data to this object, send it a `closeFile` (page 611) message to delete the descriptor. Deleting the descriptor causes the reading process to receive an end-of-data signal (an empty `NSData` object).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSFileHandle.h`

init

Returns an initialized `NSPipe` object.

```
- (id)init
```

Return Value
An initialized `NSPipe` object. Returns `nil` if the method encounters errors while attempting to create the pipe or the `NSFileHandle` objects that serve as endpoints of the pipe.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ pipe` (page 1226)

Declared In
`NSFileHandle.h`

NSPointerArray Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSFastEnumeration NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Companion guides	Collections Programming Topics for Cocoa Garbage Collection Programming Guide
Availability	Available in Mac OS X v10.5 and later.

Overview

`NSPointerArray` is a mutable collection modeled after `NSArray` but it can also hold `NULL` values, which can be inserted or extracted (and which contribute to the object's count). Moreover, unlike traditional arrays, you can set the count of the array directly. In a garbage collected environment, if you specify a zeroing weak memory configuration, if an element is collected it is replaced by a `NULL` value.

The copying and archiving protocols are applicable only when a pointer array is configured for object uses.

The fast fast enumeration protocol (that is, use a pointer array in the `for...in` language construct—see Fast Enumeration in *The Objective-C 2.0 Programming Language*) will yield `NULL` values that are present in the array. It is defined for all types of pointers although the language syntax doesn't directly support this.

Tasks

Creating and Initializing a New Pointer Array

`- initWithOptions:` (page 1233)
Initializes the receiver to use the given options.

`- initWithPointerFunctions:` (page 1234)
Initializes the receiver to use the given functions.

`+ pointerArrayWithOptions:` (page 1230)
Returns a new pointer array initialized to use the given options.

+ `pointerArrayWithPointerFunctions:` (page 1231)
  A new pointer array initialized to use the given functions.
+ `pointerArrayWithStrongObjects` (page 1231)
  Returns a new pointer array that maintains strong references to its elements.
+ `pointerArrayWithWeakObjects` (page 1232)
  Returns a new pointer array that maintains weak references to its elements.

Managing the Collection

- `count` (page 1233)
  Returns the number of elements in the receiver.
- `setCount:` (page 1236)
  Sets the count for the receiver.
- `allObjects` (page 1233)
  Returns an array containing all the objects in the receiver.
- `pointerAtIndex:` (page 1235)
  Returns the pointer at a given index.
- `addPointer:` (page 1232)
  Adds a given pointer to the receiver.
- `removePointerAtIndex:` (page 1236)
  Removes the pointer at a given index.
- `insertPointer:atIndex:` (page 1234)
  Inserts a pointer at a given index.
- `replacePointerAtIndex:withPointer:` (page 1236)
  Replaces the pointer at a given index.
- `compact` (page 1233)
  Removes `NULL` values from the receiver.

Getting the Pointer Functions

- `pointerFunctions` (page 1235)
  Returns a new `NSPointerFunctions` object reflecting the functions in use by the receiver.

Class Methods

pointerArrayWithOptions:

Returns a new pointer array initialized to use the given options.

```
+ (id)pointerArrayWithOptions:(NSPointerFunctionsOptions)options
```

Parameters

options

The pointer functions options for the new instance.

Return Value

A new pointer array initialized to use the given options.

Availability

Available in Mac OS X v10.5 and later.

See Also

+ pointerArrayWithPointerFunctions: (page 1231)
+ pointerArrayWithStrongObjects (page 1231)
+ pointerArrayWithWeakObjects (page 1232)

Declared In

NSPointerArray.h

pointerArrayWithPointerFunctions:

A new pointer array initialized to use the given functions.

```
+ (id)pointerArrayWithPointerFunctions:(NSPointerFunctions *)functions
```

Parameters

functions

The pointer functions for the new instance.

Return Value

A new pointer array initialized to use the given pointer functions.

Availability

Available in Mac OS X v10.5 and later.

See Also

+ pointerArrayWithOptions: (page 1230)
+ pointerArrayWithStrongObjects (page 1231)
+ pointerArrayWithWeakObjects (page 1232)

Declared In

NSPointerArray.h

pointerArrayWithStrongObjects

Returns a new pointer array that maintains strong references to its elements.

```
+ (id)pointerArrayWithStrongObjects
```

Return Value

A new pointer array that maintains strong references to its elements.

Availability

Available in Mac OS X v10.5 and later.

See Also
+ pointerArrayWithWeakObjects (page 1232)
+ pointerArrayWithOptions: (page 1230)
+ pointerArrayWithPointerFunctions: (page 1231)

Declared In
NSPointerArray.h

pointerArrayWithWeakObjects

Returns a new pointer array that maintains weak references to its elements.

```
+ (id)pointerArrayWithWeakObjects
```

Return Value
A new pointer array that maintains weak references to its elements.

Availability
Available in Mac OS X v10.5 and later.

See Also
+ pointerArrayWithStrongObjects (page 1231)
+ pointerArrayWithOptions: (page 1230)
+ pointerArrayWithPointerFunctions: (page 1231)

Declared In
NSPointerArray.h

Instance Methods

addPointer:

Adds a given pointer to the receiver.

```
- (void)addPointer:(void *)pointer
```

Parameters
pointer
The pointer to add. This value may be NULL.

Discussion
pointer is added at index count (page 1233).

Availability
Available in Mac OS X v10.5 and later.

Declared In
NSPointerArray.h

allObjects

Returns an array containing all the objects in the receiver.

```
- (NSArray *)allObjects
```

Return Value
An array containing all the object in the receiver.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `count` (page 1233)

Declared In
`NSPointerArray.h`

compact

Removes `NULL` values from the receiver.

```
- (void)compact
```

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSPointerArray.h`

count

Returns the number of elements in the receiver.

```
- (NSUInteger)count
```

Return Value
The number of elements in the receiver.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `setCount:` (page 1236)

Declared In
`NSPointerArray.h`

initWithOptions:

Initializes the receiver to use the given options.

```
- (id)initWithOptions:(NSPointerFunctionsOptions)options
```

Parameters

options

The pointer functions options for the new instance.

Return Value

The receiver, initialized to use the given options.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `initWithPointerFunctions:` (page 1234)
+ `pointerArrayWithOptions:` (page 1230)
+ `pointerArrayWithPointerFunctions:` (page 1231)

Declared In

`NSPointerArray.h`

initWithPointerFunctions:

Initializes the receiver to use the given functions.

```
- (id)initWithPointerFunctions:(NSPointerFunctions *)functions
```

Parameters

functions

The pointer functions for the new instance.

Return Value

The receiver, initialized to use the given functions.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `initWithOptions:` (page 1233)
+ `pointerArrayWithPointerFunctions:` (page 1231)
+ `pointerArrayWithOptions:` (page 1230)

Declared In

`NSPointerArray.h`

insertPointer:atIndex:

Inserts a pointer at a given index.

```
- (void)insertPointer:(void *)item atIndex:(NSUInteger)index
```

Parameters

item

The pointer to add.

index

The index of an element in the receiver. This value must be less than the `count` (page 1233) of the receiver.

Discussion
Elements at and above *index*, including `NULL` values, slide higher.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSPointerArray.h`

pointerAtIndex:

Returns the pointer at a given index.

```
- (void *)pointerAtIndex:(NSUInteger)index
```

Parameters

index

The index of an element in the receiver. This value must be less than the `count` (page 1233) of the receiver.

Return Value
The pointer at *index*.

Discussion
The returned value may be `NULL`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSPointerArray.h`

pointerFunctions

Returns a new `NSPointerFunctions` object reflecting the functions in use by the receiver.

```
- (NSPointerFunctions *)pointerFunctions
```

Return Value
A new `NSPointerFunctions` object reflecting the functions in use by the receiver.

Discussion
The returned object is a new `NSPointerFunctions` object that you can modify and/or use directly to create other pointer collections.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSPointerArray.h`

removePointerAtIndex:

Removes the pointer at a given index.

```
- (void)removePointerAtIndex:(NSUInteger)index
```

Parameters

index

The index of an element in the receiver. This value must be less than the `count` (page 1233) of the receiver.

Discussion

Elements above *index*, including `NULL` values, slide lower.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSPointerArray.h`

replacePointerAtIndex:withPointer:

Replaces the pointer at a given index.

```
- (void)replacePointerAtIndex:(NSUInteger)index withPointer:(void *)item
```

Parameters

index

The index of an element in the receiver. This value must be less than the `count` (page 1233) of the receiver.

item

The item with which to replace the element at *index*. This value may be `NULL`.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSPointerArray.h`

setCount:

Sets the count for the receiver.

```
- (void)setCount:(NSUInteger)count
```

Parameters

count

The count for the receiver.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `count` (page 1233)

Discussion
If *count* is greater than the `count` (page 1233) of the receiver, `NULL` values are added; if *count* is less than the `count` (page 1233) of the receiver, then elements at indexes *count* and greater are removed from the receiver.

Declared In
`NSPointerArray.h`

NSPointerFunctions Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSPointerFunctions.h
Availability	Available in Mac OS X v10.5 and later.
Companion guides	Collections Programming Topics for Cocoa Garbage Collection Programming Guide

Overview

An instance of `NSPointerFunctions` defines callout functions appropriate for managing a pointer reference held somewhere else.

The functions specified by an instance of `NSPointerFunctions` are separated into two clusters—those that define "personality" such as "object" or "C-string", and those that describe memory management issues such as a memory deallocation function. There are constants for common personalities and memory manager selections (see "Memory and Personality Options" (page 1244)).

`NSHashTable`, `NSMapTable`, and `NSPointerArray` use an `NSPointerFunctions` object to define the acquisition and retention behavior for the pointers they manage. Note, however, that not all combinations of personality and memory management behavior are valid for these collections. The pointer collection objects copy the `NSPointerFunctions` object on input and output, so you cannot usefully subclass `NSPointerFunctions`.

Tasks

Creating and Initializing an NSPointerFunctions Object

`- initWithOptions:` (page 1243)
Returns an `NSPointerFunctions` object initialized with the given options.

`+ pointerFunctionsWithOptions:` (page 1243)
Returns a new `NSPointerFunctions` object initialized with the given options.

Personality Functions

`hashFunction` (page 1241) *property*
The hash function.

`isEqualFunction` (page 1241) *property*
The function used to compare pointers.

`sizeFunction` (page 1242) *property*
The function used to determine the size of pointers.

`descriptionFunction` (page 1241) *property*
The function used to describe elements.

Memory Configuration

`acquireFunction` (page 1240) *property*
The function used to acquire memory.

`relinquishFunction` (page 1241) *property*
The function used to relinquish memory.

`usesStrongWriteBarrier` (page 1242) *property*
Specifies whether, in a garbage collected environment, pointers should be assigned using a strong write barrier.

`usesWeakReadAndWriteBarriers` (page 1242) *property*
Specifies whether, in a garbage collected environment, pointers should use weak read and write barriers.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C 2.0 Programming Language*.

acquireFunction

The function used to acquire memory.

```
@property void *(*acquireFunction)(const void *src, NSUInteger (*size)(const void
    *item), BOOL shouldCopy)
```

Discussion
This specifies the function to use for copy-in operations.

Availability
Available in Mac OS X v10.5 and later.

See Also
`@property relinquishFunction` (page 1241)

Declared In
`NSPointerFunctions.h`

descriptionFunction

The function used to describe elements.

```
@property NSString *(*descriptionFunction)(const void *item)
```

Discussion
This function is used by description methods for hash and map tables.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSPointerFunctions.h`

hashFunction

The hash function.

```
@property NSUInteger (*hashFunction)(const void *item, NSUInteger (*size)(const
    void *item))
```

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSPointerFunctions.h`

isEqualFunction

The function used to compare pointers.

```
@property BOOL (*isEqualFunction)(const void *item1, const void*item2, NSUInteger
     (*size)(const void *item))
```

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSPointerFunctions.h`

relinquishFunction

The function used to relinquish memory.

```
@property void (*relinquishFunction)(const void *item, NSUInteger (*size)(const
    void *item))
```

Discussion
This specifies the function to use when an item is removed from a table or pointer array.

Availability
Available in Mac OS X v10.5 and later.

See Also
`@property acquireFunction` (page 1240)

Declared In
`NSPointerFunctions.h`

sizeFunction

The function used to determine the size of pointers.

```
@property NSUInteger (*sizeFunction)(const void *item)
```

Discussion
This function is used for copy-in operations (unless the collection has an object personality).

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSPointerFunctions.h`

usesStrongWriteBarrier

Specifies whether, in a garbage collected environment, pointers should be assigned using a strong write barrier.

```
@property BOOL usesStrongWriteBarrier
```

Discussion
If you use garbage collection, read and write barrier functions must be used when pointers are from memory scanned by the collector.

Availability
Available in Mac OS X v10.5 and later.

See Also
`@property usesWeakReadAndWriteBarriers` (page 1242)

Declared In
`NSPointerFunctions.h`

usesWeakReadAndWriteBarriers

Specifies whether, in a garbage collected environment, pointers should use weak read and write barriers.

```
@property BOOL usesWeakReadAndWriteBarriers
```

Discussion
If you use garbage collection, read and write barrier functions must be used when pointers are from memory scanned by the collector.

Availability
Available in Mac OS X v10.5 and later.

See Also
`@property usesStrongWriteBarrier` (page 1242)

Declared In
`NSPointerFunctions.h`

Class Methods

pointerFunctionsWithOptions:

Returns a new `NSPointerFunctions` object initialized with the given options.

```
+ (id)pointerFunctionsWithOptions:(NSPointerFunctionsOptions)options
```

Parameters
options
The options for the new `NSPointerFunctions` object.

Return Value
A new `NSPointerFunctions` object initialized with the given options.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSPointerFunctions.h`

Instance Methods

initWithOptions:

Returns an `NSPointerFunctions` object initialized with the given options.

```
- (id)initWithOptions:(NSPointerFunctionsOptions)options
```

Parameters
options
The options for the new `NSPointerFunctions` object.

Return Value
The receiver, initialized with the given options.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSPointerFunctions.h`

Constants

NSPointerFunctionsOptions

Defines the memory and personality options for an `NSPointerFunctions` object.

```
typedef NSUInteger NSPointerFunctionsOptions;
```

Discussion
For values, see "Memory and Personality Options" (page 1244).

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSPointerFunctions.h`

Memory and Personality Options

Specify memory and personality options for an `NSPointerFunctions` object.

```
enum {
    NSPointerFunctionsStrongMemory = (0 << 0),
    NSPointerFunctionsZeroingWeakMemory = (1 << 0),
    NSPointerFunctionsOpaqueMemory = (2 << 0),
    NSPointerFunctionsMallocMemory = (3 << 0),
    NSPointerFunctionsMachVirtualMemory = (4 << 0),
    NSPointerFunctionsObjectPersonality = (0 << 8),
    NSPointerFunctionsOpaquePersonality = (1 << 8),
    NSPointerFunctionsObjectPointerPersonality = (2 << 8),
    NSPointerFunctionsCStringPersonality = (3 << 8),
    NSPointerFunctionsStructPersonality = (4 << 8),
    NSPointerFunctionsIntegerPersonality = (5 << 8),
    NSPointerFunctionsCopyIn = (1 << 16),
};
```

Constants

`NSPointerFunctionsStrongMemory`

Use strong write-barriers to backing store; use garbage-collected memory on copy-in.

This is the default memory value.

As a special case, if you do not use garbage collection and specify this value in conjunction with `NSPointerFunctionsObjectPersonality` (page 1245) then the `NSPointerFunctions` object uses `retain` and `release`.

Available in Mac OS X v10.5 and later.

Declared in `NSPointerFunctions.h`.

`NSPointerFunctionsZeroingWeakMemory`

Use weak read and write barriers; use garbage-collected memory on copyIn.

Available in Mac OS X v10.5 and later.

Declared in `NSPointerFunctions.h`.

`NSPointerFunctionsOpaqueMemory`
Take no action when pointers are deleted.

This is essentially a no-op relinquish function; the acquire function is only used for copy-in operations. This option is unlikely a to be a good choice for objects.

Available in Mac OS X v10.5 and later.

Declared in `NSPointerFunctions.h`.

`NSPointerFunctionsMallocMemory`
Use `free()` on removal, `calloc()` on copy in.

Available in Mac OS X v10.5 and later.

Declared in `NSPointerFunctions.h`.

`NSPointerFunctionsMachVirtualMemory`
Use Mach memory.

Available in Mac OS X v10.5 and later.

Declared in `NSPointerFunctions.h`.

`NSPointerFunctionsObjectPersonality`
Use `hash` and `isEqual` methods for hashing and equality comparisons, use the `description` method for a description.

This is the default personality value.

As a special case, if you do not use garbage collection and specify this value in conjunction with `NSPointerFunctionsStrongMemory` (page 1244) then the `NSPointerFunctions` object uses `retain` and `release`.

Available in Mac OS X v10.5 and later.

Declared in `NSPointerFunctions.h`.

`NSPointerFunctionsOpaquePersonality`
Use shifted pointer for the hash value and direct comparison to determine equality.

Available in Mac OS X v10.5 and later.

Declared in `NSPointerFunctions.h`.

`NSPointerFunctionsObjectPointerPersonality`
Use shifted pointer for the hash value and direct comparison to determine equality; use the `description` method for a description.

As a special case, if you do not use garbage collection and specify this value in conjunction with `NSPointerFunctionsStrongMemory` (page 1244) then the `NSPointerFunctions` object uses `retain` and `release`.

Available in Mac OS X v10.5 and later.

Declared in `NSPointerFunctions.h`.

`NSPointerFunctionsCStringPersonality`
Use a string hash and `strcmp`; C-string '`%s`' style description.

Available in Mac OS X v10.5 and later.

Declared in `NSPointerFunctions.h`.

`NSPointerFunctionsStructPersonality`
Use a memory hash and `memcmp` (using a size function that you must set—see `sizeFunction` (page 1242)).

Available in Mac OS X v10.5 and later.

Declared in `NSPointerFunctions.h`.

`NSPointerFunctionsIntegerPersonality`
Use unshifted value as hash and equality.

Available in Mac OS X v10.5 and later.

Declared in `NSPointerFunctions.h`.

`NSPointerFunctionsCopyIn`
Use the memory acquire function to allocate and copy items on input (see `acquireFunction` (page 1240)).

Available in Mac OS X v10.5 and later.

Declared in `NSPointerFunctions.h`.

Discussion
Memory options are mutually exclusive and personality options are mutually exclusive.

Declared In
`NSPointerFunctions.h`

NSPort Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSPort.h
Companion guides	Run Loops Distributed Objects Programming Topics
Related sample code	SimpleThreads TrivialThreads

Overview

`NSPort` is an abstract class that represents a communication channel. Communication occurs between `NSPort` objects, which typically reside in different threads or tasks. The distributed objects system uses `NSPort` objects to send `NSPortMessage` objects back and forth. You should implement interapplication communication using distributed objects whenever possible and use `NSPort` objects only when necessary.

To receive incoming messages, `NSPort` objects must be added to an `NSRunLoop` object as input sources. `NSConnection` objects automatically add their receive port when initialized.

When an `NSPort` object receives a port message, it forwards the message to its delegate in a `handleMachMessage:` (page 849) or `handlePortMessage:` (page 1255) message. The delegate should implement only one of these methods to process the incoming message in whatever form desired. `handleMachMessage:` (page 849) provides a message as a raw Mach message beginning with a `msg_header_t` structure. `handlePortMessage:` (page 1255) provides a message as an `NSPortMessage` object, which is an object-oriented wrapper for a Mach message. If a delegate has not been set, the `NSPort` object handles the message itself.

When you are finished using a port object, you must explicitly invalidate the port object prior to sending it a `release` message. Similarly, if your application uses garbage collection, you must invalidate the port object before removing any strong references to it. If you do not invalidate the port, the resulting port object may linger and create a memory leak. To invalidate the port object, invoke its `invalidate` method.

Foundation defines three concrete subclasses of `NSPort`. `NSMachPort` and `NSMessagePort` allow local (on the same machine) communication only. `NSSocketPort` allows for both local and remote communication, but may be more expensive than the others for the local case. When creating an `NSPort` object, using `allocWithZone:` (page 1249) or `port` (page 1250), an `NSMachPort` object is created instead.

Important: `NSPort` conforms to the `NSCoding` protocol, but only supports coding by an `NSPortCoder`. `NSPort` and its subclasses do not support archiving.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 2034)
- `initWithCoder:` (page 2034)

NSCopying
- `copyWithZone:` (page 2042)

Tasks

Creating Instances

`+ allocWithZone:` (page 1249)
Returns an instance of the `NSMachPort` class.

`+ port` (page 1250)
Creates and returns a new `NSPort` object capable of both sending and receiving messages.

Validation

`- invalidate` (page 1251)
Marks the receiver as invalid and posts an `NSPortDidBecomeInvalidNotification` (page 1256) to the default notification center.

`- isValid` (page 1252)
Returns a Boolean value that indicates whether the receiver is valid.

Setting the Delegate

`- setDelegate:` (page 1255)
Sets the receiver's delegate to a given object.

`- delegate` (page 1251)
Returns the receiver's delegate.

Creating Connections

- `addConnection:toRunLoop:forMode:` (page 1250)
  Adds the receiver to the list of ports monitored by a given run loop for the given input mode.
- `removeConnection:fromRunLoop:forMode:` (page 1252)
  Removes the receiver from the list of ports monitored by *`runLoop`* in the given input mode, *`mode`*.

Setting Information

- `sendBeforeDate:components:from:reserved:` (page 1254)
  This method is provided for subclasses that have custom types of `NSPort`.
- `sendBeforeDate:msgid:components:from:reserved:` (page 1254)
  This method is provided for subclasses that have custom types of `NSPort`.
- `reservedSpaceLength` (page 1253)
  Returns the number of bytes of space reserved by the receiver for sending data.

Port Monitoring

- `removeFromRunLoop:forMode:` (page 1252)
  This method should be implemented by a subclass to stop monitoring of a port when removed from a give run loop in a given input mode.
- `scheduleInRunLoop:forMode:` (page 1253)
  This method should be implemented by a subclass to set up monitoring of a port when added to a given run loop in a given input mode.

Handling Port Messages

- `handlePortMessage:` (page 1255) *delegate method*
  Processes a given incoming message on the port.

Class Methods

allocWithZone:

Returns an instance of the `NSMachPort` class.

```
+ (id)allocWithZone:(NSZone *)zone
```

Parameters

`zone`
The memory zone in which to allocate the new object.

Return Value
An instance of the `NSMachPort` class.

Discussion
For backward compatibility on Mach, `allocWithZone:` returns an instance of the `NSMachPort` class when sent to the `NSPort` class. Otherwise, it returns an instance of a concrete subclass that can be used for messaging between threads or processes on the local machine, or, in the case of `NSSocketPort`, between processes on separate machines.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSPort.h`

port

Creates and returns a new `NSPort` object capable of both sending and receiving messages.

```
+ (NSPort *)port
```

Return Value
A new `NSPort` object capable of both sending and receiving messages.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ allocWithZone:` (page 1249)

Related Sample Code
SimpleThreads
TrivialThreads

Declared In
`NSPort.h`

Instance Methods

addConnection:toRunLoop:forMode:

Adds the receiver to the list of ports monitored by a given run loop for the given input mode.

```
- (void)addConnection:(NSConnection *)connection toRunLoop:(NSRunLoop *)runLoop
    forMode:(NSString *)mode
```

Parameters

connection
The connection object that invoked this method.

runLoop
The run loop to which to add the receiver.

mode
The run loop mode in which to add the receiver.

Discussion
You should not call this method directly. The method is provided for subclassers who wish to provide their own custom types of `NSPort`. The `NSConnection` object, *connection*, calls this method at the appropriate times.

Availability
Available in Mac OS X v10.0 and later.

See Also
`addPort:forMode:` (page 1333) (NSRunLoop)

Declared In
`NSPort.h`

delegate

Returns the receiver's delegate.

```
- (id)delegate
```

Return Value
The receiver's delegate.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setDelegate:` (page 1255)

Declared In
`NSPort.h`

invalidate

Marks the receiver as invalid and posts an `NSPortDidBecomeInvalidNotification` (page 1256) to the default notification center.

```
- (void)invalidate
```

Discussion
You must call this method before releasing a port object (or removing strong references to it if your application is garbage collected).

Availability
Available in Mac OS X v10.0 and later.

See Also
`- isValid` (page 1252)

Declared In
`NSPort.h`

isValid

Returns a Boolean value that indicates whether the receiver is valid.

```
- (BOOL)isValid
```

Return Value
`NO` if the receiver is known to be invalid, otherwise `YES`.

Discussion
An `NSPort` object becomes invalid when its underlying communication resource, which is operating system dependent, is closed or damaged.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- invalidate` (page 1251)

Declared In
`NSPort.h`

removeConnection:fromRunLoop:forMode:

Removes the receiver from the list of ports monitored by *runLoop* in the given input mode, *mode*.

```
- (void)removeConnection:(NSConnection *)connection fromRunLoop:(NSRunLoop *)runLoop
    forMode:(NSString *)mode
```

Parameters

connection
The connection object that invoked this method.

runLoop
The run loop to which to add the receiver.

mode
The run loop mode in which to add the receiver.

Discussion
You should not call this method directly. The method is provided for subclassers who wish to provide their own custom types of `NSPort`. The `NSConnection` object, *connection*, calls this method at the appropriate times.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSPort.h`

removeFromRunLoop:forMode:

This method should be implemented by a subclass to stop monitoring of a port when removed from a give run loop in a given input mode.

```
- (void)removeFromRunLoop:(NSRunLoop *)runLoop forMode:(NSString *)mode
```

Parameters

runLoop

The run loop from which to remove the receiver.

mode

The run loop mode from which to remove the receiver

Discussion
This method should not be called directly.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `scheduleInRunLoop:forMode:` (page 1253)

Declared In
`NSPort.h`

reservedSpaceLength

Returns the number of bytes of space reserved by the receiver for sending data.

```
- (NSUInteger)reservedSpaceLength
```

Return Value
The number of bytes reserved by the receiver for sending data. The default length is `0`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSPort.h`

scheduleInRunLoop:forMode:

This method should be implemented by a subclass to set up monitoring of a port when added to a given run loop in a given input mode.

```
- (void)scheduleInRunLoop:(NSRunLoop *)runLoop forMode:(NSString *)mode
```

Parameters

runLoop

The run loop to which to add the receiver.

mode

The run loop mode to which to add the receiver

Discussion
This method should not be called directly.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `removeFromRunLoop:forMode:` (page 1252)

Declared In
`NSPort.h`

sendBeforeDate:components:from:reserved:

This method is provided for subclasses that have custom types of `NSPort`.

```
- (BOOL)sendBeforeDate:(NSDate *)limitDate components:(NSMutableArray *)components
    from:(NSPort *)receivePort reserved:(NSUInteger)headerSpaceReserved
```

Parameters

limitDate
The last instant that a message may be sent.

components
The message components.

receivePort
The receive port.

headerSpaceReserved
The number of bytes reserved for the header.

Discussion
`NSConnection` calls this method at the appropriate times. This method should not be called directly. This method could raise an `NSInvalidSendPortException`, `NSInvalidReceivePortException`, or an `NSPortSendException`, depending on the type of send port and the type of error.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSPort.h`

sendBeforeDate:msgid:components:from:reserved:

This method is provided for subclasses that have custom types of `NSPort`.

```
- (BOOL)sendBeforeDate:(NSDate *)limitDate msgid:(NSUInteger)msgID
    components:(NSMutableArray *)components from:(NSPort *)receivePort
    reserved:(NSUInteger)headerSpaceReserved
```

Parameters

limitDate
The last instant that a message may be sent.

msgID
The message ID.

components
The message components.

receivePort
The receive port.

headerSpaceReserved
The number of bytes reserved for the header.

Discussion
`NSConnection` calls this method at the appropriate times. This method should not be called directly. This method could raise an `NSInvalidSendPortException`, `NSInvalidReceivePortException`, or an `NSPortSendException`, depending on the type of send port and the type of error.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSPort.h`

setDelegate:

Sets the receiver's delegate to a given object.

```
- (void)setDelegate:(id)anObject
```

Parameters
anObject
The delegate for the receiver.

Discussion
Does not retain *anObject*.

Availability
Available in Mac OS X v10.0 and later.

See Also
- delegate (page 1251)

Declared In
`NSPort.h`

Delegate Methods

handlePortMessage:

Processes a given incoming message on the port.

```
- (void)handlePortMessage:(NSPortMessage *)portMessage
```

Parameters
portMessage
An incoming port message.

Discussion
See the `NSPortMessage` class specification for more information.

The delegate should implement only one of `handleMachMessage:` (page 849) and `handlePortMessage:`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSPort.h`

Notifications

NSPortDidBecomeInvalidNotification

Posted from the `invalidate` (page 1251) method, which is invoked when the `NSPort` is deallocated or when it notices that its communication channel has been damaged. The notification object is the `NSPort` object that has become invalid. This notification does not contain a *`userInfo`* dictionary.

An `NSSocketPort` object cannot detect when its connection to a remote port is lost, even if the remote port is on the same machine. Therefore, it cannot invalidate itself and post this notification. Instead, you must detect the timeout error when the next message is sent.

The `NSPort` object posting this notification is no longer useful, so all receivers should unregister themselves for any notifications involving the `NSPort`. A method receiving this notification should check to see which port became invalid before attempting to do anything. In particular, observers that receive all `NSPortDidBecomeInvalidNotification` messages should be aware that communication with the window server is handled through an `NSPort`. If this port becomes invalid, drawing operations will cause a fatal error.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSPort.h`

NSPortCoder Class Reference

Inherits from	NSCoder : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSPortCoder.h
Companion guide	Distributed Objects Programming Topics

Overview

`NSPortCoder` is a concrete subclass of `NSCoder` used in the distributed objects system to transmit object proxies (and sometimes objects themselves) between `NSConnection` objects. An `NSPortCoder` instance is always created and used by an `NSConnection` object; you should never need to explicitly create or use one directly yourself.

Tasks

Creating an NSPortCoder Object

`+ portCoderWithReceivePort:sendPort:components:` (page 1258)
Creates and returns a new `NSPortCoder` object.

`- initWithReceivePort:sendPort:components:` (page 1260)
Initializes and returns an `NSPortCoder` object.

Getting the Connection

`- connection` (page 1259)
Returns the `NSConnection` object that uses the receiver.

Encoding NSPort Objects

- `encodePortObject:` (page 1260)
  Encodes a given port so it can be properly reconstituted in the receiving process or thread.
- `decodePortObject` (page 1259)
  Decodes and returns an `NSPort` object that was previously encoded with any of the general `encode...Object:` messages.

Checking for Encoding

- `isBycopy` (page 1261)
  Returns a Boolean value that indicates whether the receiver is encoding an object by copying it.
- `isByref` (page 1261)
  Returns a Boolean value that indicates whether the receiver is encoding an object by reference.

Dispatching

- `dispatch` (page 1259)
  Processes and acts upon the distributed object message with which the receiver was initialized.

Class Methods

portCoderWithReceivePort:sendPort:components:

Creates and returns a new `NSPortCoder` object.

```
+ (id)portCoderWithReceivePort:(NSPort *)rcvPort sendPort:(NSPort *)sndPort
    components:(NSArray *)comps
```

Parameters

receiverPort
The receiver port.

sendPort
The send port.

components
An array containing an encoded distributed objects message.

Return Value
A new `NSPortCoder` object connected to the communication ports *receiverPort* and *sendPort*, with an encoded distributed objects message stored in *components*.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `dispatch` (page 1259)

- initWithReceivePort:sendPort:components: (page 1260)

Declared In
NSPortCoder.h

Instance Methods

connection

Returns the NSConnection object that uses the receiver.

```
- (NSConnection *)connection
```

Return Value
The NSConnection object that uses the receiver. In an object's encodeWithCoder: (page 2034) method, this is the sending (server) connection. In initWithCoder: (page 2034) this is the receiving (client) connection.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSPortCoder.h

decodePortObject

Decodes and returns an NSPort object that was previously encoded with any of the general encode...Object: messages.

```
- (NSPort *)decodePortObject
```

Return Value
An NSPort object that was previously encoded with any of the general encode...Object: messages.

Discussion
This method is primarily for use by NSPort objects themselves—you can always use decodeObject (page 279) to decode any object.

NSPort invokes this method in its initWithCoder: (page 2034) method so the appropriate kernel information for the port can be decoded. A subclass of NSPortCoder shouldn't decode an NSPort by sending it an initWithCoder: (page 2034) message. See Subclassing NSCoder for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSPortCoder.h

dispatch

Processes and acts upon the distributed object message with which the receiver was initialized.

```
- (void)dispatch
```

Availability
Available in Mac OS X v10.0 and later.

See Also
+ portCoderWithReceivePort:sendPort:components: (page 1258)
- initWithReceivePort:sendPort:components: (page 1260)

Declared In
NSPortCoder.h

encodePortObject:

Encodes a given port so it can be properly reconstituted in the receiving process or thread.

```
- (void)encodePortObject:(NSPort *)aPort
```

Parameters

aPort
The port to encode.

Discussion
This method is primarily for use by NSPort objects themselves—you can always use the general encode...Object: methods to encode any object.

NSPort invokes this method in its encodeWithCoder: (page 2034) method so that the appropriate kernel information for the port can be encoded. A subclass of NSPortCoder should not encode an NSPort by sending it an encodeWithCoder: (page 2034) message. See Subclassing NSCoder for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSPortCoder.h

initWithReceivePort:sendPort:components:

Initializes and returns an NSPortCoder object.

```
- (id)initWithReceivePort:(NSPort *)receiverPort sendPort:(NSPort *)sendPort
    components:(NSArray *)components
```

Parameters

receiverPort
The receive port.

sendPort
The send port.

components
An array containing an encoded distributed objects message.

Discussion
Initializes a newly allocated `NSPortCoder` object connected to the communication ports *`receiverPort`* and *`sendPort`*, with an encoded distributed objects message stored in *`components`*.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ portCoderWithReceivePort:sendPort:components:` (page 1258)
`- dispatch` (page 1259)

Declared In
`NSPortCoder.h`

isBycopy

Returns a Boolean value that indicates whether the receiver is encoding an object by copying it.

```
- (BOOL)isBycopy
```

Return Value
`YES` if the receiver is encoding an object by copying it, `NO` if it expects a proxy.

Discussion
See *Distributed Objects Programming Topics* for more information.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- isByref` (page 1261)

Declared In
`NSPortCoder.h`

isByref

Returns a Boolean value that indicates whether the receiver is encoding an object by reference.

```
- (BOOL)isByref
```

Return Value
`YES` if the receiver is encoding an object `byref`, `NO` if it expects a copy.

Discussion
See *Distributed Objects Programming Topics* for more information.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- isBycopy` (page 1261)

Declared In
`NSPortCoder.h`

NSPortMessage Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSPortMessage.h
Companion guide	Distributed Objects Programming Topics

Overview

An `NSPortMessage` defines a low-level, operating system-independent type for inter-application (and inter-thread) messages. Port messages are used primarily by the distributed objects system. You should implement inter-application communication using distributed objects whenever possible and use `NSPortMessage` only when necessary.

An `NSPortMessage` object has three major parts: the send and receive ports, which are `NSPort` object that link the sender of the message to the receiver, and the components, which form the body of the message. The components are held as an `NSArray` object containing `NSData` and `NSPort` objects. `NSPortMessage`'s `sendBeforeDate:` (page 1266) message sends the components out through the send port; any replies to the message arrive on the receive port. See the `NSPort` class specification for information on handling incoming messages.

An `NSPortMessage` instance can be initialized with a pair of `NSPort` objects and an array of components. A port message's body can contain only `NSPort` objects or `NSData` objects. In the distributed objects system the byte/character arrays are usually encoded `NSInvocation` objects that are being forwarded from a proxy to the corresponding real object.

An `NSPortMessage` object also maintains a message identifier, which can be used to indicate the class of a message, such as an Objective-C method invocation, a connection request, an error, and so on. Use the `setMsgid:` (page 1267) and `msgid` (page 1265) methods to access the identifier.

Tasks

Creating Instances

- `initWithSendPort:receivePort:components:` (page 1265)
  Initializes a newly allocated `NSPortMessage` object to send given data on a given port and to receiver replies on another given port.

Sending the Message

- `sendBeforeDate:` (page 1266)
  Attempts to send the message before *`aDate`*, returning `YES` if successful or `NO` if the operation times out.

Getting the Components

- `components` (page 1264)
  Returns the data components of the receiver.

Getting the Ports

- `receivePort` (page 1266)
  For an outgoing message, returns the port on which replies to the receiver will arrive. For an incoming message, returns the port the receiver did arrive on.
- `sendPort` (page 1267)
  For an outgoing message, returns the port the receiver will send itself through. For an incoming message, returns the port replies to the receiver should be sent through.

Accessing the Message ID

- `setMsgid:` (page 1267)
  Sets the identifier for the receiver.
- `msgid` (page 1265)
  Returns the identifier for the receiver.

Instance Methods

components

Returns the data components of the receiver.

```
- (NSArray *)components
```

Return Value
The data components of the receiver. See "Class Description" (page 1263) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSPortMessage.h

initWithSendPort:receivePort:components:

Initializes a newly allocated NSPortMessage object to send given data on a given port and to receiver replies on another given port.

```
- (id)initWithSendPort:(NSPort *)sendPort receivePort:(NSPort *)receivePort
    components:(NSArray *)components
```

Parameters

sendPort
The port on which the message is sent.

receivePort
The port on which replies to the message arrive.

components
The data to send in the message. *components* should contain only NSData and NSPort objects, and the contents of the NSData objects should be in network byte order.

Return Value
An NSPortMessage object initialized to send *components* on *sendPort* and to receiver replies on *receivePort*.

Discussion
An NSPortMessage object initialized with this method has a message identifier of 0.

This is the designated initializer for NSPortMessage.

Availability
Available in Mac OS X v10.0 and later.

See Also
- setMsgid: (page 1267)

Declared In
NSPortMessage.h

msgid

Returns the identifier for the receiver.

```
- (uint32_t)msgid
```

Return Value
The identifier for the receiver.

Discussion
Cooperating applications can use this to define different types of messages, such as connection requests, RPCs, errors, and so on.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setMsgid:` (page 1267)

Declared In
`NSPortMessage.h`

receivePort

For an outgoing message, returns the port on which replies to the receiver will arrive. For an incoming message, returns the port the receiver did arrive on.

```
- (NSPort *)receivePort
```

Return Value
For an outgoing message, the port on which replies to the receiver will arrive. For an incoming message, the port the receiver did arrive on.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- sendPort` (page 1267)

Declared In
`NSPortMessage.h`

sendBeforeDate:

Attempts to send the message before *aDate*, returning `YES` if successful or `NO` if the operation times out.

```
- (BOOL)sendBeforeDate:(NSDate *)aDate
```

Parameters

aDate

The instant before which the message should be sent.

Return Value
`YES` if the operation is successful, otherwise `NO` (for example, if the operation times out).

Discussion
If an error other than a time out occurs, this method could raise an `NSInvalidSendPortException`, `NSInvalidReceivePortException`, or an `NSPortSendException`, depending on the type of send port and the type of error.

If the message cannot be sent immediately, the sending thread blocks until either the message is sent or *aDate* is reached. Sent messages are queued to minimize blocking, but failure can occur if multiple messages are sent to a port faster than the port's owner can receive them, causing the queue to fill up. Therefore, select a value for *aDate* that provides enough time for the message to be processed before the next message is sent. See the `NSPort` class specification for information on receiving a port message.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSPortMessage.h`

sendPort

For an outgoing message, returns the port the receiver will send itself through. For an incoming message, returns the port replies to the receiver should be sent through.

```
- (NSPort *)sendPort
```

Return Value
For an outgoing message, the port the receiver will send itself through when it receives a `sendBeforeDate:` (page 1266) message. For an incoming message, the port replies to the receiver should be sent through.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- receivePort` (page 1266)

Declared In
`NSPortMessage.h`

setMsgid:

Sets the identifier for the receiver.

```
- (void)setMsgid:(uint32_t)msgid
```

Parameters
msgid
The identifier for the receiver.

Discussion
Cooperating applications can use this method to define different types of messages, such as connection requests, RPCs, errors, and so on.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- msgid` (page 1265)

Declared In
`NSPortMessage.h`

NSPortNameServer Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSPortNameServer.h
Companion guide	Distributed Objects Programming Topics

Overview

`NSPortNameServer` provides an object-oriented interface to the port registration service used by the distributed objects system. `NSConnection` objects use it to contact each other and to distribute objects over the network; you should rarely need to interact directly with an `NSPortNameServer`.

You get an `NSPortNameServer` object by using the `systemDefaultPortNameServer` (page 1270) class method—never allocate and initialize an instance directly. With the default server object you can register an `NSPort` object under a given name, making it available on the network, and also unregister it so that it can't be looked up (although other applications that have already looked up the `NSPort` object can still use it until it becomes invalid). See the `NSPort` class specification for more information.

Tasks

Getting the Server Object

`+ systemDefaultPortNameServer` (page 1270)
Returns the single instance of `NSPortNameServer` for the application.

Looking Up Ports

`- portForName:` (page 1270)
Looks up and returns the port registered under the specified name on the local host.

`- portForName:host:` (page 1271)
Looks up and returns the port registered under the specified name on a specified host.

Registering Ports

- `registerPort:name:` (page 1271)
  Makes a given port available on the network under a specified name.
- `removePortForName:` (page 1272)
  Unregisters the port for a given name on the local host.

Class Methods

systemDefaultPortNameServer

Returns the single instance of `NSPortNameServer` for the application.

```
+ (NSPortNameServer *)systemDefaultPortNameServer
```

Return Value
The single instance of `NSPortNameServer` for the application.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSPortNameServer.h`

Instance Methods

portForName:

Looks up and returns the port registered under the specified name on the local host.

```
- (NSPort *)portForName:(NSString *)portName
```

Parameters
portName
The name of the desired port.

Return Value
The port associated with *portName* on the local host. Returns `nil` if no such port exists.

Discussion
Invokes `portForName:host:` (page 1271) with `nil` as the host name.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `portForName:host:` (page 1271)

Declared In
NSPortNameServer.h

portForName:host:

Looks up and returns the port registered under the specified name on a specified host.

```
- (NSPort *)portForName:(NSString *)portName host:(NSString *)hostName
```

Parameters

portName
The name of the desired port.

hostName
The name of the host. *hostName* is an Internet domain name (for example, "sales.anycorp.com"). If *hostName* is nil or empty, the local host is checked.

Return Value
The port associated with *portName* on the host *hostName*. Returns nil if no such port exists.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSPortNameServer.h

registerPort:name:

Makes a given port available on the network under a specified name.

```
- (BOOL)registerPort:(NSPort *)aPort name:(NSString *)portName
```

Parameters

aPort
The port to make available.

portName
The name for the port.

Return Value
YES if successful, NO otherwise (for example, if another NSPort object has already been registered under *portName*).

Discussion
A port can be registered under multiple names. If it is, it must be unregistered for each name with removePortForName: (page 1272) to make it completely unavailable.

Availability
Available in Mac OS X v10.2 and later.

Declared In
NSPortNameServer.h

removePortForName:

Unregisters the port for a given name on the local host.

```
- (BOOL)removePortForName:(NSString *)portName
```

Parameters

portName

The name of the port to unregister.

Return Value

`YES` if successful, otherwise `NO`.

Discussion

If the operation is successful, the port can no longer be looked up using the name *portName*. Other applications that already have a reference to the port can continue to use it until it becomes invalid.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSPortNameServer.h`

NSPositionalSpecifier Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptObjectSpecifiers.h
Companion guide	Cocoa Scripting Guide

Overview

Instances of `NSPositionalSpecifier` specify an insertion point in a container relative to another object in the container, for example, `before first word` or `after paragraph 4`. The container is specified by an instance of `NSScriptObjectSpecifier`. `NSPositionalSpecifier` objects commonly encapsulate object specifiers used as arguments to the `make` (`create`) and `move` commands and indicate where the created or moved object is to be inserted relative to the object represented by an object specifier.

Invoking an accessor method to obtain information about an instance of `NSPositionalSpecifier` causes the object to be evaluated if it hasn't been already.

You don't normally subclass `NSPositionalSpecifier`.

Tasks

Initializing a Positional Specifier

`- initWithPosition:objectSpecifier:` (page 1274)
Initializes a positional specifier with a given position relative to another given specifier.

Accessing Information About a Positional Specifier

`- insertionContainer` (page 1275)
Returns the container in which the new or copied object or objects should be placed.

- `insertionIndex` (page 1275)
  Returns an insertion index that indicates where the new or copied object or objects should be placed.
- `insertionKey` (page 1275)
  Returns the key that identifies the relationship into which the new or copied object or objects should be inserted.
- `insertionReplaces` (page 1276)
  Returns a Boolean value that indicates whether evaluation has been successful and the object to be inserted should actually replace the keyed, indexed object in the insertion container.
- `objectSpecifier` (page 1276)
  Returns the object specifier specified at initialization time.
- `position` (page 1276)
  Returns the insertion position specified at initialization time.
- `setInsertionClassDescription:` (page 1277)
  Sets the class description for the object or objects to be inserted.

Evaluating a Positional Specifier

- `evaluate` (page 1274)
  Causes the receiver to evaluate its position.

Instance Methods

evaluate

Causes the receiver to evaluate its position.

```
- (void)evaluate
```

Discussion
Calling `insertionContainer` (page 1275), `insertionKey` (page 1275), `insertionIndex` (page 1275), or `insertionReplaces` (page 1276) also causes the receiver to be evaluated, if it hasn't already been evaluated.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptObjectSpecifiers.h`

initWithPosition:objectSpecifier:

Initializes a positional specifier with a given position relative to another given specifier.

```
- (id)initWithPosition:(NSInsertionPosition)position
    objectSpecifier:(NSScriptObjectSpecifier *)specifier
```

Parameters

position

The position for the new specifier relative to *specifier*.

specifier

The reference specifier.

Return Value

An initialized positional specifier with the position specified by *position* relative to the object specified by *specifier*.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSScriptObjectSpecifiers.h

insertionContainer

Returns the container in which the new or copied object or objects should be placed.

```
- (id)insertionContainer
```

Return Value

A container. Determined by evaluating the receiver.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSScriptObjectSpecifiers.h

insertionIndex

Returns an insertion index that indicates where the new or copied object or objects should be placed.

```
- (NSInteger)insertionIndex
```

Return Value

An insertion index. Determined by evaluating the receiver.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSScriptObjectSpecifiers.h

insertionKey

Returns the key that identifies the relationship into which the new or copied object or objects should be inserted.

```
- (NSString *)insertionKey
```

Return Value
A key. Determined by evaluating the receiver.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptObjectSpecifiers.h`

insertionReplaces

Returns a Boolean value that indicates whether evaluation has been successful and the object to be inserted should actually replace the keyed, indexed object in the insertion container.

```
- (BOOL)insertionReplaces
```

Return Value
`YES` if evaluation has been successful and the object to be inserted should actually replace the keyed, indexed object in the insertion container, instead of being inserted before it; `NO` otherwise.

Discussion
If this object has never been evaluated, evaluation is attempted.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`NSScriptObjectSpecifiers.h`

objectSpecifier

Returns the object specifier specified at initialization time.

```
- (NSScriptObjectSpecifier *)objectSpecifier
```

Return Value
An object specifier for a container.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSScriptObjectSpecifiers.h`

position

Returns the insertion position specified at initialization time.

```
- (NSInsertionPosition)position
```

Return Value
An insertion position.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSScriptObjectSpecifiers.h`

setInsertionClassDescription:

Sets the class description for the object or objects to be inserted.

```
- (void)setInsertionClassDescription:(NSScriptClassDescription *)classDescription
```

Parameters

classDescription
The class description for the object or objects to be inserted.

Discussion
This message can be sent at any time after object initialization, but must be sent before evaluation to have any effect.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`NSScriptObjectSpecifiers.h`

Constants

NSInsertionPosition

The following constants are defined by `NSPositionalSpecifier` to specify an insertion position.

```
typedef enum {
    NSPositionAfter,
    NSPositionBefore,
    NSPositionBeginning,
    NSPositionEnd,
    NSPositionReplace
} NSInsertionPosition;
```

Constants

`NSPositionAfter`
Specifies a position after another object.
Available in Mac OS X v10.0 and later.
Declared in `NSScriptObjectSpecifiers.h`.

`NSPositionBefore`
Specifies a position before another object.
Available in Mac OS X v10.0 and later.
Declared in `NSScriptObjectSpecifiers.h`.

`NSPositionBeginning`
: Specifies a position at the beginning of a collection.

  Available in Mac OS X v10.0 and later.

  Declared in `NSScriptObjectSpecifiers.h`.

`NSPositionEnd`
: Specifies a position at the end of a collection.

  Available in Mac OS X v10.0 and later.

  Declared in `NSScriptObjectSpecifiers.h`.

`NSPositionReplace`
: Specifies a position in the place of another object.

  Available in Mac OS X v10.0 and later.

  Declared in `NSScriptObjectSpecifiers.h`.

Discussion
These constants are described in NSPositionalSpecifier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptObjectSpecifiers.h`

NSPredicate Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSPredicate.h
Availability	Available in Mac OS X v10.4 and later.
Companion guide	Predicate Programming Guide
Related sample code	CoreRecipes DerivedProperty iSpend PredicateEditorSample SimpleCalendar

Overview

The `NSPredicate` class is used to define logical conditions used to constrain a search either for a fetch or for in-memory filtering.

You use predicates to represent logical conditions, used for describing objects in persistent stores and in-memory filtering of objects. Although it is common to create predicates directly from instances of `NSComparisonPredicate`, `NSCompoundPredicate`, and `NSExpression`, you often create predicates from a format string which is parsed by the class methods on `NSPredicate`. Examples of predicate format strings include:

- Simple comparisons, such as `grade == "7"` or `firstName like "Shaffiq"`
- Case/diacritic insensitive lookups, such as `name contains[cd] "itroen"`
- Logical operations, such as `(firstName like "Mark") OR (lastName like "Adderley")`
- With Mac OS X version 10.5 and later, you can create "between" predicates such as `date between {$YESTERDAY, $TOMORROW}`.

You can create predicates for relationships, such as:

- `group.name like "work*"`

- `ALL children.age > 12`
- `ANY children.age > 12`

You can create predicates for operations, such as `@sum.items.price < 1000`. For a complete syntax reference, refer to the *Predicate Programming Guide*.

You can also create predicates that include variables, so that the predicate can be pre-defined before substituting concrete values at runtime. On Mac OS X v10.4, for predicates that use variables, evaluation is a two step process (see `predicateWithSubstitutionVariables:` (page 1284) and `evaluateWithObject:` (page 1283)). Mac OS X v10.5 introduces a new method, `evaluateWithObject:substitutionVariables:` (page 1283), which combines these steps.

Tasks

Constructors

`+ predicateWithFormat:` (page 1281)
Creates and returns a new predicate formed by creating a new string with a given format and parsing the result.

`+ predicateWithFormat:argumentArray:` (page 1281)
Creates and returns a new predicate by substituting the values in a given array into a format string and parsing the result.

`+ predicateWithFormat:arguments:` (page 1282)
Creates and returns a new predicate by substituting the values in an argument list into a format string and parsing the result.

`- predicateWithSubstitutionVariables:` (page 1284)
Returns a copy of the receiver with the receiver's variables substituted by values specified in a given substitution variables dictionary.

`+ predicateWithValue:` (page 1282)
Creates and returns a predicate that always evaluates to a given value.

Evaluating a Predicate

`- evaluateWithObject:` (page 1283)
Returns a Boolean value that indicates whether a given object matches the conditions specified by the receiver.

`- evaluateWithObject:substitutionVariables:` (page 1283)
Returns a Boolean value that indicates whether a given object matches the conditions specified by the receiver after substituting in the values in a given variables dictionary.

Getting Format Information

`- predicateFormat` (page 1284)
Returns the receiver's format string.

Class Methods

predicateWithFormat:

Creates and returns a new predicate formed by creating a new string with a given format and parsing the result.

```
+ (NSPredicate *)predicateWithFormat:(NSString *)format, ...
```

Parameters

format

The format string for the new predicate.

...

A comma-separated list of arguments to substitute into *format*.

Return Value

A new predicate formed by creating a new string with *format* and parsing the result.

Discussion

For details of the format of the format string and of limitations on variable substitution, see Predicate Format String Syntax.

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

CoreRecipes

iSpend

QTMetadataEditor

SimpleCalendar

SpotlightFortunes

Declared In

NSPredicate.h

predicateWithFormat:argumentArray:

Creates and returns a new predicate by substituting the values in a given array into a format string and parsing the result.

```
+ (NSPredicate *)predicateWithFormat:(NSString *)predicateFormat
    argumentArray:(NSArray *)arguments
```

Parameters

predicateFormat

The format string for the new predicate.

arguments

The arguments to substitute into *predicateFormat*. Values are substituted into *predicateFormat* in the order they appear in the array.

Return Value
A new predicate by substituting the values in *arguments* into *predicateFormat*, and parsing the result.

Discussion
For details of the format of the format string and of limitations on variable substitution, see Predicate Format String Syntax.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSPredicate.h

predicateWithFormat:arguments:

Creates and returns a new predicate by substituting the values in an argument list into a format string and parsing the result.

```
+ (NSPredicate *)predicateWithFormat:(NSString *)format arguments:(va_list)argList
```

Parameters

format
The format string for the new predicate.

argList
The arguments to substitute into *predicateFormat*. Values are substituted into *predicateFormat* in the order they appear in the argument list.

Return Value
A new predicate by substituting the values in *argList* into *predicateFormat* and parsing the result.

Discussion
For details of the format of the format string and of limitations on variable substitution, see Predicate Format String Syntax.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSPredicate.h

predicateWithValue:

Creates and returns a predicate that always evaluates to a given value.

```
+ (NSPredicate *)predicateWithValue:(BOOL)value
```

Parameters

value
The value to which the new predicate should evaluate.

Return Value
A predicate that always evaluates to *value*.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
CoreRecipes

Declared In
`NSPredicate.h`

Instance Methods

evaluateWithObject:

Returns a Boolean value that indicates whether a given object matches the conditions specified by the receiver.

```
- (BOOL)evaluateWithObject:(id)object
```

Parameters

object

The object against which to evaluate the receiver.

Return Value
`YES` if *object* matches the conditions specified by the receiver, otherwise `NO`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSPredicate.h`

evaluateWithObject:substitutionVariables:

Returns a Boolean value that indicates whether a given object matches the conditions specified by the receiver after substituting in the values in a given variables dictionary.

```
- (BOOL)evaluateWithObject:(id)object
    substitutionVariables:(NSDictionary *)variables
```

Parameters

object

The object against which to evaluate the receiver.

variables

The substitution variables dictionary. The dictionary must contain key-value pairs for all variables in the receiver.

Return Value
`YES` if *object* matches the conditions specified by the receiver after substituting in the values in *variables* for any replacement tokens, otherwise `NO`.

Discussion
This method returns the same result as the two step process of first invoking `predicateWithSubstitutionVariables:` (page 1284) on the receiver and then invoking `evaluateWithObject:` (page 1283) on the returned predicate. This method is optimized for situations which require repeatedly evaluating a predicate with substitution variables with different variable substitutions.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSPredicate.h`

predicateFormat

Returns the receiver's format string.

```
- (NSString *)predicateFormat
```

Return Value
The receiver's format string.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSPredicate.h`

predicateWithSubstitutionVariables:

Returns a copy of the receiver with the receiver's variables substituted by values specified in a given substitution variables dictionary.

```
- (NSPredicate *)predicateWithSubstitutionVariables:(NSDictionary *)variables
```

Parameters
variables
The substitution variables dictionary. The dictionary must contain key-value pairs for all variables in the receiver.

Return Value
A copy of the receiver with the receiver's variables substituted by values specified in *variables*.

Discussion
The receiver itself is not modified by this method, so you can reuse it for any number of substitutions.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
DerivedProperty

Declared In
`NSPredicate.h`

NSProcessInfo Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSProcessinfo.h
Companion guide	Interacting with the Operating System
Related sample code	CocoaEcho Quartz Composer WWDC 2005 TextEdit Sproing TextEditPlus URL CacheInfo

Overview

The `NSProcessInfo` class provides methods to access information about the current process. Each process has a single, shared `NSProcessInfo` object, known as **process information agent**.

The process information agent can return such information as the arguments, environment variables, host name, or process name. The `processInfo` (page 1287) class method returns the shared agent for the current process—that is, the process whose object sent the message. For example, the following line returns the `NSProcessInfo` object, which then provides the name of the current process:

```
NSString *processName = [[NSProcessInfo processInfo] processName];
```

The `NSProcessInfo` class also includes the `operatingSystem` (page 1289) method, which returns an `enum` constant identifying the operating system on which the process is executing.

`NSProcessInfo` objects attempt to interpret environment variables and command-line arguments in the user's default C string encoding if they cannot be converted to Unicode as UTF-8 strings. If neither conversion works, these values are ignored by the `NSProcessInfo` object.

Tasks

Getting the Process Information Agent

+ `processInfo` (page 1287)
  Returns the process information agent for the process.

Accessing Process Information

- `arguments` (page 1288)
  Returns the command-line arguments for the process.
- `environment` (page 1288)
  Returns the variable names and their values in the environment from which the process was launched.
- `processIdentifier` (page 1290)
  Returns the identifier of the process.
- `globallyUniqueString` (page 1288)
  Returns a global unique identifier for the process.
- `processName` (page 1291)
  Returns the name of the process.
- `setProcessName:` (page 1291)
  Sets the name of the process.

Getting Host Information

- `hostName` (page 1289)
  Returns the name of the host computer.
- `operatingSystem` (page 1289)
  Returns a constant to indicate the operating system on which the process is executing.
- `operatingSystemName` (page 1289)
  Returns a string containing the name of the operating system on which the process is executing.
- `operatingSystemVersionString` (page 1290)
  Returns a string containing the version of the operating system on which the process is executing.

Getting Computer Information

- `physicalMemory` (page 1290)
  Provides the amount of physical memory on the computer.
- `processorCount` (page 1291)
  Provides the number of processing cores available on the computer.
- `activeProcessorCount` (page 1287)
  Provides the number of active processing cores available on the computer.

Class Methods

processInfo

Returns the process information agent for the process.

```
+ (NSProcessInfo *)processInfo
```

Return Value
Shared process information agent for the process.

Discussion
An `NSProcessInfo` (page 1285) object is created the first time this method is invoked, and that same object is returned on each subsequent invocation.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CocoaEcho
CocoaHTTPServer
Quartz Composer WWDC 2005 TextEdit
TextEditPlus
URL CacheInfo

Declared In
`NSProcessInfo.h`

Instance Methods

activeProcessorCount

Provides the number of active processing cores available on the computer.

```
- (NSUInteger)activeProcessorCount
```

Return Value
Number of active processing cores.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- processorCount` (page 1291)

Declared In
`NSProcessInfo.h`

arguments

Returns the command-line arguments for the process.

```
- (NSArray *)arguments
```

Return Value
Array of strings with the process's command-line arguments.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSProcessInfo.h

environment

Returns the variable names and their values in the environment from which the process was launched.

```
- (NSDictionary *)environment
```

Return Value
Dictionary of environment-variable names (keys) and their values.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSProcessInfo.h

globallyUniqueString

Returns a global unique identifier for the process.

```
- (NSString *)globallyUniqueString
```

Return Value
Global ID for the process. The ID includes the host name, process ID, and a time stamp, which ensures that the ID is unique for the network.

Discussion
This method generates a new string each time it is invoked, so it also uses a counter to guarantee that strings created from the same process are unique.

Availability
Available in Mac OS X v10.0 and later.

See Also
- processName (page 1291)

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
NSProcessInfo.h

hostName

Returns the name of the host computer.

```
- (NSString *)hostName
```

Return Value
Host name of the computer.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSProcessInfo.h

operatingSystem

Returns a constant to indicate the operating system on which the process is executing.

```
- (unsigned int)operatingSystem
```

Return Value
Operating system identifier. See "Constants" (page 1292) for a list of possible values. In Mac OS X, it's `NSMACHOperatingSystem`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSProcessInfo.h

operatingSystemName

Returns a string containing the name of the operating system on which the process is executing.

```
- (NSString *)operatingSystemName
```

Return Value
Operating system name. In Mac OS X, it's `@"NSMACHOperatingSystem"`

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSProcessInfo.h

operatingSystemVersionString

Returns a string containing the version of the operating system on which the process is executing.

```
- (NSString *)operatingSystemVersionString
```

Return Value
Operating system version. This string is human readable, localized, and is appropriate for displaying to the user. This string is *not* appropriate for parsing.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`NSProcessInfo.h`

physicalMemory

Provides the amount of physical memory on the computer.

```
- (unsigned long long)physicalMemory
```

Return Value
Amount of physical memory in bytes.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSProcessInfo.h`

processIdentifier

Returns the identifier of the process.

```
- (int)processIdentifier
```

Return Value
Process ID of the process.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- processName` (page 1291)

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSProcessInfo.h`

processName

Returns the name of the process.

```
- (NSString *)processName
```

Return Value
Name of the process.

Discussion
The process name is used to register application defaults and is used in error messages. It does not uniquely identify the process.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `processIdentifier` (page 1290)
- `setProcessName:` (page 1291)

Related Sample Code
URL CacheInfo

Declared In
`NSProcessInfo.h`

processorCount

Provides the number of processing cores available on the computer.

```
- (NSUInteger)processorCount
```

Return Value
Number of processing cores.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `activeProcessorCount` (page 1287)

Declared In
`NSProcessInfo.h`

setProcessName:

Sets the name of the process.

```
- (void)setProcessName:(NSString *)name
```

Parameters

name
New name for the process.

Discussion

Warning: User defaults and other aspects of the environment might depend on the process name, so be very careful if you change it. Setting the process name in this manner is not thread safe.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- processName` (page 1291)

Declared In
`NSProcessInfo.h`

Constants

NSProcessInfo—Operating Systems

The following constants are provided by the `NSProcessInfo` class as return values for `operatingSystem` (page 1289).

```
enum {
   NSWindowsNTOperatingSystem = 1,
   NSWindows95OperatingSystem,
   NSSolarisOperatingSystem,
   NSHPUXOperatingSystem,
   NSMACHOperatingSystem,
   NSSunOSOperatingSystem,
   NSOSF1OperatingSystem
};
```

Constants

`NSHPUXOperatingSystem`
Indicates the HP UX operating system.
Available in Mac OS X v10.0 and later.
Declared in `NSProcessInfo.h`.

`NSMACHOperatingSystem`
Indicates the Mac OS X operating system.
Available in Mac OS X v10.0 and later.
Declared in `NSProcessInfo.h`.

`NSOSF1OperatingSystem`
Indicates the OSF/1 operating system.
Available in Mac OS X v10.0 and later.
Declared in `NSProcessInfo.h`.

`NSSolarisOperatingSystem`
Indicates the Solaris operating system.

Available in Mac OS X v10.0 and later.

Declared in `NSProcessInfo.h`.

`NSSunOSOperatingSystem`
Indicates the Sun OS operating system.

Available in Mac OS X v10.0 and later.

Declared in `NSProcessInfo.h`.

`NSWindows95OperatingSystem`
Indicates the Windows 95 operating system.

Available in Mac OS X v10.0 and later.

Declared in `NSProcessInfo.h`.

`NSWindowsNTOperatingSystem`
Indicates the Windows NT operating system.

Available in Mac OS X v10.0 and later.

Declared in `NSProcessInfo.h`.

Declared In
`NSProcessInfo.h`

NSPropertyListSerialization Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSPropertyList.h
Availability	Available in Mac OS X v10.2 and later.
Companion guides	Archives and Serializations Programming Guide for Cocoa Property List Programming Guide
Related sample code	EnhancedAudioBurn iSpend People Sketch-112 Squiggles

Overview

The `NSPropertyListSerialization` class provides methods that convert property list objects to and from several serialized formats. Property list objects include `NSData`, `NSString`, `NSArray`, `NSDictionary`, `NSDate`, and `NSNumber` objects. These objects are toll-free bridged with their respective Core Foundation types (`CFData`, `CFString`, and so on). For more about toll-free bridging, see Interchangeable Data Types.

Property list serialization automatically takes account of endianness on different platforms—for example, you can correctly read on an Intel-based Macintosh a binary property list created on a PowerPC-based Macintosh.

Tasks

Serializing a Property List

`+ dataFromPropertyList:format:errorDescription:` (page 1296)
Returns an `NSData` object containing a given property list in a specified format.

Deserializing a Property List

`+ propertyListFromData:mutabilityOption:format:errorDescription:` (page 1297)
Returns a property list object corresponding to the representation in a given `NSData` object.

Validating a Property List

`+ propertyList:isValidForFormat:` (page 1297)
Returns a Boolean value that indicates whether a given property list is valid for a given format.

Class Methods

dataFromPropertyList:format:errorDescription:

Returns an `NSData` object containing a given property list in a specified format.

```
+ (NSData *)dataFromPropertyList:(id)plist format:(NSPropertyListFormat)format
    errorDescription:(NSString **)errorString
```

Parameters

plist
A property list object. *plist* must be a kind of `NSData`, `NSString`, `NSNumber`, `NSDate`, `NSArray`, or `NSDictionary` object. Container objects must also contain only these kinds of objects.

format
A property list format. Possible values for *format* are described in NSPropertyListFormat (page 1299).

errorString
Upon return, if the conversion is successful, *errorString* is `nil`. If the conversion fails, upon return contains a string describing the nature of the error. If you receive a string, you must release it.

Return Value
An `NSData` object containing *plist* in the format specified by *format*.

Special Considerations

Unlike the normal memory management rules for Cocoa, strings returned in *errorString* need to be released by the caller.

Availability
Available in Mac OS X v10.2 and later.

See Also
`+ propertyListFromData:mutabilityOption:format:errorDescription:` (page 1297)

Related Sample Code
EnhancedAudioBurn
iSpend
People
SpotlightFortunes
Squiggles

Declared In
NSPropertyList.h

propertyList:isValidForFormat:

Returns a Boolean value that indicates whether a given property list is valid for a given format.

```
+ (BOOL)propertyList:(id)plist isValidForFormat:(NSPropertyListFormat)format
```

Parameters

plist
A property list object.

format
A property list format. Possible values for *format* are listed in NSPropertyListFormat (page 1299).

Return Value
YES if *plist* is a valid property list in format *format*, otherwise NO.

Availability
Available in Mac OS X v10.2 and later.

Declared In
NSPropertyList.h

propertyListFromData:mutabilityOption:format:errorDescription:

Returns a property list object corresponding to the representation in a given NSData object.

```
+ (id)propertyListFromData:(NSData *)data
    mutabilityOption:(NSPropertyListMutabilityOptions)opt
    format:(NSPropertyListFormat *)format errorDescription:(NSString **)errorString
```

Parameters

data
A data object containing a serialized property list.

opt
Determines whether the property list's contents are created as mutable objects, where possible. Possible values are described in NSPropertyListMutabilityOptions (page 1298).

format
If the property list is valid, upon return contains the format. *format* can be NULL, in which case the property list format is not returned. Possible values are described in NSPropertyListFormat (page 1299).

errorString
Upon return, if the conversion is successful, *errorString* is nil. If the conversion fails, upon return contains a string describing the nature of the error. If you receive a string, you must release it.

Return Value
A property list object corresponding to the representation in *data*. If data is not in a supported format, returns nil.

Special Considerations

Unlike the normal memory management rules for Cocoa, strings returned in *errorString* need to be released by the caller.

Availability

Available in Mac OS X v10.2 and later.

See Also

+ dataFromPropertyList:format:errorDescription: (page 1296)

Related Sample Code

ImageBrowser
iSpend
People
Sketch-112
Squiggles

Declared In

NSPropertyList.h

Constants

NSPropertyListMutabilityOptions

These constants specify mutability options in property lists.

```
typedef enum {
    NSPropertyListImmutable = kCFPropertyListImmutable,
    NSPropertyListMutableContainers = kCFPropertyListMutableContainers,
    NSPropertyListMutableContainersAndLeaves =
kCFPropertyListMutableContainersAndLeaves
} NSPropertyListMutabilityOptions;
```

Constants

NSPropertyListImmutable

Causes the returned property list to contain immutable objects.

Available in Mac OS X v10.2 and later.

Declared in NSPropertyList.h.

NSPropertyListMutableContainers

Causes the returned property list to have mutable containers but immutable leaves.

Available in Mac OS X v10.2 and later.

Declared in NSPropertyList.h.

NSPropertyListMutableContainersAndLeaves

Causes the returned property list to have mutable containers and leaves.

Available in Mac OS X v10.2 and later.

Declared in NSPropertyList.h.

Availability

Available in Mac OS X v10.2 and later.

Declared In
NSPropertyList.h

NSPropertyListFormat

These constants are used to specify a property list serialization format.

```
typedef enum {
    NSPropertyListOpenStepFormat = kCFPropertyListOpenStepFormat,
    NSPropertyListXMLFormat_v1_0 = kCFPropertyListXMLFormat_v1_0,
    NSPropertyListBinaryFormat_v1_0 = kCFPropertyListBinaryFormat_v1_0
} NSPropertyListFormat;
```

Constants

NSPropertyListOpenStepFormat
Specifies the old-style ASCII property list format inherited from the OpenStep APIs.

Important: The NSPropertyListOpenStepFormat constant is not supported for writing. It can be used only for reading old-style property lists.

Available in Mac OS X v10.2 and later.

Declared in NSPropertyList.h.

NSPropertyListXMLFormat_v1_0
Specifies the XML property list format.

Available in Mac OS X v10.2 and later.

Declared in NSPropertyList.h.

NSPropertyListBinaryFormat_v1_0
Specifies the binary property list format.

Available in Mac OS X v10.2 and later.

Declared in NSPropertyList.h.

Availability
Available in Mac OS X v10.2 and later.

Declared In
NSPropertyList.h

NSPropertySpecifier Class Reference

Inherits from	NSScriptObjectSpecifier : NSObject
Conforms to	NSCoding (NSScriptObjectSpecifier) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptObjectSpecifiers.h
Companion guide	Cocoa Scripting Guide

Overview

Specifies a simple attribute value, a one-to-one relationship, or all elements of a to-many relationship. You don't normally subclass `NSPropertySpecifier`.

NSProtocolChecker Class Reference

Inherits from	NSProxy
Conforms to	NSObject (NSProxy)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSProtocolChecker.h
Companion guide	Distributed Objects Programming Topics

Overview

The NSProtocolChecker class defines an object that restricts the messages that can be sent to another object (referred to as the checker's delegate). This fact can be particularly useful when an object with many methods, only a few of which ought to be remotely accessible, is made available using the distributed objects system.

A protocol checker acts as a kind of proxy; when it receives a message that is in its designated protocol, it forwards the message to its target and consequently appears to be the target object itself. However, when it receives a message not in its protocol, it raises an `NSInvalidArgumentException` to indicate that the message isn't allowed, whether or not the target object implements the method.

Typically, an object that is to be distributed (yet must restrict messages) creates an NSProtocolChecker for itself and returns the checker rather than returning itself in response to any messages. The object might also register the checker as the root object of an NSConnection.

The object should be careful about vending references to `self`—the protocol checker will convert a return value of `self` to indicate the checker rather than the object for any messages forwarded by the checker, but direct references to the object (bypassing the checker) could be passed around by other objects.

Tasks

Creating a Checker

`+ protocolCheckerWithTarget:protocol:` (page 1304)
Allocates and initializes an NSProtocolChecker instance that will forward any messages in *aProtocol* to *anObject*, the protocol checker's target.

- `initWithTarget:protocol:` (page 1304)
  Initializes a newly allocated NSProtocolChecker instance that will forward any messages in *aProtocol* to *anObject*, the protocol checker's target.

Getting Information

- `protocol` (page 1305)
  Returns the protocol object the receiver uses.
- `target` (page 1305)
  Returns the target of the receiver.

Class Methods

protocolCheckerWithTarget:protocol:

Allocates and initializes an NSProtocolChecker instance that will forward any messages in *aProtocol* to *anObject*, the protocol checker's target.

```
+ (id)protocolCheckerWithTarget:(NSObject *)anObject protocol:(Protocol *)aProtocol
```

Discussion
Thus, the checker can be vended in lieu of *anObject* to restrict the messages that can be sent to *anObject*. Returns the new instance.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSProtocolChecker.h`

Instance Methods

initWithTarget:protocol:

Initializes a newly allocated NSProtocolChecker instance that will forward any messages in *aProtocol* to *anObject*, the protocol checker's target.

```
- (id)initWithTarget:(NSObject *)anObject protocol:(Protocol *)aProtocol
```

Discussion
Thus, the checker can be vended in lieu of *anObject* to restrict the messages that can be sent to *anObject*. If *anObject* is allowed to be freed or dereferenced by clients, the `free` method should be included in *aProtocol*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSProtocolChecker.h

protocol

Returns the protocol object the receiver uses.

```
- (Protocol *)protocol
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSProtocolChecker.h

target

Returns the target of the receiver.

```
- (NSObject *)target
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSProtocolChecker.h

NSProxy Class Reference

Inherits from	none (NSProxy is a root class)
Conforms to	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSProxy.h
Companion guide	Distributed Objects Programming Topics

Overview

`NSProxy` is an abstract superclass defining an API for objects that act as stand-ins for other objects or for objects that don't exist yet. Typically, a message to a proxy is forwarded to the real object or causes the proxy to load (or transform itself into) the real object. Subclasses of `NSProxy` can be used to implement transparent distributed messaging (for example, `NSDistantObject`) or for lazy instantiation of objects that are expensive to create.

`NSProxy` implements the basic methods required of a root class, including those defined in the `NSObject` protocol. However, as an abstract class it doesn't provide an initialization method, and it raises an exception upon receiving any message it doesn't respond to. A concrete subclass must therefore provide an initialization or creation method and override the `forwardInvocation:` (page 1311) and `methodSignatureForSelector:` (page 1312) methods to handle messages that it doesn't implement itself. A subclass's implementation of `forwardInvocation:` (page 1311) should do whatever is needed to process the invocation, such as forwarding the invocation over the network or loading the real object and passing it the invocation. `methodSignatureForSelector:` (page 1312) is required to provide argument type information for a given message; a subclass's implementation should be able to determine the argument types for the messages it needs to forward and should construct an `NSMethodSignature` object accordingly. See the `NSDistantObject`, `NSInvocation`, and `NSMethodSignature` class specifications for more information.

Adopted Protocols

NSObject

- `autorelease` (page 2099)
- `class` (page 2100)
- `conformsToProtocol:` (page 2100)

- description (page 2100)
- hash (page 2101)
- isEqual: (page 2101)
- isKindOfClass: (page 2102)
- isMemberOfClass: (page 2103)
- isProxy (page 2104)
- performSelector: (page 2104)
- performSelector:withObject: (page 2105)
- performSelector:withObject:withObject: (page 2105)
- release (page 2106)
- respondsToSelector: (page 2107)
- retain (page 2108)
- retainCount (page 2109)
- self (page 2109)
- superclass (page 2110)
- zone (page 2110)

Tasks

Creating Instances

+ alloc (page 1309)
Returns a new instance of the receiving class

+ allocWithZone: (page 1309)
Returns a new instance of the receiving class

Deallocating Instances

- dealloc (page 1310)
Deallocates the memory occupied by the receiver.

Finalizing an Object

- finalize (page 1311)
The garbage collector invokes this method on the receiver before disposing of the memory it uses.

Handling Unimplemented Methods

- forwardInvocation: (page 1311)
Passes a given invocation to the real object the proxy represents.

- `methodSignatureForSelector:` (page 1312)
  Raises `NSInvalidArgumentException`. Override this method in your concrete subclass to return a proper `NSMethodSignature` object for the given selector and the class your proxy objects stand in for.

Introspecting a Proxy Class

+ `respondsToSelector:` (page 1310)
  Returns a Boolean value that indicates whether the receiving class responds to a given selector.

Describing a Proxy Class or Object

+ `class` (page 1310)
  Returns `self` (the class object).

- `description` (page 1311)
  Returns an `NSString` object containing the real class name and the `id` of the receiver as a hexadecimal number.

Class Methods

alloc

Returns a new instance of the receiving class

```
+ (id)alloc
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSProxy.h`

allocWithZone:

Returns a new instance of the receiving class

```
+ (id)allocWithZone:(NSZone *)zone
```

Return Value
A new instance of the receiving class, as described in the `NSObject` class specification under the `allocWithZone:` (page 1152) class method.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSProxy.h`

class

Returns `self` (the class object).

```
+ (Class)class
```

Return Value
`self`. Because this is a class method, it returns the class object

Availability
Available in Mac OS X v10.0 and later.

See Also
`class` (page 1155) (`NSObject`)
`class` (page 2100) (`NSObject` protocol)

Declared In
`NSProxy.h`

respondsToSelector:

Returns a Boolean value that indicates whether the receiving class responds to a given selector.

```
+ (BOOL)respondsToSelector:(SEL)aSelector
```

Parameters
aSelector
A selector.

Return Value
`YES` if the receiving class responds to *aSelector* messages, otherwise `NO`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSProxy.h`

Instance Methods

dealloc

Deallocates the memory occupied by the receiver.

```
- (void)dealloc
```

Discussion
This method behaves as described in the `NSObject` class specification under the `dealloc` (page 1174) instance method.

Availability
Available in Mac OS X v10.0 and later.

See Also
- finalize (page 1311)

Declared In
NSProxy.h

description

Returns an NSString object containing the real class name and the id of the receiver as a hexadecimal number.

```
- (NSString *)description
```

Return Value
An NSString object containing the real class name and the id of the receiver as a hexadecimal number.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSProxy.h

finalize

The garbage collector invokes this method on the receiver before disposing of the memory it uses.

```
- (void)finalize
```

Discussion
This method behaves as described in the NSObject class specification under the finalize (page 1176) instance method. Note that a finalize method must be thread-safe.

Availability
Available in Mac OS X v10.5 and later.

See Also
- dealloc (page 1310)

Declared In
NSProxy.h

forwardInvocation:

Passes a given invocation to the real object the proxy represents.

```
- (void)forwardInvocation:(NSInvocation *)anInvocation
```

Parameters
anInvocation
The invocation to forward.

Discussion
`NSProxy`'s implementation merely raises `NSInvalidArgumentException`. Override this method in your subclass to handle *`anInvocation`* appropriately, at the very least by setting its return value.

For example, if your proxy merely forwards messages to an instance variable named *`realObject`*, it can implement `forwardInvocation:` like this:

```
- (void)forwardInvocation:(NSInvocation *)anInvocation
{
    [anInvocation setTarget:realObject];
    [anInvocation invoke];
    return;
}
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSProxy.h`

methodSignatureForSelector:

Raises `NSInvalidArgumentException`. Override this method in your concrete subclass to return a proper `NSMethodSignature` object for the given selector and the class your proxy objects stand in for.

```
- (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector
```

Parameters
`aSelector`
The selector for which to return a method signature.

Return Value
Not applicable. The implementation provided by `NSProxy` raises an exception.

Discussion
Be sure to avoid an infinite loop when necessary by checking that *`aSelector`* isn't the selector for this method itself and by not sending any message that might invoke this method.

For example, if your proxy merely forwards messages to an instance variable named *`realObject`*, it can implement `methodSignatureForSelector:` like this:

```
- (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector
{
    return [realObject methodSignatureForSelector:aSelector];
}
```

Availability
Available in Mac OS X v10.0 and later.

See Also
`methodSignatureForSelector:` (page 1181) (`NSObject`)

Declared In
`NSProxy.h`

NSQuitCommand Class Reference

Inherits from	NSScriptCommand : NSObject
Conforms to	NSCoding (NSScriptCommand) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptStandardSuiteCommands.h
Companion guide	Cocoa Scripting Guide

Overview

An instance of `NSQuitCommand` quits the specified application. The command may optionally specify how to handle modified documents (automatically save changes, don't save them, or ask the user). For details, see the description for the `quit` command in "Apple Events Sent By the Mac OS" in How Cocoa Applications Handle Apple Events in*Cocoa Scripting Guide*.

`NSQuitCommand` is part of Cocoa's built-in scripting support. Most applications don't need to subclass `NSQuitCommand` or call its methods.

Tasks

Accessing Options

`- saveOptions` (page 1313)
Returns a constant indicating how to deal with closing any modified documents.

Instance Methods

saveOptions

Returns a constant indicating how to deal with closing any modified documents.

```
- (NSSaveOptions)saveOptions
```

Return Value
A constant indicating how to deal with closing any modified documents.

The default value returned is `NSSaveOptionsAsk`. See "Constants" in `NSCloseCommand` for a list of possible return values.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptStandardSuiteCommands.h`

NSRandomSpecifier Class Reference

Inherits from	NSScriptObjectSpecifier : NSObject
Conforms to	NSCoding (NSScriptObjectSpecifier) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptObjectSpecifiers.h
Companion guide	Cocoa Scripting Guide

Overview

Specifies an arbitrary object in a collection or, if not a one-to-many relationship, the sole object.

NSRangeSpecifier Class Reference

Inherits from	NSScriptObjectSpecifier : NSObject
Conforms to	NSCoding (NSScriptObjectSpecifier) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptObjectSpecifiers.h
Companion guide	Cocoa Scripting Guide
Related sample code	Sketch-112

Overview

An `NSRangeSpecifier` object specifies a range (that is, an uninterrupted series) of objects in a container through two delimiting objects. The range is represented by two object specifiers, a start specifier and an end specifier, which can be of any specifier type (such as `NSIndexSpecifier` or `NSWhoseSpecifier` object). These specifiers are evaluated in the context of the same container object as the range specifier itself.

You don't normally subclass `NSRangeSpecifier`.

Tasks

Initializing a Range Specifier

`- initWithContainerClassDescription:containerSpecifier:key:startSpecifier:endSpecifier:` (page 1318)

Returns a range specifier initialized with the given properties.

Accessing a Range Specifier

`- endSpecifier` (page 1318)

Returns the object specifier representing the last object of the range.

- `setEndSpecifier:` (page 1319)
  Sets the object specifier representing the last object of the range to a given object.
- `setStartSpecifier:` (page 1319)
  Sets the object specifier representing the first object of the range to a given object.
- `startSpecifier` (page 1319)
  Returns the object specifier representing the first object of the range.

Instance Methods

endSpecifier

Returns the object specifier representing the last object of the range.

```
- (NSScriptObjectSpecifier *)endSpecifier
```

Return Value
The object specifier representing the last object of the range.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Sketch-112

Declared In
`NSScriptObjectSpecifiers.h`

initWithContainerClassDescription:containerSpecifier:key:startSpecifier:endSpecifier:

Returns a range specifier initialized with the given properties.

```
- (id)initWithContainerClassDescription:(NSScriptClassDescription *)classDescription
    containerSpecifier:(NSScriptObjectSpecifier *)container key:(NSString *)property
    startSpecifier:(NSScriptObjectSpecifier *)startSpec
   endSpecifier:(NSScriptObjectSpecifier *)endSpec
```

Parameters

classDescription
The class description.

container
The container.

property
The property.

startSpec
The object specifier representing the first object of the range.

endSpec
The object specifier representing the last object of the range.

Return Value
A range specifier initialized with the given properties.

Discussion
Invokes the super class's `initWithContainerClassDescription:containerSpecifier:key:` (page 1418) method and initializes the instance with the object specifiers representing the starting element, *startSpec*, and the ending element, *endSpec*, of a range of elements in the container.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptObjectSpecifiers.h`

setEndSpecifier:

Sets the object specifier representing the last object of the range to a given object.

```
- (void)setEndSpecifier:(NSScriptObjectSpecifier *)endSpec
```

Parameters
endSpec
The object specifier representing the last object of the range.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptObjectSpecifiers.h`

setStartSpecifier:

Sets the object specifier representing the first object of the range to a given object.

```
- (void)setStartSpecifier:(NSScriptObjectSpecifier *)startSpec
```

Parameters
startSpec
The object specifier representing the first object of the range.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptObjectSpecifiers.h`

startSpecifier

Returns the object specifier representing the first object of the range.

```
- (NSScriptObjectSpecifier *)startSpecifier
```

Return Value
The object specifier representing the first object of the range.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Sketch-112

Declared In
`NSScriptObjectSpecifiers.h`

NSRecursiveLock Class Reference

Inherits from	NSObject
Conforms to	NSLocking NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSLock.h
Companion guide	Threading Programming Guide
Related sample code	CIVideoDemoGL LSMSmartCategorizer QTCoreImage101 QTCoreVideo103 QTCoreVideo201

Overview

`NSRecursiveLock` defines a lock that may be acquired multiple times by the same thread without causing a deadlock, a situation where a thread is permanently blocked waiting for itself to relinquish a lock. While the locking thread has one or more locks, all other threads are prevented from accessing the code protected by the lock.

Adopted Protocols

NSLocking
- `lock` (page 2091)
- `unlock` (page 2092)

Tasks

Acquiring a Lock

- `lockBeforeDate:` (page 1322)
  Attempts to acquire a lock before a given date.
- `tryLock` (page 1323)
  Attempts to acquire a lock, and immediately returns a Boolean value that indicates whether the attempt was successful.

Naming the Lock

- `setName:` (page 1323)
  Assigns a name to the receiver
- `name` (page 1322)
  Returns the name associated with the receiver.

Instance Methods

lockBeforeDate:

Attempts to acquire a lock before a given date.

```
- (BOOL)lockBeforeDate:(NSDate *)limit
```

Parameters

limit
  The time before which the lock should be acquired.

Return Value
`YES` if the lock is acquired before *limit*, otherwise `NO`.

Discussion
The thread is blocked until the receiver acquires the lock or *limit* is reached.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSLock.h`

name

Returns the name associated with the receiver.

```
- (NSString *)name
```

Return Value
The name of the receiver.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `setName:` (page 1323)

Declared In
`NSLock.h`

setName:

Assigns a name to the receiver

```
- (void)setName:(NSString *)newName
```

Parameters

newName
The new name for the receiver. This method makes a copy of the specified string.

Discussion
You can use a name string to identify a lock within your code. Cocoa also uses this name as part of any error descriptions involving the receiver.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `name` (page 1322)

Declared In
`NSLock.h`

tryLock

Attempts to acquire a lock, and immediately returns a Boolean value that indicates whether the attempt was successful.

```
- (BOOL)tryLock
```

Return Value
`YES` if successful, otherwise `NO`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSLock.h`

NSRelativeSpecifier Class Reference

Inherits from	NSScriptObjectSpecifier : NSObject
Conforms to	NSCoding (NSScriptObjectSpecifier) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptObjectSpecifiers.h
Companion guide	Cocoa Scripting Guide
Related sample code	Sketch-112

Overview

Specifies an object in a collection by its position relative to another object. You don't normally subclass `NSRelativeSpecifier`.

Tasks

Initializing a Relative Specifier

`- initWithContainerClassDescription:containerSpecifier:key:relativePosition:baseSpecifier:` (page 1326)

Invokes the super class's `initWithContainerClassDescription:containerSpecifier:key:` (page 1418) method and initializes the relative position and base specifier to *relPos* and *baseSpecifier*.

Accessing a Relative Specifier

`- baseSpecifier` (page 1326)

Returns a specifier for the base object.

`- relativePosition` (page 1326)

Returns the relative position encapsulated by the receiver.

- `setBaseSpecifier:` (page 1327)
  Sets the specifier for the base object.
- `setRelativePosition:` (page 1327)
  Sets the relative position encapsulated by the receiver.

Instance Methods

baseSpecifier

Returns a specifier for the base object.

```
- (NSScriptObjectSpecifier *)baseSpecifier
```

Return Value
A specifier for the base object—the object to which the relative specifier is related.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Sketch-112

Declared In
`NSScriptObjectSpecifiers.h`

initWithContainerClassDescription:containerSpecifier:key:relativePosition:baseSpecifier:

Invokes the super class's `initWithContainerClassDescription:containerSpecifier:key:` (page 1418) method and initializes the relative position and base specifier to *relPos* and *baseSpecifier*.

```
- (id)initWithContainerClassDescription:(NSScriptClassDescription *)classDescription
    containerSpecifier:(NSScriptObjectSpecifier *)specifier key:(NSString *)property
    relativePosition:(NSRelativePosition)relPos
   baseSpecifier:(NSScriptObjectSpecifier *)baseSpecifier
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptObjectSpecifiers.h`

relativePosition

Returns the relative position encapsulated by the receiver.

```
- (NSRelativePosition)relativePosition
```

Return Value
The relative position encapsulated by the receiver.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Sketch-112

Declared In
NSScriptObjectSpecifiers.h

setBaseSpecifier:

Sets the specifier for the base object.

```
- (void)setBaseSpecifier:(NSScriptObjectSpecifier *)baseSpecifier
```

Parameters
baseSpecifier
The specifier for the base object—the object to which the relative specifier is related.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptObjectSpecifiers.h

setRelativePosition:

Sets the relative position encapsulated by the receiver.

```
- (void)setRelativePosition:(NSRelativePosition)relPos
```

Parameters
relPos
The relative position encapsulated by the receiver.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptObjectSpecifiers.h

Constants

NSRelativePosition

These constants are used by `relativePosition` (page 1326) and `setRelativePosition:` (page 1327).

```
typedef enum {
    NSRelativeAfter = 0,
    NSRelativeBefore
} NSRelativePosition;
```

Constants

`NSRelativeAfter`

Specifies a position after another object.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptObjectSpecifiers.h`.

`NSRelativeBefore`

Specifies a position before another object.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptObjectSpecifiers.h`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSScriptObjectSpecifiers.h`

NSRunLoop Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSRunLoop.h
Companion guide	Threading Programming Guide
Related sample code	CocoaEcho CocoaSOAP QTAudioExtractionPanel SimpleCocoaJavaMovie WhackedTV

Overview

The `NSRunLoop` class declares the programmatic interface to objects that manage input sources. An `NSRunLoop` object processes input for sources such as mouse and keyboard events from the window system, `NSPort` objects, and `NSConnection` objects. An `NSRunLoop` object also processes `NSTimer` events.

In general, your application does not need to either create or explicitly manage `NSRunLoop` objects. Each `NSThread` object, including the application's main thread, has an `NSRunLoop` object automatically created for it as needed. If you need to access the current thread's run loop, you do so with the class method `currentRunLoop` (page 1331).

Note that from the perspective of `NSRunloop`, `NSTimer` objects are not "input"—they are a special type, and one of the things that means is that they do not cause the run loop to return when they fire.

Warning: The `NSRunLoop` class is generally not considered to be thread-safe and its methods should only be called within the context of the current thread. You should never try to call the methods of an `NSRunLoop` object running in a different thread, as doing so might cause unexpected results.

Tasks

Accessing Run Loops and Modes

+ `currentRunLoop` (page 1331)
  Returns the `NSRunLoop` object for the current thread.

- `currentMode` (page 1335)
  Returns the receiver's current input mode.

- `limitDateForMode:` (page 1336)
  Performs one pass through the run loop in the specified mode and returns the date at which the next timer is scheduled to fire.

+ `mainRunLoop` (page 1332)
  Returns the run loop of the main thread.

- `getCFRunLoop` (page 1336)
  Returns the receiver's underlying *CFRunLoop Reference* object.

Managing Timers

- `addTimer:forMode:` (page 1333)
  Registers a given timer with a given input mode.

Managing Ports

- `addPort:forMode:` (page 1333)
  Adds a port as an input source to the specified mode of the run loop.

- `removePort:forMode:` (page 1337)
  Removes a port from the specified input mode of the run loop.

Configuring as Server Process

- `configureAsServer` (page 1335) Deprecated in Mac OS X v10.5
  Deprecated. Does nothing. (Deprecated. Deprecated since Mac OS X v10.5. There is no alternative method.)

Running a Loop

- `run` (page 1338)
  Puts the receiver into a permanent loop, during which time it processes data from all attached input sources.

- `runMode:beforeDate:` (page 1339)
  Runs the loop once, blocking for input in the specified mode until a given date.

- `runUntilDate:` (page 1339)
  Runs the loop until the specified date, during which time it processes data from all attached input sources.
- `acceptInputForMode:beforeDate:` (page 1332)
  Runs the loop once or until the specified date, accepting input only for the specified mode.

Scheduling and Canceling Messages

- `performSelector:target:argument:order:modes:` (page 1336)
  Schedules the sending of a message on the current run loop.
- `cancelPerformSelector:target:argument:` (page 1334)
  Cancels the sending of a previously scheduled message.
- `cancelPerformSelectorsWithTarget:` (page 1334)
  Cancels all outstanding ordered performs scheduled with a given target.

Class Methods

currentRunLoop

Returns the `NSRunLoop` object for the current thread.

```
+ (NSRunLoop *)currentRunLoop
```

Return Value
The `NSRunLoop` object for the current thread.

Discussion
If a run loop does not yet exist for the thread, one is created and returned.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `currentMode` (page 1335)

Related Sample Code
CocoaEcho
CocoaSOAP
QTAudioExtractionPanel
Quartz Composer Texture
WhackedTV

Declared In
`NSRunLoop.h`

mainRunLoop

Returns the run loop of the main thread.

```
+ (NSRunLoop *)mainRunLoop
```

Return Value
An object representing the main thread's run loop.

Availability
Available in Mac OS X v10.5.

Declared In
NSRunLoop.h

Instance Methods

acceptInputForMode:beforeDate:

Runs the loop once or until the specified date, accepting input only for the specified mode.

```
- (void)acceptInputForMode:(NSString *)mode beforeDate:(NSDate *)limitDate
```

Parameters

mode

The mode in which to run. You may specify custom modes or use one of the modes listed in "Run Loop Modes" (page 1340).

limitDate

The date up until which to run.

Discussion
If no input sources or timers are attached to the run loop, this method exits immediately; otherwise, it runs the run loop once, returning as soon as one input source processes a message or the specifed time elapses.

Note: A timer is not considered an input source and may fire multiple times while waiting for this method to return

Manually removing all known input sources and timers from the run loop is not a guarantee that the run loop will exit. Mac OS X can install and remove additional input sources as needed to process requests targeted at the receiver's thread. Those sources could therefore prevent the run loop from exiting.

Availability
Available in Mac OS X v10.0 and later.

See Also
- runMode:beforeDate: (page 1339)

Declared In
NSRunLoop.h

addPort:forMode:

Adds a port as an input source to the specified mode of the run loop.

```
- (void)addPort:(NSPort *)aPort forMode:(NSString *)mode
```

Parameters

aPort

The port to add to the receiver.

mode

The mode in which to add *aPort*. You may specify a custom mode or use one of the modes listed in "Run Loop Modes" (page 1340).

Discussion

This method schedules the port with the receiver. You can add a port to multiple input modes. When the receiver is running in the specified mode, it dispatches messages destined for that port to the port's designated handler routine.

Availability

Available in Mac OS X v10.0 and later.

See Also

- removePort:forMode: (page 1337)

Declared In

NSRunLoop.h

addTimer:forMode:

Registers a given timer with a given input mode.

```
- (void)addTimer:(NSTimer *)aTimer forMode:(NSString *)mode
```

Parameters

aTimer

The timer to register with the receiver.

mode

The mode in which to add *aTimer*. You may specify a custom mode or use one of the modes listed in "Run Loop Modes" (page 1340).

Discussion

You can add a timer to multiple input modes. While running in the designated mode, the receiver causes the timer to fire on or after its scheduled fire date. Upon firing, the timer invokes its associated handler routine, which is a selector on a designated object.

The receiver retains *aTimer*. To remove a timer from all run loop modes on which it is installed, send an invalidate (page 1660) message to the timer.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

OpenGLCaptureToMovie

OpenGLCompositorLab

Quartz Composer QCTV
Quartz Composer Texture
WhackedTV

Declared In
NSRunLoop.h

cancelPerformSelector:target:argument:

Cancels the sending of a previously scheduled message.

```
- (void)cancelPerformSelector:(SEL)aSelector target:(id)target
    argument:(id)anArgument
```

Parameters

aSelector
The previously-specified selector.

target
The previously-specified target.

anArgument
The previously-specified argument.

Discussion
You can use this method to cancel a message previously scheduled using the `performSelector:target:argument:order:modes:` (page 1336) method. The parameters identify the message you want to cancel and must match those originally specified when the selector was scheduled. This method removes the perform request from all modes of the run loop.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSRunLoop.h

cancelPerformSelectorsWithTarget:

Cancels all outstanding ordered performs scheduled with a given target.

```
- (void)cancelPerformSelectorsWithTarget:(id)target
```

Parameters

target
The previously-specified target.

Discussion
This method cancels the previously scheduled messages associated with the target, ignoring the selector and argument of the scheduled operation. This is in contrast to `cancelPerformSelector:target:argument:` (page 1334), which requires you to match the selector and argument as well as the target. This method removes the perform requests for the object from all modes of the run loop.

Availability
Available in Mac OS X v10.2 and later.

Declared In
NSRunLoop.h

configureAsServer

Deprecated. Does nothing. (Deprecated in Mac OS X v10.5. Deprecated since Mac OS X v10.5. There is no alternative method.)

```
- (void)configureAsServer
```

Discussion
On Mac OS X, this method does nothing.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
NSRunLoop.h

currentMode

Returns the receiver's current input mode.

```
- (NSString *)currentMode
```

Return Value
The receiver's current input mode. This method returns the current input mode *only* while the receiver is running; otherwise, it returns `nil`.

Discussion
The current mode is set by the methods that run the run loop, such as `acceptInputForMode:beforeDate:` (page 1332) and `runMode:beforeDate:` (page 1339).

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ currentRunLoop` (page 1331)
`- limitDateForMode:` (page 1336)
`- run` (page 1338)
`- runUntilDate:` (page 1339)

Declared In
NSRunLoop.h

getCFRunLoop

Returns the receiver's underlying *CFRunLoop Reference* object.

```
- (CFRunLoopRef)getCFRunLoop
```

Return Value
The receiver's underlying *CFRunLoop Reference* object.

Discussion
You can use the returned run loop to configure the current run loop using Core Foundation function calls. For example, you might use this function to set up a run loop observer.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSRunLoop.h`

limitDateForMode:

Performs one pass through the run loop in the specified mode and returns the date at which the next timer is scheduled to fire.

```
- (NSDate *)limitDateForMode:(NSString *)mode
```

Parameters

mode

The run loop mode to search. You may specify custom modes or use one of the modes listed in "Run Loop Modes" (page 1340).

Return Value
The date at which the next timer is scheduled to fire, or `nil` if there are no input sources for this mode.

Discussion
The run loop is entered with an immediate timeout, so the run loop does not block, waiting for input, if no input sources need processing.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSRunLoop.h`

performSelector:target:argument:order:modes:

Schedules the sending of a message on the current run loop.

```
- (void)performSelector:(SEL)aSelector target:(id)target argument:(id)anArgument
    order:(NSUInteger)order modes:(NSArray *)modes
```

Parameters

aSelector

A selector that identifies the method to invoke. This method should not have a significant return value and should take a single argument of type `id`.

target

The object that defines the selector in *aSelector*.

anArgument

The argument to pass to the method when it is invoked. Pass `nil` if the method does not take an argument.

order

The priority for the message. If multiple messages are scheduled, the messages with a lower order value are sent before messages with a higher order value.

modes

An array of input modes for which the message may be sent. You may specify custom modes or use one of the modes listed in "Run Loop Modes" (page 1340).

Discussion

This method sets up a timer to perform the *aSelector* message on the current thread's run loop at the start of the next run loop iteration. 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.

This method returns before the *aSelector* message is sent. The receiver retains the *target* and *anArgument* objects until the timer for the selector fires, and then releases them as part of its cleanup.

Use this method if you want multiple messages to be sent after the current event has been processed and you want to make sure these messages are sent in a certain order.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- cancelPerformSelector:target:argument:` (page 1334)

Related Sample Code

StickiesExample

Declared In

`NSRunLoop.h`

removePort:forMode:

Removes a port from the specified input mode of the run loop.

```
- (void)removePort:(NSPort *)aPort forMode:(NSString *)mode
```

Parameters

aPort

The port to remove from the receiver.

mode

The mode from which to remove *aPort*. You may specify a custom mode or use one of the modes listed in "Run Loop Modes" (page 1340).

Discussion
If you added the port to multiple input modes, you must remove it from each mode separately.

Availability
Available in Mac OS X v10.0 and later.

See Also
- addPort:forMode: (page 1333)

Declared In
NSRunLoop.h

run

Puts the receiver into a permanent loop, during which time it processes data from all attached input sources.

```
- (void)run
```

Discussion
If no input sources or timers are attached to the run loop, this method exits immediately; otherwise, it runs the receiver in the NSDefaultRunLoopMode by repeatedly invoking runMode:beforeDate: (page 1339). In other words, this method effectively begins an infinite loop that processes data from the run loop's input sources and timers.

Manually removing all known input sources and timers from the run loop is not a guarantee that the run loop will exit. Mac OS X can install and remove additional input sources as needed to process requests targeted at the receiver's thread. Those sources could therefore prevent the run loop from exiting.

If you want the run loop to terminate, you shouldn't use this method. Instead, use one of the other run methods and also check other arbitrary conditions of your own, in a loop. A simple example would be:

```
BOOL shouldKeepRunning = YES;        // global
NSRunLoop *theRL = [NSRunLoop currentRunLoop];
while (shouldKeepRunning && [theRL runMode:NSDefaultRunLoopMode beforeDate:[NSDate
 distantFuture]]);
```

where shouldKeepRunning is set to NO somewhere else in the program.

Availability
Available in Mac OS X v10.0 and later.

See Also
- runUntilDate: (page 1339)

Related Sample Code
CocoaEcho
CocoaHTTPServer
CocoaSOAP
SimpleThreads
TrivialThreads

Declared In
NSRunLoop.h

runMode:beforeDate:

Runs the loop once, blocking for input in the specified mode until a given date.

```
- (BOOL)runMode:(NSString *)mode beforeDate:(NSDate *)limitDate
```

Parameters

mode

The mode in which to run. You may specify custom modes or use one of the modes listed in "Run Loop Modes" (page 1340).

limitDate

The date until which to block.

Return Value
NO without starting the run loop if there are no input sources in *mode*; otherwise YES.

Discussion
If no input sources or timers are attached to the run loop, this method exits immediately; otherwise, it returns after either the first input source is processed or *limitDate* is reached. Manually removing all known input sources and timers from the run loop is not a guarantee that the run loop will exit. Mac OS X may install and remove additional input sources as needed to process requests targeted at the receiver's thread. Those sources could therefore prevent the run loop from exiting.

Note: A timer is not considered an input source and may fire multiple times while waiting for this method to return

Availability
Available in Mac OS X v10.0 and later.

See Also
- run (page 1338)
- runUntilDate: (page 1339)

Related Sample Code
CocoaSOAP

Declared In
NSRunLoop.h

runUntilDate:

Runs the loop until the specified date, during which time it processes data from all attached input sources.

```
- (void)runUntilDate:(NSDate *)limitDate
```

Parameters

limitDate

The date up until which to run.

Discussion
If no input sources or timers are attached to the run loop, this method exits immediately; otherwise, it runs the receiver in the `NSDefaultRunLoopMode` by repeatedly invoking `runMode:beforeDate:` (page 1339) until the specified expiration date.

Manually removing all known input sources and timers from the run loop is not a guarantee that the run loop will exit. Mac OS X can install and remove additional input sources as needed to process requests targeted at the receiver's thread. Those sources could therefore prevent the run loop from exiting.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `run` (page 1338)

Related Sample Code
EnhancedAudioBurn
QTAudioExtractionPanel

Declared In
`NSRunLoop.h`

Constants

Run Loop Modes

`NSRunLoop` defines the following run loop mode.

```
extern NSString *NSDefaultRunLoopMode;
```

Constants

`NSDefaultRunLoopMode`
The mode to deal with input sources other than `NSConnection` objects.

This is the most commonly used run-loop mode.

Available in Mac OS X v10.0 and later.

Declared in `NSRunLoop.h`.

`NSRunLoopCommonModes`
Objects added to a run loop using this value as the mode are monitored by all run loop modes that have been declared as a member of the set of "common" modes; see the description of `CFRunLoopAddCommonMode` for details.

Available in Mac OS X v10.5 and later.

Declared in `NSRunLoop.h`.

Declared In
`Foundation/NSRunLoop.h`

Additional run loop modes are defined by `NSConnection` and `NSApplication`.

`NSConnectionReplyMode`	Use this mode to indicate NSConnection objects waiting for replies. Defined in the Foundation/NSConnection.h header file. You rarely need to use this mode.
`NSModalPanelRunLoopMode`	A run loop should be set to this mode when waiting for input from a modal panel, such as NSSavePanel or NSOpenPanel.
`NSEventTrackingRunLoopMode`	A run loop should be set to this mode when tracking events modally, such as a mouse-dragging loop.

NSScanner Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScanner.h Foundation/NSDecimalNumber.h
Companion guide	String Programming Guide for Cocoa
Related sample code	ImageMapExample iSpend NumberInput_IMKit_Sample QTAudioExtractionPanel Quartz Composer QCTV

Overview

The `NSScanner` class is an abstract superclass of a class cluster that declares the programmatic interface for an object that scans values from an `NSString` object.

An `NSScanner` object interprets and converts the characters of an `NSString` object into number and string values. You assign the scanner's string on creating it, and the scanner progresses through the characters of that string from beginning to end as you request items.

Because of the nature of class clusters, scanner objects aren't actual instances of the `NSScanner` class but one of its private subclasses. Although a scanner object's class is private, its interface is public, as declared by this abstract superclass, `NSScanner`. The primitive methods of `NSScanner` are `string` (page 1359) and all of the methods listed under "Configuring a Scanner" (page 1344) in the "Methods by Task" section. The objects you create using this class are referred to as scanner objects (and when no confusion will result, merely as scanners).

You can set an `NSScanner` object to ignore a set of characters as it scans the string using the `setCharactersToBeSkipped:` (page 1357) method. The default set of characters to skip is the whitespace and newline character set.

To retrieve the unscanned remainder of the string, use `[[scanner string]substringFromIndex:` (page 1609)`[scanner scanLocation]]`.

Adopted Protocols

NSCopying
- `copyWithZone:` (page 2042)

Tasks

Creating an Scanner

`+ scannerWithString:` (page 1346)
Returns an `NSScanner` object that scans a given string.

`+ localizedScannerWithString:` (page 1345)
Returns an `NSScanner` object that scans a given string according to the user's default locale.

`- initWithString:` (page 1347)
Returns an `NSScanner` object initialized to scan a given string.

Getting a Scanner's String

`- string` (page 1359)
Returns the string with which the receiver was created or initialized.

Configuring a Scanner

`- setScanLocation:` (page 1358)
Sets the location at which the next scan operation will begin to a given index.

`- scanLocation` (page 1354)
Returns the character position at which the receiver will begin its next scanning operation.

`- setCaseSensitive:` (page 1357)
Sets whether the receiver is case sensitive when scanning characters.

`- caseSensitive` (page 1346)
Returns a Boolean value that indicates whether the receiver distinguishes case in the characters it scans.

`- setCharactersToBeSkipped:` (page 1357)
Sets the set of characters to ignore when scanning for a value representation.

`- charactersToBeSkipped` (page 1347)
Returns a character set containing the characters the receiver ignores when looking for a scannable element.

`- setLocale:` (page 1358)
Sets the receiver's locale to a given locale.

`- locale` (page 1348)
Returns the receiver's locale.

Scanning a String

- `scanCharactersFromSet:intoString:` (page 1349)
  Scans the string as long as characters from a given character set are encountered, accumulating characters into a string that's returned by reference.
- `scanUpToCharactersFromSet:intoString:` (page 1355)
  Scans the string until a character from a given character set is encountered, accumulating characters into a string that's returned by reference.
- `scanDecimal:` (page 1349)
  Scans for an `NSDecimal` value, returning a found value by reference.
- `scanDouble:` (page 1350)
  Scans for a `double` value, returning a found value by reference.
- `scanFloat:` (page 1350)
  Scans for a `float` value, returning a found value by reference.
- `scanHexDouble:` (page 1351)
  Scans for a `double` value from a hexadecimal representation, returning a found value by reference.
- `scanHexFloat:` (page 1352)
  Scans for a `double` value from a hexadecimal representation, returning a found value by reference.
- `scanHexInt:` (page 1352)
  Scans for an `unsigned` value from a hexadecimal representation, returning a found value by reference.
- `scanHexLongLong:` (page 1352)
  Scans for a `double` value from a hexadecimal representation, returning a found value by reference.
- `scanInteger:` (page 1353)
  Scans for an `NSInteger` value from a decimal representation, returning a found value by reference
- `scanInt:` (page 1353)
  Scans for an `int` value from a decimal representation, returning a found value by reference.
- `scanLongLong:` (page 1354)
  Scans for a `long long` value from a decimal representation, returning a found value by reference.
- `scanString:intoString:` (page 1355)
  Scans a given string, returning an equivalent string object by reference if a match is found.
- `scanUpToString:intoString:` (page 1356)
  Scans the string until a given string is encountered, accumulating characters into a string that's returned by reference.
- `isAtEnd` (page 1348)
  Returns a Boolean value that indicates whether the receiver has exhausted all significant characters

Class Methods

localizedScannerWithString:

Returns an `NSScanner` object that scans a given string according to the user's default locale.

```
+ (id)localizedScannerWithString:(NSString *)aString
```

Parameters

aString

The string to scan.

Return Value

An `NSScanner` object that scans *aString* according to the user's default locale.

Discussion

Sets the string to scan by invoking `initWithString:` (page 1347) with *aString*. The locale is set with `setLocale:` (page 1358).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSScanner.h`

scannerWithString:

Returns an `NSScanner` object that scans a given string.

```
+ (id)scannerWithString:(NSString *)aString
```

Parameters

aString

The string to scan.

Return Value

An `NSScanner` object that scans *aString*.

Discussion

Sets the string to scan by invoking `initWithString:` (page 1347) with *aString*.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

iSpend

NumberInput_IMKit_Sample

QTAudioExtractionPanel

Quartz Composer QCTV

Sproing

Declared In

`NSScanner.h`

Instance Methods

caseSensitive

Returns a Boolean value that indicates whether the receiver distinguishes case in the characters it scans.

```
- (BOOL)caseSensitive
```

Return Value
`YES` if the receiver distinguishes case in the characters it scans, otherwise `NO`.

Discussion
Scanners are not case sensitive by default. Note that case sensitivity doesn't apply to the characters to be skipped.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setCaseSensitive:` (page 1357)
`- setCharactersToBeSkipped:` (page 1357)

Declared In
`NSScanner.h`

charactersToBeSkipped

Returns a character set containing the characters the receiver ignores when looking for a scannable element.

```
- (NSCharacterSet *)charactersToBeSkipped
```

Return Value
A character set containing the characters the receiver ignores when looking for a scannable element.

Discussion
For example, if a scanner ignores spaces and you send it a `scanInt:` (page 1353) message, it skips spaces until it finds a decimal digit or other character. While an element is being scanned, however, no characters are skipped. If you scan for something made of characters in the set to be skipped (for example, using `scanInt:` (page 1353) when the set of characters to be skipped is the decimal digits), the result is undefined.

The default set to skip is the whitespace and newline character set.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setCharactersToBeSkipped:` (page 1357)
`whitespaceAndNewlineCharacterSet` (page 252) (`NSCharacterSet`)

Declared In
`NSScanner.h`

initWithString:

Returns an `NSScanner` object initialized to scan a given string.

```
- (id)initWithString:(NSString *)aString
```

Parameters

aString

The string to scan.

Return Value

An `NSScanner` object initialized to scan *aString* from the beginning. The returned object might be different than the original receiver.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ localizedScannerWithString:` (page 1345)

`+ scannerWithString:` (page 1346)

Declared In

`NSScanner.h`

isAtEnd

Returns a Boolean value that indicates whether the receiver has exhausted all significant characters

```
- (BOOL)isAtEnd
```

Return Value

`YES` if the receiver has exhausted all significant characters in its string, otherwise `NO`.

If only characters from the set to be skipped remain, returns `YES`.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- charactersToBeSkipped` (page 1347)

Related Sample Code

QTAudioExtractionPanel

Declared In

`NSScanner.h`

locale

Returns the receiver's locale.

```
- (id)locale
```

Return Value

The receiver's locale, or `nil` if it has none.

Discussion
A scanner's locale affects the way it interprets numeric values from the string. In particular, a scanner uses the locale's decimal separator to distinguish the integer and fractional parts of floating-point representations. A scanner with no locale set uses non-localized values.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setLocale:` (page 1358)

Declared In
`NSScanner.h`

scanCharactersFromSet:intoString:

Scans the string as long as characters from a given character set are encountered, accumulating characters into a string that's returned by reference.

```
- (BOOL)scanCharactersFromSet:(NSCharacterSet *)scanSet intoString:(NSString
    **)stringValue
```

Parameters

scanSet
The set of characters to scan.

stringValue
Upon return, contains the characters scanned.

Return Value
`YES` if the receiver scanned any characters, otherwise `NO`.

Discussion
Invoke this method with `NULL` as *stringValue* to simply scan past a given set of characters.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- scanUpToCharactersFromSet:intoString:` (page 1355)

Declared In
`NSScanner.h`

scanDecimal:

Scans for an `NSDecimal` value, returning a found value by reference.

```
- (BOOL)scanDecimal:(NSDecimal *)decimalValue
```

Parameters

decimalValue
Upon return, contains the scanned value. See the `NSDecimalNumber` class specification for more information about `NSDecimal` values.

Return Value
`YES` if the receiver finds a valid `NSDecimal` representation, otherwise `NO`.

Discussion
Invoke this method with `NULL` as *`decimalValue`* to simply scan past an `NSDecimal` representation.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
NumberInput_IMKit_Sample

Declared In
`NSDecimalNumber.h`

scanDouble:

Scans for a `double` value, returning a found value by reference.

```
- (BOOL)scanDouble:(double *)doubleValue
```

Parameters
`doubleValue`
Upon return, contains the scanned value. Contains `HUGE_VAL` or `-HUGE_VAL` on overflow, or `0.0` on underflow.

Return Value
`YES` if the receiver finds a valid floating-point representation, otherwise `NO`.

Discussion
Skips past excess digits in the case of overflow, so the scanner's position is past the entire floating-point representation.

Invoke this method with `NULL` as *`doubleValue`* to simply scan past a `double` value representation. Floating-point representations are assumed to be IEEE compliant.

Availability
Available in Mac OS X v10.0 and later.

See Also
`doubleValue` (page 1552) (`NSString`)

Declared In
`NSScanner.h`

scanFloat:

Scans for a `float` value, returning a found value by reference.

```
- (BOOL)scanFloat:(float *)floatValue
```

Parameters

floatValue

Upon return, contains the scanned value. Contains HUGE_VAL or -HUGE_VAL on overflow, or 0.0 on underflow.

Return Value

YES if the receiver finds a valid floating-point representation, otherwise NO.

Discussion

Skips past excess digits in the case of overflow, so the scanner's position is past the entire floating-point representation.

Invoke this method with NULL as *floatValue* to simply scan past a float value representation. Floating-point representations are assumed to be IEEE compliant.

Availability

Available in Mac OS X v10.0 and later.

See Also

floatValue (page 1553) (NSString)

Related Sample Code

iSpend

Quartz Composer QCTV

Declared In

NSScanner.h

scanHexDouble:

Scans for a double value from a hexadecimal representation, returning a found value by reference.

```
- (BOOL)scanHexDouble:(double *)result
```

Parameters

result

Upon return, contains the scanned value.

Return Value

YES if the receiver finds a valid double-point representation, otherwise NO.

Discussion

This corresponds to %a or %A formatting. The hexadecimal double representation must be preceded by 0x or 0X.

Invoke this method with NULL as *result* to simply scan past a hexadecimal double representation.

Availability

Available in Mac OS X v10.5 and later.

Declared In

NSScanner.h

scanHexFloat:

Scans for a `double` value from a hexadecimal representation, returning a found value by reference.

```
- (BOOL)scanHexFloat:(float *)result
```

Parameters

result

Upon return, contains the scanned value.

Return Value

`YES` if the receiver finds a valid float-point representation, otherwise `NO`.

Discussion

This corresponds to `%a` or `%A` formatting. The hexadecimal float representation must be preceded by `0x` or `0X`.

Invoke this method with `NULL` as *result* to simply scan past a hexadecimal float representation.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSScanner.h`

scanHexInt:

Scans for an `unsigned` value from a hexadecimal representation, returning a found value by reference.

```
- (BOOL)scanHexInt:(unsigned *)intValue
```

Parameters

intValue

Upon return, contains the scanned value. Contains `INT_MAX` or `INT_MIN` on overflow.

Return Value

Returns `YES` if the receiver finds a valid hexadecimal integer representation, otherwise `NO`.

Discussion

The hexadecimal integer representation may optionally be preceded by `0x` or `0X`. Skips past excess digits in the case of overflow, so the receiver's position is past the entire hexadecimal representation.

Invoke this method with `NULL` as *intValue* to simply scan past a hexadecimal integer representation.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSScanner.h`

scanHexLongLong:

Scans for a `double` value from a hexadecimal representation, returning a found value by reference.

```
- (BOOL)scanHexLongLong:(unsigned long long *)result
```

Parameters

result

Upon return, contains the scanned value.

Return Value

YES if the receiver finds a valid double-point representation, otherwise NO.

Discussion

Invoke this method with NULL as *result* to simply scan past a hexadecimal long long representation.

Availability

Available in Mac OS X v10.5 and later.

Declared In

NSScanner.h

scanInt:

Scans for an int value from a decimal representation, returning a found value by reference.

```
- (BOOL)scanInt:(int *)intValue
```

Parameters

intValue

Upon return, contains the scanned value. Contains INT_MAX or INT_MIN on overflow.

Return Value

YES if the receiver finds a valid decimal integer representation, otherwise NO.

Discussion

Skips past excess digits in the case of overflow, so the receiver's position is past the entire decimal representation.

Invoke this method with NULL as *intValue* to simply scan past a decimal integer representation.

Availability

Available in Mac OS X v10.0 and later.

See Also

intValue (page 1577) (NSString)

- scanInteger: (page 1353)

Declared In

NSScanner.h

scanInteger:

Scans for an NSInteger value from a decimal representation, returning a found value by reference

```
- (BOOL)scanInteger:(NSInteger *)value
```

Parameters

value

Upon return, contains the scanned value.

Return Value
`YES` if the receiver finds a valid integer representation, otherwise `NO`.

Discussion
Skips past excess digits in the case of overflow, so the receiver's position is past the entire integer representation.

Invoke this method with `NULL` as *`value`* to simply scan past a decimal integer representation.

Availability
Available in Mac OS X v10.5 and later.

See Also
`integerValue` (page 1576) (`NSString`)
`- scanInt:` (page 1353)

Declared In
`NSScanner.h`

scanLocation

Returns the character position at which the receiver will begin its next scanning operation.

```
- (NSUInteger)scanLocation
```

Return Value
The character position at which the receiver will begin its next scanning operation.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setScanLocation:` (page 1358)

Declared In
`NSScanner.h`

scanLongLong:

Scans for a `long long` value from a decimal representation, returning a found value by reference.

```
- (BOOL)scanLongLong:(long long *)longLongValue
```

Parameters
`longLongValue`
Upon return, contains the scanned value. Contains `LLONG_MAX` or `LLONG_MIN` on overflow.

Return Value
`YES` if the receiver finds a valid decimal integer representation, otherwise `NO`.

Discussion
All overflow digits are skipped. Skips past excess digits in the case of overflow, so the receiver's position is past the entire decimal representation.

Invoke this method with NULL as *longLongValue* to simply scan past a long decimal integer representation.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScanner.h

scanString:intoString:

Scans a given string, returning an equivalent string object by reference if a match is found.

```
- (BOOL)scanString:(NSString *)string intoString:(NSString **)stringValue
```

Parameters

string
The string for which to scan at the current scan location.

stringValue
Upon return, if the receiver contains a string equivalent to *string* at the current scan location, contains a string equivalent to *string*.

Return Value
YES if *stringValue* matches the characters at the scan location, otherwise NO.

Discussion
If *string* is present at the current scan location, then the current scan location is advanced to after the string; otherwise the scan location does not change.

Invoke this method with NULL as *stringValue* to simply scan past a given string.

Availability
Available in Mac OS X v10.0 and later.

See Also
- scanUpToString:intoString: (page 1356)

Declared In
NSScanner.h

scanUpToCharactersFromSet:intoString:

Scans the string until a character from a given character set is encountered, accumulating characters into a string that's returned by reference.

```
- (BOOL)scanUpToCharactersFromSet:(NSCharacterSet *)stopSet intoString:(NSString
    **)stringValue
```

Parameters

stopSet
The set of characters up to which to scan.

stringValue
Upon return, contains the characters scanned.

Return Value
YES if the receiver scanned any characters, otherwise NO.

If the only scanned characters are in the charactersToBeSkipped (page 1347) character set (which is the whitespace and newline character set by default), then returns NO.

Discussion
Invoke this method with NULL as *stringValue* to simply scan up to a given set of characters.

If no characters in *stopSet* are present in the scanner's source string, the remainder of the source string is put into *stringValue*, the receiver's scanLocation is advanced to the end of the source string, and the method returns YES.

Availability
Available in Mac OS X v10.0 and later.

See Also
- scanCharactersFromSet:intoString: (page 1349)

Declared In
NSScanner.h

scanUpToString:intoString:

Scans the string until a given string is encountered, accumulating characters into a string that's returned by reference.

```
- (BOOL)scanUpToString:(NSString *)stopString intoString:(NSString **)stringValue
```

Parameters
stopString
The string to scan up to.
stringValue
Upon return, contains any characters that were scanned.

Return Value
YES if the receiver scans any characters, otherwise NO.

If the only scanned characters are in the charactersToBeSkipped (page 1347) character set (which by default is the whitespace and newline character set), then this method returns NO.

Discussion
If stopString is present in the receiver, then on return the scan location is set to the beginning of that string.

If stopString is the first string in the receiver, then the method returns NO and *stringValue* is not changed.

If the search string (stopString) isn't present in the scanner's source string, the remainder of the source string is put into *stringValue*, the receiver's scanLocation is advanced to the end of the source string, and the method returns YES.

Invoke this method with NULL as *stringValue* to simply scan up to a given string.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `scanString:intoString:` (page 1355)

Declared In
`NSScanner.h`

setCaseSensitive:

Sets whether the receiver is case sensitive when scanning characters.

```
- (void)setCaseSensitive:(BOOL)flag
```

Parameters

flag

If `YES`, the receiver will distinguish case when scanning characters, otherwise it will ignore case distinctions.

Discussion
Scanners are not case sensitive by default. Note that case sensitivity doesn't apply to the characters to be skipped.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `caseSensitive` (page 1346)
- `setCharactersToBeSkipped:` (page 1357)

Declared In
`NSScanner.h`

setCharactersToBeSkipped:

Sets the set of characters to ignore when scanning for a value representation.

```
- (void)setCharactersToBeSkipped:(NSCharacterSet *)skipSet
```

Parameters

skipSet

The characters to ignore when scanning for a value representation.

Discussion
For example, if a scanner ignores spaces and you send it a `scanInt:` (page 1353) message, it skips spaces until it finds a decimal digit or other character. While an element is being scanned, however, no characters are skipped. If you scan for something made of characters in the set to be skipped (for example, using `scanInt:` (page 1353) when the set of characters to be skipped is the decimal digits), the result is undefined.

The characters to be skipped are treated literally as single values. A scanner doesn't apply its case sensitivity setting to these characters and doesn't attempt to match composed character sequences with anything in the set of characters to be skipped (though it does match pre-composed characters individually). If you want to skip all vowels while scanning a string, for example, you can set the characters to be skipped to those in the string "AEIOUaeiou" (plus any accented variants with pre-composed characters).

The default set of characters to skip is the whitespace and newline character set.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `charactersToBeSkipped` (page 1347)
`whitespaceAndNewlineCharacterSet` (page 252) (`NSCharacterSet`)

Related Sample Code
ImageMapExample
QTAudioExtractionPanel
Quartz Composer QCTV

Declared In
`NSScanner.h`

setLocale:

Sets the receiver's locale to a given locale.

```
- (void)setLocale:(id)aLocale
```

Parameters
aLocale
The locale for the receiver.

Discussion
A scanner's locale affects the way it interprets values from the string. In particular, a scanner uses the locale's decimal separator to distinguish the integer and fractional parts of floating-point representations. A new scanner's locale is by default `nil`, which causes it to use non-localized values.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `locale` (page 1348)

Declared In
`NSScanner.h`

setScanLocation:

Sets the location at which the next scan operation will begin to a given index.

```
- (void)setScanLocation:(NSUInteger)index
```

Parameters
index
The location at which the next scan operation will begin. Raises an `NSRangeException` if *index* is beyond the end of the string being scanned.

Discussion
This method is useful for backing up to rescan after an error.

Rather than setting the scan location directly to skip known sequences of characters, use `scanString:intoString:` (page 1355) or `scanCharactersFromSet:intoString:` (page 1349), which allow you to verify that the expected substring (or set of characters) is in fact present.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- scanLocation` (page 1354)

Declared In
`NSScanner.h`

string

Returns the string with which the receiver was created or initialized.

```
- (NSString *)string
```

Return Value
The string with which the receiver was created or initialized.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- locale` (page 1348)

Declared In
`NSScanner.h`

NSScriptClassDescription Class Reference

Inherits from	NSClassDescription : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptClassDescription.h
Companion guides	Cocoa Scripting Guide Key-Value Coding Programming Guide
Related sample code	Quartz Composer WWDC 2005 TextEdit TextEditPlus

Overview

An instance of `NSScriptClassDescription` describes a script class that a Cocoa application supports.

A scriptable application provides scriptability information that describes the commands and objects scripters can use in scripts that target the application. That includes information about the classes those scriptable objects are created from.

An application's scriptability information is collected automatically by an instance of `NSScriptSuiteRegistry`. The registry object creates an `NSScriptClassDescription` for each class it finds and caches these objects in memory. Cocoa scripting uses registry information in handling scripting requests that target the application.

A class description instance stores the name, attributes, relationships, and supported commands for a class. For example, a scriptable `document` class for a drawing application might support attributes such as `file` and `file type`, relationships such as collections of `circles`, `rectangles`, and `lines`, and commands such as `align` and `rotate`.

As with many of the classes in Cocoa's built-in scripting support, your application may never need to directly work with instances of `NSScriptClassDescription`. However, one case where you might need access to a class description is if you override `objectSpecifier` in a scriptable class. For information on how to do this, see Object Specifiers in *Cocoa Scripting Guide*.

Another case where your application may need access to class description information is if you override `indicesOfObjectsByEvaluatingWithContainer:count:` in a specifier class.

Although you can subclass `NSScriptClassDescription`, it is unlikely that you would need to do so, or even to create instances of it.

Tasks

Initializing a Script Class Description

- `initWithSuiteName:className:dictionary:` (page 1368)
  Initializes and returns a newly allocated instance of `NSScriptClassDescription`.

Getting a Script Class Description

+ `classDescriptionForClass:` (page 1363)
  Returns the class description for the specified class or, if it is not scriptable, for the first superclass that is.
- `classDescriptionForKey:` (page 1365)
  Returns the class description instance for the class type of the specified attribute or relationship.
- `superclassDescription` (page 1371)
  Returns the class description instance for the superclass of the receiver's class.

Getting Basic Information About the Script Class

- `className` (page 1365)
  Returns the name of the class the receiver describes, as provided at initialization time.
- `defaultSubcontainerAttributeKey` (page 1366)
  Returns the value of the `DefaultSubcontainerAttribute` entry of the class dictionary from which the receiver was instantiated.
- `implementationClassName` (page 1367)
  Returns the name of the Objective-C class instantiated to implement the scripting class.
- `isLocationRequiredToCreateForKey:` (page 1368)
  Returns a Boolean value indicating whether an insertion location must be specified when creating a new object in the specified to-many relationship of the receiver.
- `suiteName` (page 1371)
  Returns the name of the receiver's suite.

Getting and Comparing Apple Event Codes

- `appleEventCode` (page 1364)
  Returns the Apple event code associated with the receiver's class.
- `appleEventCodeForKey:` (page 1364)
  Returns the Apple event code for the specified attribute or relationship in the receiver.

- `matchesAppleEventCode:` (page 1370)
  Returns a Boolean value indicating whether a primary or secondary Apple event code in the receiver matches the passed code.

Getting Attribute and Relationship Information

- `hasOrderedToManyRelationshipForKey:` (page 1366)
  Returns a Boolean value indicating whether the described class has an ordered to-many relationship identified by the specified key.
- `hasPropertyForKey:` (page 1366)
  Returns a Boolean value indicating whether the described class has a property identified by the specified key.
- `hasReadablePropertyForKey:` (page 1367)
  Returns a Boolean value indicating whether the described class has a readable property identified by the specified key.
- `hasWritablePropertyForKey:` (page 1367)
  Returns a Boolean value indicating whether the described class has a writable property identified by the specified key.
- `keyWithAppleEventCode:` (page 1369)
  Given an Apple event code that identifies a property or element class, returns the key for the corresponding attribute, one-to-one relationship, or one-to-many relationship.
- `typeForKey:` (page 1372)
  Returns the name of the declared type of the attribute or relationship identified by the passed key.
- `isReadOnlyKey:` (page 1369) Deprecated in Mac OS X v10.5
  Returns a Boolean value indicating whether a specified property in the receiver is read-only. (Deprecated. Use `hasWritablePropertyForKey:` (page 1367) instead.)

Getting Command Information

- `selectorForCommand:` (page 1370)
  Returns the selector associated with the receiver for the specified command description.
- `supportsCommand:` (page 1372)
  Returns a Boolean value indicating whether the receiver or any superclass supports the specified command.

Class Methods

classDescriptionForClass:

Returns the class description for the specified class or, if it is not scriptable, for the first superclass that is.

```
+ (NSScriptClassDescription *)classDescriptionForClass:(Class)aClass
```

Parameters

aClass

The class whose description is needed.

Return Value

The class description for the class specified by *aClass* or, if that class isn't scriptable, for the class description for the first superclass that is. Returns `nil` if it doesn't find a scriptable class.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSScriptClassDescription.h`

Instance Methods

appleEventCode

Returns the Apple event code associated with the receiver's class.

```
- (FourCharCode)appleEventCode
```

Return Value

The Apple event code associated with the receiver's class. This is the primary code used to identify the class in Apple events.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `appleEventCodeForKey:` (page 1364)
- `matchesAppleEventCode:` (page 1370)

Declared In

`NSScriptClassDescription.h`

appleEventCodeForKey:

Returns the Apple event code for the specified attribute or relationship in the receiver.

```
- (FourCharCode)appleEventCodeForKey:(NSString *)key
```

Parameters

key

The identifying key for an attribute or relationship of the receiver.

Return Value

The four-character Apple event code associated with the attribute or relationship identified by *key* in the receiver or, if none exists, in the class description for the receiver's superclass. Returns `0` if no such attribute or relationship is found.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `appleEventCode` (page 1364)
- `matchesAppleEventCode:` (page 1370)

Declared In
`NSScriptClassDescription.h`

classDescriptionForKey:

Returns the class description instance for the class type of the specified attribute or relationship.

```
- (NSScriptClassDescription *)classDescriptionForKey:(NSString *)key
```

Parameters

key
The identifying key for an attribute or relationship of the receiver.

Return Value
The instance of `NSScriptClassDescription` for the type of the attribute or relationship specified by *key*. Returns `nil` if no scriptable property corresponds to *key*.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `superclassDescription` (page 1371)

Declared In
`NSScriptClassDescription.h`

className

Returns the name of the class the receiver describes, as provided at initialization time.

```
- (NSString *)className
```

Return Value
A class name. This may be either the human-readable name for the class—that is, the name that is used in a script—or the name of the Objective-C class that is instantiated to implement the class. To reliably obtain the implementation name, use `implementationClassName` (page 1367).

Availability
Available in Mac OS X v10.0 and later.

See Also
- `suiteName` (page 1371)

Declared In
`NSScriptClassDescription.h`

defaultSubcontainerAttributeKey

Returns the value of the `DefaultSubcontainerAttribute` entry of the class dictionary from which the receiver was instantiated.

```
- (NSString *)defaultSubcontainerAttributeKey
```

Return Value
The value of the default subcontainer attribute entry. Returns `nil` if the there was no such entry.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`NSScriptClassDescription.h`

hasOrderedToManyRelationshipForKey:

Returns a Boolean value indicating whether the described class has an ordered to-many relationship identified by the specified key.

```
- (BOOL)hasOrderedToManyRelationshipForKey:(NSString *)key
```

Parameters

key
The identifying key for a property of the receiver.

Return Value
`YES` if the described class has an ordered to-many relationship identified by the specified key; otherwise, `NO`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSScriptClassDescription.h`

hasPropertyForKey:

Returns a Boolean value indicating whether the described class has a property identified by the specified key.

```
- (BOOL)hasPropertyForKey:(NSString *)key
```

Parameters

key
The identifying key for a property of the receiver.

Return Value
`YES` if the described class has a property identified by the specified key; otherwise, `NO`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSScriptClassDescription.h`

hasReadablePropertyForKey:

Returns a Boolean value indicating whether the described class has a readable property identified by the specified key.

```
- (BOOL)hasReadablePropertyForKey:(NSString *)key
```

Parameters

key

The identifying key for a property of the receiver.

Return Value

`YES` if the described class has a readable property identified by the specified key; otherwise, `NO`.

Discussion

To determine if a property is read-only, invoke `hasWritablePropertyForKey:` (page 1367)/

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSScriptClassDescription.h`

hasWritablePropertyForKey:

Returns a Boolean value indicating whether the described class has a writable property identified by the specified key.

```
- (BOOL)hasWritablePropertyForKey:(NSString *)key
```

Parameters

key

The identifying key for a property of the receiver.

Return Value

`YES` if the described class has a writable property identified by the specified key; otherwise, `NO`.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSScriptClassDescription.h`

implementationClassName

Returns the name of the Objective-C class instantiated to implement the scripting class.

```
- (NSString *)implementationClassName
```

Return Value

An Objective-C class name.

Discussion
The name returned by the `className` (page 1365) method for an instance of `NSScriptClassDescription` resulting from an sdef class declaration is the human-readable name for the class—that is, the name that is used in a script. To obtain the name of the Objective-C class instantiated to implement the class, use `implementationClassName`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSScriptClassDescription.h`

initWithSuiteName:className:dictionary:

Initializes and returns a newly allocated instance of `NSScriptClassDescription`.

```
- (id)initWithSuiteName:(NSString *)suiteName className:(NSString *)className
    dictionary:(NSDictionary *)classDeclaration
```

Parameters

suiteName
The name of the suite (in the application's scriptability information) that the class belongs to. For example, `"AppName Suite"`.

className
The name of the class that this instance describes.

classDeclaration
A class declaration dictionary of the sort that is valid in script suite property list files. This dictionary provides information about the class such as its attributes and relationships.

Return Value
The initialized instance. Returns `nil` if the event code value for the class description itself is missing or is not an `NSString`. Also returns `nil` if the superclass name or any of the subdictionaries of descriptions are not of the right type.

Discussion
This method registers `self` with the application's global instance of `NSScriptSuiteRegistry`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptClassDescription.h`

isLocationRequiredToCreateForKey:

Returns a Boolean value indicating whether an insertion location must be specified when creating a new object in the specified to-many relationship of the receiver.

```
- (BOOL)isLocationRequiredToCreateForKey:(NSString *)toManyRelationshipKey
```

Parameters

toManyRelationshipKey

The key for the to-many relationship that may require an insertion location.

Return Value

YES if an insertion location must be specified; otherwise, NO.

Discussion

A script command object that creates a new object in a to-many relationship needs to know whether an explicitly specified insertion location is required. It can get this information from an instance of NSScriptClassDescription. For example, NSMakeCommand uses this method to determine whether or not a specific make AppleScript command must have an at parameter.

Availability

Available in Mac OS X v10.2 and later.

Declared In

NSScriptClassDescription.h

isReadOnlyKey:

Returns a Boolean value indicating whether a specified property in the receiver is read-only. (Deprecated in Mac OS X v10.5. Use hasWritablePropertyForKey: (page 1367) instead.)

```
- (BOOL)isReadOnlyKey:(NSString *)key
```

Parameters

key

The identifying key for a property of the receiver.

Return Value

YES if the property specified by *key* exists in the receiver or in the NSScriptClassDescription for any superclass, and is read only; otherwise, NO.

Special Considerations

This method could return NO either because key is unrecognized or because writing to the property is not supported. Use hasWritablePropertyForKey: (page 1367) instead.

Availability

Available in in Mac OS X v10.0.

Deprecated in Mac OS X v10.5.

See Also

- keyWithAppleEventCode: (page 1369)
- typeForKey: (page 1372)

Declared In

NSScriptClassDescription.h

keyWithAppleEventCode:

Given an Apple event code that identifies a property or element class, returns the key for the corresponding attribute, one-to-one relationship, or one-to-many relationship.

```
- (NSString *)keyWithAppleEventCode:(FourCharCode)appleEventCode
```

Parameters

appleEventCode

An Apple event code that identifies a property or element class.

Return Value

The key that corresponds to the property or element class identified by *appleEventCode* in the receiver or, if none exists, in a class description in the receiver's superclasses.

The four-character Apple event code associated with the attribute or relationship identified by *key* Returns `0` if no such attribute or relationship is found. Returns `nil` if it cannot find any such attribute or relationship.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `- isReadOnlyKey:` (page 1369)
- `- typeForKey:` (page 1372)

Declared In

`NSScriptClassDescription.h`

matchesAppleEventCode:

Returns a Boolean value indicating whether a primary or secondary Apple event code in the receiver matches the passed code.

```
- (BOOL)matchesAppleEventCode:(FourCharCode)appleEventCode
```

Parameters

appleEventCode

An Apple event code to compare against the receiver's primary or secondary codes.

Return Value

`YES` if the receiver's primary four-character Apple event code or any of its secondary codes (its synonyms) matches *code*; otherwise, `NO`.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `- appleEventCode` (page 1364)
- `- appleEventCodeForKey:` (page 1364)

Declared In

`NSScriptClassDescription.h`

selectorForCommand:

Returns the selector associated with the receiver for the specified command description.

```
- (SEL)selectorForCommand:(NSScriptCommandDescription *)commandDescription
```

Parameters

commandDescription

A description for a script command, such as `duplicate`, `make`, or `move`. Encapsulates the scriptability information for that command, such as its Objective-C selector, its argument names and types, and its return type (if any).

Return Value

The selector from the receiver for the command specified by *commandDescription*. Searches in the receiver first, then in any superclass. Returns `NULL` if no matching selector is found.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- supportsCommand:` (page 1372)

Declared In

`NSScriptClassDescription.h`

suiteName

Returns the name of the receiver's suite.

```
- (NSString *)suiteName
```

Return Value

The receiver's suite name. Within an application's scriptability information, named suites contain related sets of information.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- className` (page 1365)

Declared In

`NSScriptClassDescription.h`

superclassDescription

Returns the class description instance for the superclass of the receiver's class.

```
- (NSScriptClassDescription *)superclassDescription
```

Return Value

A class description instance that describes the superclass of the receiver's class. Returns `nil` if the class has no superclass.

Discussion

The instance of `NSScriptClassDescription` that describes the superclass can be in the same suite as the receiver or in a different suite.

Availability

Available in Mac OS X v10.0 and later.

See Also
- `classDescriptionForKey:` (page 1365)

Declared In
`NSScriptClassDescription.h`

supportsCommand:

Returns a Boolean value indicating whether the receiver or any superclass supports the specified command.

```
- (BOOL)supportsCommand:(NSScriptCommandDescription *)commandDescription
```

Parameters

commandDescription

A description for a script command, such as `duplicate`, `make`, or `move`. Encapsulates the scriptability information for that command, such as its Objective-C selector, its argument names and types, and its return type (if any).

Return Value
`YES` if an the receiver or the instance of `NSScriptClassDescription` of any superclass of the receiver's class lists the command described by *commandDesc* among its supported commands; otherwise, `NO`.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `selectorForCommand:` (page 1370)

Declared In
`NSScriptClassDescription.h`

typeForKey:

Returns the name of the declared type of the attribute or relationship identified by the passed key.

```
- (NSString *)typeForKey:(NSString *)key
```

Parameters

key

The identifying key for an attribute, one-to-one relationship, or one-to-many relationship of the receiver.

Return Value
The name of the declared type of the attribute or relationship identified by *key*; for example, "NSString". Searches in the receiver first, then in any superclass. Returns `nil` if no match is found.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `isReadOnlyKey:` (page 1369)
- `keyWithAppleEventCode:` (page 1369)

Declared In
`NSScriptClassDescription.h`

NSScriptCoercionHandler Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptCoercionHandler.h
Companion guide	Cocoa Scripting Guide
Related sample code	Quartz Composer WWDC 2005 TextEdit Sketch-112 TextEditPlus

Overview

Provides a mechanism for converting one kind of scripting data to another. A shared instance of this class coerces (converts) object values to objects of another class, using information supplied by classes that register with it. Coercions frequently are required during key-value coding.

Tasks

Accessing the Application's Handler

`+ sharedCoercionHandler` (page 1376)
Returns the shared `NSScriptCoercionHandler` for the application.

Working with Handlers

`- coerceValue:toClass:` (page 1376)
Returns an object of a given class representing a given value.

`- registerCoercer:selector:toConvertFromClass:toClass:` (page 1377)
Registers a given object (typically a class) to handle coercions (conversions) from one given class to another.

Class Methods

sharedCoercionHandler

Returns the shared `NSScriptCoercionHandler` for the application.

```
+ (NSScriptCoercionHandler *)sharedCoercionHandler
```

Return Value
The shared `NSScriptCoercionHandler` for the application.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus

Declared In
`NSScriptCoercionHandler.h`

Instance Methods

coerceValue:toClass:

Returns an object of a given class representing a given value.

```
- (id)coerceValue:(id)value toClass:(Class)toClass
```

Parameters

value
The value to coerce.

toClass
The class with which to represent *value*.

Return Value
An object of the class *toClass* representing the value specified by *value*. Returns `nil` if an error occurs.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus

Declared In
`NSScriptCoercionHandler.h`

registerCoercer:selector:toConvertFromClass:toClass:

Registers a given object (typically a class) to handle coercions (conversions) from one given class to another.

```
- (void)registerCoercer:(id)coercer selector:(SEL)selector
    toConvertFromClass:(Class)fromClass toClass:(Class)toClass
```

Parameters

coercer

The object that performs the coercion. *coercer* should typically be a class object.

selector

A selector that specifies the method to perform the coercion. *selector* should typically be a factory method, and must take two arguments. The first is the value to be converted. The second is the class to convert it to.

fromClass

The class for which instances are coerced.

toClass

The class to which instances of *fromClass* are coerced.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSScriptCoercionHandler.h

NSScriptCommand Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptCommand.h
Companion guide	Cocoa Scripting Guide
Related sample code	Quartz Composer WWDC 2005 TextEdit SimpleCarbonAppleScript SimpleScriptingVerbs Sketch-112 TextEditPlus

Overview

An instance of `NSScriptCommand` represents a scripting statement, such as `set word 5 of the front document to word 1 of the second document`, and contains the information needed to perform the operation specified by the statement.

When an Apple event reaches a Cocoa application, Cocoa's built-in scripting support transforms it into a script command (that is, an instance of `NSScriptCommand` or one of the subclasses provided by Cocoa scripting or by your application) and executes the command in the context of the application. Executing a command means either invoking the selector associated with the command on the object or objects designated to receive the command, or having the command perform its default implementation method (`performDefaultImplementation` (page 1387)).

Your application most likely calls methods of `NSScriptCommand` to extract the command arguments. You do this either in the `performDefaultImplementation` method of a command subclass you have created, or in an object method designated as the selector to handle a particular command.

As part of Cocoa's standard scripting implementation, `NSScriptCommand` and its subclasses can handle the default command set for AppleScript's Standard suite for most applications without any subclassing. The Standard suite includes commands such as `copy`, `count`, `create`, `delete`, `exists`, and `move`, as well as common object classes such as `application`, `document`, and `window`.

For more information on working with script commands, see Script Commands in *Cocoa Scripting Guide*.

Adopted Protocols

NSCoding
- `- encodeWithCoder:` (page 2034)
- `- initWithCoder:` (page 2034)

Tasks

Initializing a Script Command

`- initWithCommandDescription:` (page 1386)
Returns an a script command object initialized from the passed command description.

Getting the Current Command

`+ currentCommand` (page 1382)
If a command is being executed in the current thread by Cocoa scripting's built-in Apple event handling, return the command.

Getting the Apple Event

`- appleEvent` (page 1383)
If the receiver was constructed by Cocoa scripting's built-in Apple event handling, returns the Apple event descriptor from which it was constructed.

Executing the Command

`- executeCommand` (page 1385)
Executes the command if it is valid and returns the result, if any.

`- performDefaultImplementation` (page 1387)
Overridden by subclasses to provide a default implementation for the command represented by the receiver.

Accessing Receivers

`- evaluatedReceivers` (page 1385)
Returns the object or objects to which the command is to be sent (called both the “receivers” or “targets” of script commands).

`- receiversSpecifier` (page 1387)
Returns the object specifier that, when evaluated, yields the receiver or receivers of the command.

- `setReceiversSpecifier:` (page 1390)
  Sets the object specifier to *receiversSpec* that, when evaluated, indicates the receiver or receivers of the command.

Accessing Arguments

- `arguments` (page 1383)
  Returns the arguments of the command.
- `evaluatedArguments` (page 1384)
  Returns a dictionary containing the arguments of the command, evaluated from object specifiers to objects if necessary. The keys in the dictionary are the argument names.
- `setArguments:` (page 1390)
  Sets the arguments of the command to *args*.

Accessing the Direct Parameter

- `directParameter` (page 1384)
  Returns the object that corresponds to the direct parameter of the Apple event from which the receiver derives.
- `setDirectParameter:` (page 1390)
  Sets the object that corresponds to the direct parameter of the Apple event from which the receiver derives.

Getting Command Information

- `commandDescription` (page 1383)
  Returns the command description for the command.
- `isWellFormed` (page 1386)
  Returns a Boolean value indicating whether the receiver is well formed according to its command description.

Handling Script Execution Errors

- `scriptErrorExpectedTypeDescriptor` (page 1388)
  Returns the type descriptor that was put in the reply Apple event if the sender requested a reply, execution of the receiver completed, and an error number was set.
- `scriptErrorNumber` (page 1388)
  Returns the script error number, if any, associated with execution of the command.
- `scriptErrorOffendingObjectDescriptor` (page 1389)
  Returns the object descriptor that was put in the reply Apple event if the sender requested a reply, execution of the receiver completed, and an error number was set.
- `scriptErrorString` (page 1389)
  Returns the script error string, if any, associated with execution of the command.

- `setScriptErrorExpectedTypeDescriptor:` (page 1391)
  Sets a descriptor for the expected type that will be put in the reply Apple event if the sender requested a reply, execution of the receiver completes, and an error number was set.
- `setScriptErrorOffendingObjectDescriptor:` (page 1392)
  Sets a descriptor for an object that will be put in the reply Apple event if the sender requested a reply, execution of the receiver completes, and an error number was set.
- `setScriptErrorNumber:` (page 1391)
  Sets a script error number that is associated with the execution of the command and is returned in the reply Apple event, if a reply was requested by the sender.
- `setScriptErrorString:` (page 1392)
  Sets a script error string that is associated with execution of the command.

Suspending and Resuming Commands

- `suspendExecution` (page 1393)
  Suspends the execution of the receiver.
- `resumeExecutionWithResult:` (page 1387)
  If a successful, unmatched, invocation of `suspendExecution` (page 1393) has been made, resume the execution of the command.

Class Methods

currentCommand

If a command is being executed in the current thread by Cocoa scripting's built-in Apple event handling, return the command.

```
+ (NSScriptCommand *)currentCommand
```

Discussion
A command is being executed in the current thread by Cocoa scripting's built-in Apple event handling if an instance of `NSScriptCommand` is handling an `executeCommand` (page 1385) message at this instant as the result of the dispatch of an Apple event. Returns `nil` otherwise. `setScriptErrorNumber:` (page 1391) and `setScriptErrorString:` (page 1392) messages sent to the returned command object will affect the reply event sent to the sender of the event from which the command was constructed, if the sender has requested a reply.

A suspended command is not considered the current command. If a command is suspended and no other command is being executed in the current thread, `currentCommand` returns `nil`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`NSScriptCommand.h`

Instance Methods

appleEvent

If the receiver was constructed by Cocoa scripting's built-in Apple event handling, returns the Apple event descriptor from which it was constructed.

```
- (NSAppleEventDescriptor *)appleEvent
```

Discussion
The effects of mutating or retaining this descriptor are undefined, although it may be copied.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`NSScriptCommand.h`

arguments

Returns the arguments of the command.

```
- (NSDictionary *)arguments
```

Discussion
If there are no arguments, returns an empty `NSDictionary` object. When you subclass `NSScriptCommand` or one of its subclasses, you rarely call this method because it returns the arguments directly, without evaluating any arguments that are object specifiers. If any of a command's arguments may be object specifiers, which is generally the case, call `evaluatedArguments` (page 1384) instead.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setArguments:` (page 1390)

Declared In
`NSScriptCommand.h`

commandDescription

Returns the command description for the command.

```
- (NSScriptCommandDescription *)commandDescription
```

Discussion
Once a command is created, its command description is immutable.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `isWellFormed` (page 1386)

Declared In
`NSScriptCommand.h`

directParameter

Returns the object that corresponds to the direct parameter of the Apple event from which the receiver derives.

```
- (id)directParameter
```

Return Value
An object. Returns `nil` if the received Apple event doesn't contain a direct parameter.

Discussion
For example, the direct parameter of a `print documents` Apple event contains a list of documents. This method may return the same object or objects returned by `receiversSpecifier` (page 1387).

Availability
Available in Mac OS X v10.0 and later.

See Also
- `setDirectParameter:` (page 1390)

Related Sample Code
SimpleCarbonAppleScript
SimpleScriptingVerbs

Declared In
`NSScriptCommand.h`

evaluatedArguments

Returns a dictionary containing the arguments of the command, evaluated from object specifiers to objects if necessary. The keys in the dictionary are the argument names.

```
- (NSDictionary *)evaluatedArguments
```

Discussion
Arguments initially can be either a normal object or an object specifier such as `word 5` (represented as an instance of an `NSScriptObjectSpecifier` subclass). If arguments are object specifiers, the receiver evaluates them before using the referenced objects. Returns `nil` if the command is not well formed. Also returns `nil` if an object specifier does not evaluate to an object or if there is no type defined for the argument in the command description.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `isWellFormed` (page 1386)
- `arguments` (page 1383)

- setArguments: (page 1390)

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
SimpleScriptingVerbs
Sketch-112
TextEditPlus

Declared In
NSScriptCommand.h

evaluatedReceivers

Returns the object or objects to which the command is to be sent (called both the "receivers" or "targets" of script commands).

```
- (id)evaluatedReceivers
```

Discussion
It evaluates receivers, which are always object specifiers, to a proper object. If the command does not specify a receiver, or if the receiver doesn't accept the command, it returns nil.

Availability
Available in Mac OS X v10.0 and later.

See Also
- receiversSpecifier (page 1387)
- setReceiversSpecifier: (page 1390)

Declared In
NSScriptCommand.h

executeCommand

Executes the command if it is valid and returns the result, if any.

```
- (id)executeCommand
```

Discussion
Before this method executes the command (through NSInvocation mechanisms), it evaluates all object specifiers involved in the command, validates that the receivers can actually handle the command, and verifies that the types of any arguments that were initially object specifiers are valid.

You shouldn't have to override this method. If the command's receivers want to handle the command themselves, this method invokes their defined handler. Otherwise, it invokes performDefaultImplementation (page 1387).

Availability
Available in Mac OS X v10.0 and later.

See Also
- evaluatedArguments (page 1384)

- evaluatedReceivers (page 1385)

Declared In
NSScriptCommand.h

initWithCommandDescription:

Returns an a script command object initialized from the passed command description.

```
- (id)initWithCommandDescription:(NSScriptCommandDescription *)commandDesc
```

Parameters

commandDesc
A command description for the command to be created.

Return Value
A newly initialized instance of NSScriptCommand or a subclass.

Discussion
To make this command object usable, you must set its receiving objects and arguments (if any) after invoking this method.

Availability
Available in Mac OS X v10.0 and later.

See Also
- setArguments: (page 1390)
- setReceiversSpecifier: (page 1390)

Declared In
NSScriptCommand.h

isWellFormed

Returns a Boolean value indicating whether the receiver is well formed according to its command description.

```
- (BOOL)isWellFormed
```

Discussion
The method ensures that there is a description of the command and that the number of arguments and the types of non-specifier arguments conform to the command description.

Availability
Available in Mac OS X v10.0 and later.

See Also
- commandDescription (page 1383)

Declared In
NSScriptCommand.h

performDefaultImplementation

Overridden by subclasses to provide a default implementation for the command represented by the receiver.

```
- (id)performDefaultImplementation
```

Discussion
Do not invoke this method directly. `executeCommand` (page 1385) invokes this method when the command being executed is not supported by the class of the objects receiving the command. The default implementation returns `nil`.

You need to create a subclass of `NSScriptCommand` only if you need to provide a default implementation of a command.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptCommand.h`

receiversSpecifier

Returns the object specifier that, when evaluated, yields the receiver or receivers of the command.

```
- (NSScriptObjectSpecifier *)receiversSpecifier
```

Discussion
The receiver is typically a container. For example, if the original command is `get the third paragraph of the first document`, the receiver specifier is `the first document`—it's the document that knows how to get or set words or paragraphs it contains.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `evaluatedReceivers` (page 1385)
- `setReceiversSpecifier:` (page 1390)

Declared In
`NSScriptCommand.h`

resumeExecutionWithResult:

If a successful, unmatched, invocation of `suspendExecution` (page 1393) has been made, resume the execution of the command.

```
- (void)resumeExecutionWithResult:(id)result
```

Discussion
Resumes the execution of the command if a successful, unmatched, invocation of `suspendExecution` (page 1393) has been made—otherwise, does nothing. The value for *`result`* is dependent on the segment of command execution that was suspended:

- If `suspendExecution` was invoked from within a command handler of one of the command's receivers, *`result`* is considered to be the return value of the handler. Unless the command has received a `setScriptErrorNumber:` (page 1391) message with a nonzero error number, execution of the command will continue and the command handlers of other receivers will be invoked.
- If `suspendExecution` was invoked from within an override of `performDefaultImplementation` (page 1387) the result is treated as if it were the return value of the invocation of `performDefaultImplementation`.

`resumeExecutionWithResult:` may be invoked in any thread, not just the one in which the corresponding invocation of `suspendExecution` (page 1393) occurred.

Important: The script command handler that is being executed when `suspendExecution` is invoked must return before you invoke `resumeExecutionWithResult:`. That is, it is not valid to suspend a command's execution and then resume it immediately.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`NSScriptCommand.h`

scriptErrorExpectedTypeDescriptor

Returns the type descriptor that was put in the reply Apple event if the sender requested a reply, execution of the receiver completed, and an error number was set.

```
- (NSAppleEventDescriptor *)scriptErrorExpectedTypeDescriptor
```

Return Value
A descriptor that specifies a type.

Discussion
When an error occurs during script command execution because an Apple event descriptor wasn't of the expected type, and the sender requested a reply, Cocoa scripting returns a descriptor for the expected type in a reply Apple event. You can invoke `setScriptErrorExpectedTypeDescriptor:` (page 1391) to set this descriptor directly.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSScriptCommand.h`

scriptErrorNumber

Returns the script error number, if any, associated with execution of the command.

```
- (int)scriptErrorNumber
```

Discussion
When you subclass NSScriptCommand or one of its subclasses, you shouldn't need to override this method.

For error conditions specific to your application you can define your own error return values. For some common errors, you may want to return error values defined in `MacErrors.h`, a header in `CarbonCore.framework` (a subframework of `CoreServices.framework`). Look for error constants that start with `errAE`. For example, `errAEEventNotHandled` indicates a handler wasn't able to handle the Apple event.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setScriptErrorNumber:` (page 1391)

Declared In
`NSScriptCommand.h`

scriptErrorOffendingObjectDescriptor

Returns the object descriptor that was put in the reply Apple event if the sender requested a reply, execution of the receiver completed, and an error number was set.

```
- (NSAppleEventDescriptor *)scriptErrorOffendingObjectDescriptor
```

Return Value
A descriptor that specifies an object.

Discussion
When an error that occurs during script command execution is caused by a specific object, and the sender requested a reply, Cocoa scripting returns a descriptor for the offending object in a reply Apple event. You can invoke `setScriptErrorOffendingObjectDescriptor:` (page 1392) to set this descriptor directly.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- setScriptErrorOffendingObjectDescriptor:` (page 1392)

Declared In
`NSScriptCommand.h`

scriptErrorString

Returns the script error string, if any, associated with execution of the command.

```
- (NSString *)scriptErrorString
```

Discussion
When you subclass `NSScriptCommand` or one of its subclasses, you shouldn't need to override this method.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setScriptErrorString:` (page 1392)

Declared In
NSScriptCommand.h

setArguments:

Sets the arguments of the command to *args*.

```
- (void)setArguments:(NSDictionary *)args
```

Discussion
Each argument in the dictionary is identified by the same name key used for the argument in the command's class declaration in the script suite file.

Availability
Available in Mac OS X v10.0 and later.

See Also
- arguments (page 1383)
- evaluatedArguments (page 1384)

Declared In
NSScriptCommand.h

setDirectParameter:

Sets the object that corresponds to the direct parameter of the Apple event from which the receiver derives.

```
- (void)setDirectParameter:(id)directParameter
```

Parameters
directParameter
An object to be set as the direct parameter.

Discussion
You don't normally override this method.

Availability
Available in Mac OS X v10.0 and later.

See Also
- directParameter (page 1384)

Declared In
NSScriptCommand.h

setReceiversSpecifier:

Sets the object specifier to *receiversSpec* that, when evaluated, indicates the receiver or receivers of the command.

```
- (void)setReceiversSpecifier:(NSScriptObjectSpecifier *)receiversSpec
```

Discussion
If you create a subclass of `NSScriptCommand`, you don't necessarily need to override this method, though some of Cocoa's subclasses do. An override should perform the same function as the superclass method, with a critical difference: it causes the container specifier part of the passed-in object specifier to become the receiver specifier of the command, and the key part of the passed-in object specifier to become the key specifier. In an override, for example, if *`receiversRef`* is a specifier for `the third rectangle of the first document`, the receiver specifier is `the first document` while the key specifier is `the third rectangle`.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `evaluatedReceivers` (page 1385)
- `receiversSpecifier` (page 1387)

Declared In
`NSScriptCommand.h`

setScriptErrorExpectedTypeDescriptor:

Sets a descriptor for the expected type that will be put in the reply Apple event if the sender requested a reply, execution of the receiver completes, and an error number was set.

```
- (void)setScriptErrorExpectedTypeDescriptor:(NSAppleEventDescriptor
    *)errorExpectedTypeDescriptor
```

Parameters
`errorExpectedTypeDescriptor`
A descriptor that specifies a type.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `scriptErrorExpectedTypeDescriptor` (page 1388)

Declared In
`NSScriptCommand.h`

setScriptErrorNumber:

Sets a script error number that is associated with the execution of the command and is returned in the reply Apple event, if a reply was requested by the sender.

```
- (void)setScriptErrorNumber:(int)errorNumber
```

Parameters
`errorNumber`
An error number to associate with the command.

Discussion
If you override `performDefaultImplementation` (page 1387) and an error occurs, you should call this method to supply an appropriate error number. In fact, any script handler should call this method when an error occurs. The error number you supply is returned in the reply Apple event.

Invoking `setScriptErrorNumber:` causes an error message to be displayed. To associate a specific error message with the error number, you invoke `setScriptErrorString:` (page 1392). This make sense, for example, when you set an error number that is specific to your application, or when you can supply a specific and useful error message to the user.

If `setScriptErrorNumber:` is invoked on an `NSScriptCommand` with multiple receivers, the command will stop sending command handling messages to more receivers.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- scriptErrorNumber` (page 1388)

Related Sample Code
Sketch-112

Declared In
`NSScriptCommand.h`

setScriptErrorOffendingObjectDescriptor:

Sets a descriptor for an object that will be put in the reply Apple event if the sender requested a reply, execution of the receiver completes, and an error number was set.

```
- (void)setScriptErrorOffendingObjectDescriptor:(NSAppleEventDescriptor
    *)errorOffendingObjectDescriptor
```

Parameters
errorOffendingObjectDescriptor
A descriptor that specifies an object that was responsible for an error.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- scriptErrorOffendingObjectDescriptor` (page 1389)

Declared In
`NSScriptCommand.h`

setScriptErrorString:

Sets a script error string that is associated with execution of the command.

```
- (void)setScriptErrorString:(NSString *)errorString
```

Parameters

errorString

A string that describes an error.

Discussion

If you override `performDefaultImplementation` (page 1387) and an error occurs, you should call this method to supply a string that provides a useful explanation. In fact, any script handler should call this method when an error occurs.

Calling this method alone does not cause an error message to be be displayed—you must also call `setScriptErrorNumber:` (page 1391) to supply an error number.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- scriptErrorString` (page 1389)

Declared In

`NSScriptCommand.h`

suspendExecution

Suspends the execution of the receiver.

```
- (void)suspendExecution
```

Discussion

Suspends the execution of the receiver only if the receiver is being executed in the current thread by Cocoa scripting's built-in Apple event handling (that is, the receiver would be returned by `[NSScriptCommand currentCommand]`)—otherwise, does nothing. A matching invocation of `resumeExecutionWithResult:` (page 1387) must be made.

Important: The script command handler that is being executed when this method is invoked must return before the subsequent invocation of `resumeExecutionWithResult:` (page 1387). That is, it is not valid to suspend a command's execution and then resume it immediately.

Another command can execute while a command is suspended.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`NSScriptCommand.h`

Constants

NSScriptCommand—General Command Execution Errors

`NSScriptCommand` uses the following error codes for general command execution problems:

```
enum {
    NSNoScriptError = 0,
    NSReceiverEvaluationScriptError,
    NSKeySpecifierEvaluationScriptError,
    NSArgumentEvaluationScriptError,
    NSReceiversCantHandleCommandScriptError,
    NSRequiredArgumentsMissingScriptError,
    NSArgumentsWrongScriptError,
    NSUnknownKeyScriptError,
    NSInternalScriptError,
    NSOperationNotSupportedForKeyScriptError,
    NSCannotCreateScriptCommandError
};
```

Constants

`NSNoScriptError`
No error.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptCommand.h`.

`NSReceiverEvaluationScriptError`
The object or objects specified by the direct parameter to a command could not be found.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptCommand.h`.

`NSKeySpecifierEvaluationScriptError`
The object or objects specified by a key (for commands that support key specifiers) could not be found.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptCommand.h`.

`NSArgumentEvaluationScriptError`
The object specified by an argument could not be found.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptCommand.h`.

`NSReceiversCantHandleCommandScriptError`
The receivers don't support the command sent to them.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptCommand.h`.

`NSRequiredArgumentsMissingScriptError`
An argument (or more than one argument) is missing.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptCommand.h`.

`NSArgumentsWrongScriptError`
An argument (or more than one argument) is of the wrong type or is otherwise invalid.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptCommand.h`.

`NSUnknownKeyScriptError`
: An unidentified error occurred; indicates an error in the scripting support of your application.

  Available in Mac OS X v10.0 and later.

  Declared in `NSScriptCommand.h`.

`NSInternalScriptError`
: An unidentified internal error occurred; indicates an error in the scripting support of your application.

  Available in Mac OS X v10.0 and later.

  Declared in `NSScriptCommand.h`.

`NSOperationNotSupportedForKeyScriptError`
: The implementation of a scripting command signaled an error.

  Available in Mac OS X v10.0 and later.

  Declared in `NSScriptCommand.h`.

`NSCannotCreateScriptCommandError`
: Could not create the script command; an invalid or unrecognized Apple event was received.

  Available in Mac OS X v10.0 and later.

  Declared in `NSScriptCommand.h`.

Declared In
`NSScriptCommand.h`

NSScriptCommandDescription Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptCommandDescription.h
Companion guide	Cocoa Scripting Guide

Overview

An instance of `NSScriptCommandDescription` describes a script command that a Cocoa application supports.

A scriptable application provides scriptability information that describes the commands and objects scripters can use in scripts that target the application. An application's scripting information is collected automatically by an instance of `NSScriptSuiteRegistry`, which creates an `NSScriptCommandDescription` for each command it finds, caches these objects in memory, and installs a command handler for each command.

A script command instance stores the name, class, argument types, and return type of a command. For example, commands in AppleScript's Core suite include `clone`, `count`, `create`, `delete`, `exists`, and `move`.

The public methods of `NSScriptCommandDescription` are used primarily by Cocoa's built-in scripting support in responding to Apple events that target the application. Although you can subclass the `NSScriptCommandDescription` class, it is unlikely that you would need to do so, or to create instances of it.

Adopted Protocols

NSCoding
- `- encodeWithCoder:` (page 2034)
- `- initWithCoder:` (page 2034)

Tasks

Initializing a Script Command Description

- `initWithSuiteName:commandName:dictionary:` (page 1402)
  Initializes and returns a newly allocated instance of `NSScriptCommandDescription`.

Getting Basic Information About the Command

- `appleEventClassCode` (page 1399)
  Returns the four-character code for the Apple event class of the receiver's command.
- `appleEventCode` (page 1399)
  Returns the four-character code for the Apple event ID of the receiver's command.
- `commandClassName` (page 1401)
  Returns the name of the class that will be instantiated to handle the command.
- `commandName` (page 1401)
  Returns the name of the command.
- `suiteName` (page 1404)
  Returns the name of the suite that contains the command described by the receiver.

Getting Command Argument Information

- `appleEventCodeForArgumentWithName:` (page 1400)
  Returns the Apple event code for the specified command argument of the receiver.
- `argumentNames` (page 1401)
  Returns the names (or keys) for all arguments of the receiver's command.
- `isOptionalArgumentWithName:` (page 1403)
  Returns a Boolean value that indicates whether the command argument identified by the specified argument key is an optional argument.
- `typeForArgumentWithName:` (page 1404)
  Returns the type of the command argument identified by the specified key.

Getting Command Return-Type Information

- `appleEventCodeForReturnType` (page 1400)
  Returns the Apple event code that identifies the command's return type.
- `returnType` (page 1403)
  Returns the return type of the command.

Creating Commands

- `createCommandInstance` (page 1402)
  Creates and returns an instance of the command object described by the receiver.
- `createCommandInstanceWithZone:` (page 1402)
  Creates and returns an instance of the command object described by the receiver in the specified memory zone.

Instance Methods

appleEventClassCode

Returns the four-character code for the Apple event class of the receiver's command.

```
- (FourCharCode)appleEventClassCode
```

Return Value
The Apple event code associated with the receiver's command. This is the primary code used to identify the command in Apple events.

Discussion
In an Apple event that specifies a script command, two four character codes—the event class and event ID—together identify the command. You use this method to obtain the event class. You use `appleEventCode` (page 1399) to obtain the event ID.

For example, commands in AppleScript's Core suite, such as `clone`, `count`, and `create`, have an event class code of `'core'`. This code and the event ID code returned by `appleEventCode` together specify the necessary information for identifying and dispatching an Apple event.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptCommandDescription.h`

appleEventCode

Returns the four-character code for the Apple event ID of the receiver's command.

```
- (FourCharCode)appleEventCode
```

Return Value
The code for the event ID of the receiver's command.

Discussion
This value of the event ID returned by this method, together with the event class code returned by `appleEventClassCode` (page 1399), specifies the necessary information for identifying and dispatching an Apple event.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `appleEventCodeForArgumentWithName:` (page 1400)
- `appleEventCodeForReturnType` (page 1400)

Declared In
`NSScriptCommandDescription.h`

appleEventCodeForArgumentWithName:

Returns the Apple event code for the specified command argument of the receiver.

```
- (FourCharCode)appleEventCodeForArgumentWithName:(NSString *)argumentName
```

Parameters

argumentName
The argument name (used as a key) for which to obtain the corresponding Apple event code.

Return Value
The code for the specified argument.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `argumentNames` (page 1401)

Declared In
`NSScriptCommandDescription.h`

appleEventCodeForReturnType

Returns the Apple event code that identifies the command's return type.

```
- (FourCharCode)appleEventCodeForReturnType
```

Return Value
The event code for the command's return type.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `appleEventCodeForArgumentWithName:` (page 1400)
- `returnType` (page 1403)

Declared In
`NSScriptCommandDescription.h`

argumentNames

Returns the names (or keys) for all arguments of the receiver's command.

```
- (NSArray *)argumentNames
```

Return Value
The array of argument names. If there are no arguments for the command, returns an empty array.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptCommandDescription.h`

commandClassName

Returns the name of the class that will be instantiated to handle the command.

```
- (NSString *)commandClassName
```

Return Value
The Objective-C class name (for example, `"NSGetCommand"`). This is always `NSScriptCommand` or a subclass.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- commandName` (page 1401)

Declared In
`NSScriptCommandDescription.h`

commandName

Returns the name of the command.

```
- (NSString *)commandName
```

Return Value
The command name as it appears in the application's scriptability information; may be different from what is displayed to the scripter.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- commandClassName` (page 1401)

Declared In
`NSScriptCommandDescription.h`

createCommandInstance

Creates and returns an instance of the command object described by the receiver.

```
- (NSScriptCommand *)createCommandInstance
```

Return Value
The command object, instantiated from `NSScriptCommand` or a subclass.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptCommandDescription.h`

createCommandInstanceWithZone:

Creates and returns an instance of the command object described by the receiver in the specified memory zone.

```
- (NSScriptCommand *)createCommandInstanceWithZone:(NSZone *)zone
```

Parameters

zone
The memory zone from which to allocate the command.

Return Value
The command object, instantiated from `NSScriptCommand` or a subclass.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptCommandDescription.h`

initWithSuiteName:commandName:dictionary:

Initializes and returns a newly allocated instance of `NSScriptCommandDescription`.

```
- (id)initWithSuiteName:(NSString *)suiteName commandName:(NSString *)commandName
    dictionary:(NSDictionary *)commandDeclaration
```

Parameters

suiteName
The name of the suite (in the application's scriptability information) that the command belongs to. For example, `"AppName Suite"`.

commandName
The name of the script command that this instance describes.

commandDeclaration
A command declaration dictionary of the sort that is valid in script suite property list files. This dictionary provides information about the command such as its argument names and types and return type (if any).

Return Value
The initialized command description instance. Returns `nil` if the event constant or class name for the command description is missing; also returns `nil` if the return type or argument values are of the wrong type.

Discussion
This method registers `self` with the application's global instance of `NSScriptSuiteRegistry` and also registers all command arguments with the registry.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptCommandDescription.h`

isOptionalArgumentWithName:

Returns a Boolean value that indicates whether the command argument identified by the specified argument key is an optional argument.

```
- (BOOL)isOptionalArgumentWithName:(NSString *)argumentName
```

Parameters

argumentName
Argument name (used as a key) that identifies the command argument to examine.

Return Value
`YES` if the specified argument exists and is optional; otherwise, `NO`.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `argumentNames` (page 1401)

Declared In
`NSScriptCommandDescription.h`

returnType

Returns the return type of the command.

```
- (NSString *)returnType
```

Return Value
The receiver's command return type; for example, `"NSNumber"` or `"NSDictionary"`).

Availability
Available in Mac OS X v10.0 and later.

See Also
- `appleEventCodeForReturnType` (page 1400)

Declared In
NSScriptCommandDescription.h

suiteName

Returns the name of the suite that contains the command described by the receiver.

```
- (NSString *)suiteName
```

Return Value
The receiver's suite name. Within an application's scriptability information, named suites contain related sets of information.

Availability
Available in Mac OS X v10.0 and later.

See Also
- appleEventCode (page 1399)

Declared In
NSScriptCommandDescription.h

typeForArgumentWithName:

Returns the type of the command argument identified by the specified key.

```
- (NSString *)typeForArgumentWithName:(NSString *)argumentName
```

Parameters
argumentName
Argument name (used as a key) that identifies the command argument to examine.

Return Value
The type of the specified command argument. Returns `nil` if there is no such argument.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptCommandDescription.h

NSScriptExecutionContext Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptExecutionContext.h
Companion guide	Cocoa Scripting Guide
Related sample code	SimpleCarbonAppleScript

Overview

An `NSScriptExecutionContext` object is a shared instance (there is only one instance of the class) that represents the context in which the current script command is executed. `NSScriptExecutionContext` tracks global state relating to the command being executed, especially the top-level container object (that is, the container implied by a specifier object that specifies no container) used in an evaluation of an `NSScriptObjectSpecifier` object.

In most cases, the top-level container for a complete series of nested object specifiers is automatically set to the application object (`NSApp`), and you can get this object with the `topLevelObject` (page 1408) method. But you can also set this top-level container to something else (using `setTopLevelObject:` (page 1408)) if the situation warrants it.

It is unlikely that you will need to subclass `NSScriptExecutionContext`.

Tasks

Getting the Current Context

`+ sharedScriptExecutionContext` (page 1406)
Returns the shared `NSScriptExecutionContext` instance.

Getting and Setting the Container Object

- `topLevelObject` (page 1408)
  Returns the top-level object for an object-specifier evaluation.
- `setTopLevelObject:` (page 1408)
  Sets the top-level object for an object-specifier evaluation.
- `objectBeingTested` (page 1406)
  Returns the top-level container object currently being tested in a “whose” qualifier.
- `setObjectBeingTested:` (page 1407)
  Sets the top-level container object currently being tested in a “whose” qualifier to a given object.
- `rangeContainerObject` (page 1407)
  Returns the top-level container object for an object specifier (encapsulated in an `NSRangeSpecifier` object) that represents the first or last element in a range of elements.
- `setRangeContainerObject:` (page 1408)
  Sets the top-level container object for a range-specifier evaluation to a give object.

Class Methods

sharedScriptExecutionContext

Returns the shared `NSScriptExecutionContext` instance.

```
+ (NSScriptExecutionContext *)sharedScriptExecutionContext
```

Return Value
The shared `NSScriptExecutionContext` instance, creating it first if it doesn’t exist.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
SimpleCarbonAppleScript

Declared In
`NSScriptExecutionContext.h`

Instance Methods

objectBeingTested

Returns the top-level container object currently being tested in a “whose” qualifier.

```
- (id)objectBeingTested
```

Return Value
The top-level container object currently being tested in a "whose" qualifier. Returns `nil` if such an object does not exist.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setObjectBeingTested:` (page 1407)
`containerIsObjectBeingTested` (page 1415) (NSScriptObjectSpecifier)

Declared In
`NSScriptExecutionContext.h`

rangeContainerObject

Returns the top-level container object for an object specifier (encapsulated in an `NSRangeSpecifier` object) that represents the first or last element in a range of elements.

```
- (id)rangeContainerObject
```

Return Value
The top-level container object for an object specifier (encapsulated in an `NSRangeSpecifier` object) that represents the first or last element in a range of elements.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setObjectBeingTested:` (page 1407)
`containerIsRangeContainerObject` (page 1416) (`NSScriptObjectSpecifier`)

Declared In
`NSScriptExecutionContext.h`

setObjectBeingTested:

Sets the top-level container object currently being tested in a "whose" qualifier to a given object.

```
- (void)setObjectBeingTested:(id)object
```

Parameters
object
The top-level container object currently being tested.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- objectBeingTested` (page 1406)

Declared In
`NSScriptExecutionContext.h`

setRangeContainerObject:

Sets the top-level container object for a range-specifier evaluation to a give object.

```
- (void)setRangeContainerObject:(id)container
```

Parameters

container

The top-level container object for a range-specifier evaluation.

Discussion

Instances of `NSRangeSpecifier` contain object specifiers representing the first or last element in a range of elements, and these specifiers are evaluated in the context of *container*.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `rangeContainerObject` (page 1407)

Declared In

`NSScriptExecutionContext.h`

setTopLevelObject:

Sets the top-level object for an object-specifier evaluation.

```
- (void)setTopLevelObject:(id)anObject
```

Parameters

anObject

The top-level object for an object-specifier evaluation.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `topLevelObject` (page 1408)

Related Sample Code

SimpleCarbonAppleScript

Declared In

`NSScriptExecutionContext.h`

topLevelObject

Returns the top-level object for an object-specifier evaluation.

```
- (id)topLevelObject
```

Return Value

The top-level object for an object-specifier evaluation.

Discussion
For applications, this object is automatically set to the application object, but can be set to some other container object.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setTopLevelObject:` (page 1408)

Declared In
`NSScriptExecutionContext.h`

NSScriptObjectSpecifier Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptObjectSpecifiers.h
Companion guide	Cocoa Scripting Guide
Related sample code	Quartz Composer WWDC 2005 TextEdit SimpleScriptingObjects Sketch-112 TextEditPlus

Overview

`NSScriptObjectSpecifier` is the abstract superclass for classes that instantiate objects called "object specifiers." An object specifier represents an AppleScript reference form, which is a natural-language expression such as `words 10 through 20` or `front document` or `words whose color is red`.

The scripting system maps these words or phrases to attributes and relationships of scriptable objects. A reference form rarely occurs in isolation; usually a script statement consists of a series of reference forms preceded by a command and typically connected to each other by `of`, such as:

```
get words whose color is blue of paragraph 10 of front document
```

The expression `words whose color is blue of paragraph 10 of front document` specifies a location in the application's AppleScript object model—the objects the application makes available to scripters. The classes of objects in the object model often closely match the classes of actual objects in the application, but they are not required to. An object specifier locates objects in the running application that correspond to the specified object model objects.

Your application typically creates object specifiers when it implements the `objectSpecifier` method for its scriptable classes. That method is defined by the `NSScriptObjectSpecifiers` protocol.

It is unlikely that you would ever need to create your own subclass of `NSScriptObjectSpecifier`; the set of valid AppleScript reference forms is determined by Apple Computer and object specifier classes are already implemented for this set. If for some reason you do need to create a subclass, you must override the primitive method `indicesOfObjectsByEvaluatingWithContainer:count:` (page 1418) to return indices to the

elements within the container whose values are matched with the child specifier's key. In addition, you probably need to declare any special instance variables and implement an initializer that invokes super's designated initializer, `initWithContainerClassDescription:containerSpecifier:key:` (page 1418), and initializes these variables.

For a comprehensive treatment of object specifiers, including sample code, see Object Specifiers in *Cocoa Scripting Guide*.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 2034)
- `initWithCoder:` (page 2034)

Tasks

Obtaining an Object Specifier for a Descriptor

`+ objectSpecifierWithDescriptor:` (page 1414)
Returns a new object specifier for an Apple event descriptor.

Initializing an Object Specifier

`- initWithContainerClassDescription:containerSpecifier:key:` (page 1418)
Returns an `NSScriptObjectSpecifier` object initialized with the given attributes.

`- initWithContainerSpecifier:key:` (page 1418)
Returns an `NSScriptObjectSpecifier` object initialized with a given container specifier and key.

Evaluating an Object Specifier

`- indicesOfObjectsByEvaluatingWithContainer:count:` (page 1418)
This primitive method must be overridden by subclasses to return a pointer to an array of indices identifying objects in the key of a given container that are identified by the receiver of the message.

`- objectsByEvaluatingSpecifier` (page 1420)
Returns the actual object represented by the nested series of object specifiers.

`- objectsByEvaluatingWithContainers:` (page 1420)
Returns the actual object or objects specified by the receiver as evaluated in the context of given container object.

Getting, Testing, and Setting Containers

- `containerClassDescription` (page 1415)
  Returns the class description of the object indicated by the receiver's container specifier.
- `containerIsObjectBeingTested` (page 1415)
  If the receiver's container specifier is `nil`, returns a Boolean value that indicates whether the receiver's container is the object involved in a specifier test.
- `containerIsRangeContainerObject` (page 1416)
  If the receiver's container specifier is `nil`, returns a Boolean value that indicates whether the container for the receiver contains the range of elements represented by an `NSRangeSpecifier`.
- `containerSpecifier` (page 1416)
  Returns the receiver's container specifier.
- `setContainerClassDescription:` (page 1421)
  Sets the class description of the receiver's container specifier to a given specifier.
- `setContainerIsObjectBeingTested:` (page 1422)
  Sets whether the receiver's container should be an object involved in a filter reference or the top-level object.
- `setContainerSpecifier:` (page 1422)
  Sets the container specifier of the receiver.
- `setContainerIsRangeContainerObject:` (page 1422)
  Sets whether the receiver's container is to be the container for a range specifier or a top-level object.

Getting and Setting Child References

- `childSpecifier` (page 1414)
  Returns the receiver's child reference.
- `setChildSpecifier:` (page 1421)
  Sets the receiver's child reference.

Getting and Setting Object Keys

- `key` (page 1419)
  Returns the key of the receiver.
- `keyClassDescription` (page 1419)
  Returns the class description of the objects specified by the receiver.
- `setKey:` (page 1423)
  Sets the key of the receiver.

Getting Evaluation Errors

- `evaluationErrorSpecifier` (page 1417)
  Returns the object specifier in which an evaluation error occurred.

- `evaluationErrorNumber` (page 1417)
  Returns the constant identifying the type of error that caused evaluation to fail.
- `setEvaluationErrorNumber:` (page 1423)
  Sets the value of the evaluation error.

Getting a Descriptor for the Object Specifier

- `descriptor` (page 1416)
  Returns an Apple event descriptor that represents the receiver.

Class Methods

objectSpecifierWithDescriptor:

Returns a new object specifier for an Apple event descriptor.

```
+ (NSScriptObjectSpecifier *)objectSpecifierWithDescriptor:(NSAppleEventDescriptor
    *)descriptor
```

Parameters

descriptor
  An Apple event descriptor. The descriptor must have the type `typeObjectSpecifier`.

Return Value
An object specifier, or `nil` if an error occurs.

Discussion
If `objectSpecifierWithDescriptor:` is invoked and fails during the execution of a script command, information about the error that caused the failure is recorded in `[NSScriptCommand currentCommand]`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSScriptObjectSpecifiers.h`

Instance Methods

childSpecifier

Returns the receiver's child reference.

```
- (NSScriptObjectSpecifier *)childSpecifier
```

Return Value
The receiver's child reference, that is, the object specifier evaluating to the object or objects that the receiver contains.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `setChildSpecifier:` (page 1421)

Declared In
`NSScriptObjectSpecifiers.h`

containerClassDescription

Returns the class description of the object indicated by the receiver's container specifier.

```
- (NSScriptClassDescription *)containerClassDescription
```

Return Value
The class description of the object indicated by the receiver's container specifier.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `setContainerClassDescription:` (page 1421)

Declared In
`NSScriptObjectSpecifiers.h`

containerIsObjectBeingTested

If the receiver's container specifier is `nil`, returns a Boolean value that indicates whether the receiver's container is the object involved in a specifier test.

```
- (BOOL)containerIsObjectBeingTested
```

Return Value
`YES` if the receiver's container is the object involved in a specifier test, otherwise `NO`.

Discussion
An example of a specifier test is `whose color is blue`). If the returned value is `YES`, then the top-level object is the object being tested (that is, the specifier has no container specifier).

Availability
Available in Mac OS X v10.0 and later.

See Also
- `objectBeingTested` (page 1406) (`NSScriptExecutionContext`)

Declared In
`NSScriptObjectSpecifiers.h`

containerIsRangeContainerObject

If the receiver's container specifier is `nil`, returns a Boolean value that indicates whether the container for the receiver contains the range of elements represented by an `NSRangeSpecifier`.

```
- (BOOL)containerIsRangeContainerObject
```

Return Value
`YES` if the container for the receiver contains the range of elements represented by an `NSRangeSpecifier`, otherwise `NO`.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `setContainerIsRangeContainerObject:` (page 1422)

Declared In
`NSScriptObjectSpecifiers.h`

containerSpecifier

Returns the receiver's container specifier.

```
- (NSScriptObjectSpecifier *)containerSpecifier
```

Return Value
The receiver's container specifier, which is the object specifier that must be evaluated to provide a context for the evaluation of the receiver.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `childSpecifier` (page 1414)
- `containerClassDescription` (page 1415)
- `setContainerSpecifier:` (page 1422)

Declared In
`NSScriptObjectSpecifiers.h`

descriptor

Returns an Apple event descriptor that represents the receiver.

```
- (NSAppleEventDescriptor *)descriptor
```

Return Value
An Apple event descriptor of type `typeObjectSpecifier`.

Discussion
If the receiver was created with `objectSpecifierWithDescriptor:` (page 1414), the passed-in descriptor is returned. Otherwise, a new descriptor is created and returned, autoreleased.

Availability
Available in Mac OS X v10.5 and later.

Declared In
NSScriptObjectSpecifiers.h

evaluationErrorNumber

Returns the constant identifying the type of error that caused evaluation to fail.

```
- (NSInteger)evaluationErrorNumber
```

Return Value
The constant identifying the type of error that caused evaluation to fail.

Discussion
This error code could be associated with the receiver or any container specifier "above" the receiver. Possible return values are defined in "Constants" (page 1424).

Availability
Available in Mac OS X v10.0 and later.

See Also
- evaluationErrorSpecifier (page 1417)

Declared In
NSScriptObjectSpecifiers.h

evaluationErrorSpecifier

Returns the object specifier in which an evaluation error occurred.

```
- (NSScriptObjectSpecifier *)evaluationErrorSpecifier
```

Return Value
The object specifier in which an evaluation error occurred.

Discussion
The object specifier failing to evaluate could be the receiver or any container specifier "above" the receiver.

Availability
Available in Mac OS X v10.0 and later.

See Also
- evaluationErrorNumber (page 1417)

Declared In
NSScriptObjectSpecifiers.h

indicesOfObjectsByEvaluatingWithContainer:count:

This primitive method must be overridden by subclasses to return a pointer to an array of indices identifying objects in the key of a given container that are identified by the receiver of the message.

```
- (NSInteger *)indicesOfObjectsByEvaluatingWithContainer:(id)aContainer
    count:(NSInteger *)numRefs
```

Discussion
This primitive method must be overridden by subclasses to return a pointer to an array of indices identifying objects in the key of the container *aContainer* that are identified by the receiver of the message. The method uses key-value coding to obtain values based on the receiver's key. It returns the number of such matching objects by indirection in *numRefs*. It returns `nil` directly and –1 via *numRefs* if all objects in the container (or the sole object) match the value of the receiver's key. This method is invoked by `objectsByEvaluatingWithContainers:` (page 1420). The default implementation returns `nil` directly and –1 indirectly via *numRefs*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptObjectSpecifiers.h`

initWithContainerClassDescription:containerSpecifier:key:

Returns an `NSScriptObjectSpecifier` object initialized with the given attributes.

```
- (id)initWithContainerClassDescription:(NSScriptClassDescription *)classDescription
    containerSpecifier:(NSScriptObjectSpecifier *)specifier key:(NSString *)key
```

Return Value
An `NSScriptObjectSpecifier` object initialized with container specifier *specifier*, key *key*, and the class description of the object specifier *classDescription*, derived from the value of the specifier's key.

Discussion
You should never pass `nil` for the value of *classDescription*. The receiver's child reference is set to `nil`.

This is the designated initializer for `NSScriptObjectSpecifier`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptObjectSpecifiers.h`

initWithContainerSpecifier:key:

Returns an `NSScriptObjectSpecifier` object initialized with a given container specifier and key.

```
- (id)initWithContainerSpecifier:(NSScriptObjectSpecifier *)specifier key:(NSString
    *)key
```

Return Value
An `NSScriptObjectSpecifier` object initialized with container specifier *specifier* and key *key*.

Discussion
The class description of the container is set automatically.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptObjectSpecifiers.h`

key

Returns the key of the receiver.

```
- (NSString *)key
```

Return Value
The key of the receiver.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `keyClassDescription` (page 1419)
- `setKey:` (page 1423)

Related Sample Code
Sketch-112

Declared In
`NSScriptObjectSpecifiers.h`

keyClassDescription

Returns the class description of the objects specified by the receiver.

```
- (NSScriptClassDescription *)keyClassDescription
```

Return Value
The class description of the objects specified by the receiver.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `key` (page 1419)
- `setKey:` (page 1423)

Related Sample Code
Sketch-112

Declared In
`NSScriptObjectSpecifiers.h`

objectsByEvaluatingSpecifier

Returns the actual object represented by the nested series of object specifiers.

```
- (id)objectsByEvaluatingSpecifier
```

Return Value
The actual object represented by the nested series of object specifiers.

Discussion
Recursively obtains the next container in a nested series of object specifiers until it reaches the top-level container specifier (which is either an `NSWhoseSpecifier` or the application object), after which it begins evaluating each object specifier (`objectsByEvaluatingWithContainers:` (page 1420)) going in the opposite direction (top-level to innermost) as it unwinds from the stack. Returns the actual object represented by the nested series of object specifiers. Returns `nil` if a container specifier could not be evaluated or if no top-level container specifier could be found. Thus `nil` can be a valid value or can indicate an error; you can use `evaluationErrorNumber` (page 1417) to determine if and which error occurred and `evaluationErrorSpecifier` (page 1417) to find the container specifier responsible for the error. In the normal course of command processing, this method is invoked by an `NSScriptCommand` object's `evaluatedArguments` (page 1384) and `evaluatedReceivers` (page 1385) methods, which take as message receiver the innermost object specifier.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- indicesOfObjectsByEvaluatingWithContainer:count:` (page 1418)

Related Sample Code
Sketch-112

Declared In
`NSScriptObjectSpecifiers.h`

objectsByEvaluatingWithContainers:

Returns the actual object or objects specified by the receiver as evaluated in the context of given container object.

```
- (id)objectsByEvaluatingWithContainers:(id)containers
```

Return Value
The actual object or objects specified by the receiver as evaluated in the context of its container object or objects (*containers*).

Discussion
Invokes `indicesOfObjectsByEvaluatingWithContainer:count:` (page 1418) on `self` to get an array of pointers to indices of elements in *containers* that have values paired with the message receiver's key. This method then uses key-value coding to obtain the object or objects associated with the key; it returns these objects or `nil` if there are no matching values in containers. If there are multiple matching values, they are returned in an `NSArray`; if matching values are `nil`, `NSNull` objects are substituted. If *containers* is an `NSArray`, the method recursively evaluates each element in the array and returns an `NSArray` with evaluated objects (including `NSNulls`) in their corresponding slots.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `objectsByEvaluatingSpecifier` (page 1420)

Related Sample Code
Sketch-112

Declared In
`NSScriptObjectSpecifiers.h`

setChildSpecifier:

Sets the receiver's child reference.

```
- (void)setChildSpecifier:(NSScriptObjectSpecifier *)child
```

Parameters
child
The receiver's child reference.

Discussion
Do not invoke this method directly; it is automatically invoked by `setContainerSpecifier:` (page 1422).

Availability
Available in Mac OS X v10.0 and later.

See Also
- `childSpecifier` (page 1414)

Declared In
`NSScriptObjectSpecifiers.h`

setContainerClassDescription:

Sets the class description of the receiver's container specifier to a given specifier.

```
- (void)setContainerClassDescription:(NSScriptClassDescription *)classDescription
```

Parameters
classDescription
The class description of the receiver's container specifier.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `containerClassDescription` (page 1415)

Declared In
`NSScriptObjectSpecifiers.h`

setContainerIsObjectBeingTested:

Sets whether the receiver's container should be an object involved in a filter reference or the top-level object.

```
- (void)setContainerIsObjectBeingTested:(BOOL)flag
```

Discussion
If the receiver's container specifier is `nil` and *`flag`* is `YES`, sets the receiver's container to be an object involved in a filter reference (for example, `whose color is blue`). If the receiver's container specifier is `nil` and *`flag`* is `NO`, sets the receiver's container to be the top-level object.

If *`flag`* is `YES` `setContainerIsRangeContainerObject:` (page 1422) should not also be invoked with an argument of `YES`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- containerIsObjectBeingTested` (page 1415)

Declared In
`NSScriptObjectSpecifiers.h`

setContainerIsRangeContainerObject:

Sets whether the receiver's container is to be the container for a range specifier or a top-level object.

```
- (void)setContainerIsRangeContainerObject:(BOOL)flag
```

Discussion
If the receiver's container specifier is `nil` and *`flag`* is `YES`, sets the receiver's container to be the container for a range specifier. If the receiver's container specifier is `nil` and *`flag`* is `NO`, sets the receiver's container to be the top-level object.

If *`flag`* is `YES`, `setContainerIsObjectBeingTested:` (page 1422) should not also be invoked with an argument of `YES`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- containerIsRangeContainerObject` (page 1416)

Declared In
`NSScriptObjectSpecifiers.h`

setContainerSpecifier:

Sets the container specifier of the receiver.

```
- (void)setContainerSpecifier:(NSScriptObjectSpecifier *)objSpecifier
```

Parameters

objSpecifier

The container specifier for the receiver.

Availability
Available in Mac OS X v10.0 and later.

See Also
- containerSpecifier (page 1416)

Declared In
NSScriptObjectSpecifiers.h

setEvaluationErrorNumber:

Sets the value of the evaluation error.

```
- (void)setEvaluationErrorNumber:(NSInteger)error
```

Parameters

error

The value for the evaluation error.

Availability
Available in Mac OS X v10.0 and later.

See Also
- evaluationErrorNumber (page 1417)

Declared In
NSScriptObjectSpecifiers.h

setKey:

Sets the key of the receiver.

```
- (void)setKey:(NSString *)key
```

Parameters

key

The key for the receiver.

Availability
Available in Mac OS X v10.0 and later.

See Also
- key (page 1419)
- keyClassDescription (page 1419)

Declared In
NSScriptObjectSpecifiers.h

Constants

NSScriptObjectSpecifier—Specifier Evaluation Errors

`NSScriptObjectSpecifier` provides the following constants for error codes for specific problems evaluating specifiers:

```
enum {
    NSNoSpecifierError = 0,
    NSNoTopLevelContainersSpecifierError,
    NSContainerSpecifierError,
    NSUnknownKeySpecifierError,
    NSInvalidIndexSpecifierError,
    NSInternalSpecifierError,
    NSOperationNotSupportedForKeySpecifierError
};
```

Constants

`NSNoSpecifierError`
No error encountered.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptObjectSpecifiers.h`.

`NSNoTopLevelContainersSpecifierError`
Someone called `evaluate` with `nil`.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptObjectSpecifiers.h`.

`NSContainerSpecifierError`
Error evaluating container specifier.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptObjectSpecifiers.h`.

`NSUnknownKeySpecifierError`
Receivers do not understand the key.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptObjectSpecifiers.h`.

`NSInvalidIndexSpecifierError`
Index out of bounds.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptObjectSpecifiers.h`.

`NSInternalSpecifierError`
Other internal error.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptObjectSpecifiers.h`.

`NSOperationNotSupportedForKeySpecifierError`
Attempt made to perform an unsupported operation on some key.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptObjectSpecifiers.h`.

Declared In
`NSScriptObjectSpecifier.h`

NSScriptSuiteRegistry Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptSuiteRegistry.h
Companion guide	Cocoa Scripting Guide
Related sample code	SimpleCarbonAppleScript

Overview

`NSScriptSuiteRegistry` functions as the top-level repository of scriptability information for an application at runtime.

Scriptability information specifies the terminology available for use in scripts that target an application. It also provides information, used by AppleScript and by Cocoa, about how support for that terminology is implemented in the application. This information includes descriptions of the scriptable object classes in an application and of the commands the application supports.

There are two standard formats for supplying scriptability information: the older script suite format, consisting of a script suite file and one or more script terminology files, and the newer scripting definition (or sdef) format, consisting of a single sdef file.

There is one instance of `NSScriptSuiteRegistry` per scriptable application. This registry object collects scriptability information when the application first needs to respond to an Apple event for which Cocoa hasn't installed a default event handler. It then creates one instance of `NSScriptClassDescription` for each object class and one instance of `NSScriptCommandDescription` for each command class, and installs a command handler for each command.

When a user executes an AppleScript script, Apple events are sent to the targeted application. Using the information stored in the registry object, Cocoa automatically converts incoming Apple events into script commands (based on `NSScriptCommand` or a subclass) that manipulate objects in the application.

The public methods of `NSScriptSuiteRegistry` are used primarily by Cocoa's built-in scripting support. You should not need to create a subclass of `NSScriptSuiteRegistry`.

For information on scriptability information formats, loading of scriptability information, and related topics, see "Scriptability Information" in Overview of Cocoa Support for Scriptable Applications in *Cocoa Scripting Guide*.

Tasks

Getting and Setting the Shared Instance

`+ setSharedScriptSuiteRegistry:` (page 1429)
Sets the single, shared instance of `NSScriptSuiteRegistry` to *registry*.

`+ sharedScriptSuiteRegistry` (page 1429)
Returns the single, shared instance of `NSScriptSuiteRegistry`, creating it first if it doesn't exist.

Getting Suite Information

`- suiteForAppleEventCode:` (page 1434)
Returns the name of the suite definition associated with the given four-character Apple event code, *code*.

`- suiteNames` (page 1434)
Returns the names of the suite definitions currently loaded by the application.

Getting and Registering Class Descriptions

`- classDescriptionsInSuite:` (page 1431)
Returns the class descriptions contained in the suite identified by *suiteName*.

`- classDescriptionWithAppleEventCode:` (page 1431)
Returns the class description associated with the given four-character Apple event code, *code*.

`- registerClassDescription:` (page 1433)
Registers class description *classDescription* for use by Cocoa's built-in scripting support by storing it in a per-suite internal dictionary under the class name.

Getting and Registering Command Descriptions

`- commandDescriptionsInSuite:` (page 1432)
Returns the command descriptions contained in the suite identified by *suiteName*.

`- commandDescriptionWithAppleEventClass:andAppleEventCode:` (page 1432)
Returns the command description identified by a suite's four-character Apple event code of the class (*eventClass*) and the four-character Apple event code of the command (*commandCode*).

`- registerCommandDescription:` (page 1434)
Registers command description *commandDesc* for use by Cocoa's built-in scripting support by storing it in a per-suite internal dictionary under the command name.

Getting Other Suite Information

- `aeteResource:` (page 1430)
  Returns an `NSData` object that contains data in `'aete'` resource format describing the scriptability information currently known to the application.
- `appleEventCodeForSuite:` (page 1430)
  Returns the Apple event code associated with the suite named *`suiteName`*, such as ‘`core`’ for the Core suite.
- `bundleForSuite:` (page 1431)
  Returns the bundle containing the suite-definition property list (extension `.scriptSuite`) identified by *`suiteName`*.

Loading Suites

- `loadSuiteWithDictionary:fromBundle:` (page 1433)
  Loads the suite definition encapsulated in *`dictionary`*; previously, this suite definition was parsed from a `.scriptSuite` property list contained in a framework or in *`bundle`*.
- `loadSuitesFromBundle:` (page 1432)
  Loads the suite definitions in bundle *`aBundle`*, invoking `loadSuiteWithDictionary:fromBundle:` (page 1433) for each suite found.

Class Methods

setSharedScriptSuiteRegistry:

Sets the single, shared instance of `NSScriptSuiteRegistry` to *`registry`*.

```
+ (void)setSharedScriptSuiteRegistry:(NSScriptSuiteRegistry *)registry
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptSuiteRegistry.h`

sharedScriptSuiteRegistry

Returns the single, shared instance of `NSScriptSuiteRegistry`, creating it first if it doesn't exist.

```
+ (NSScriptSuiteRegistry *)sharedScriptSuiteRegistry
```

Discussion
If it creates an instance, and if the application provides scriptability information in the script suite format, the method loads suite definitions in all frameworks and other bundles that the application currently imports or includes; if information is provided in the sdef format, the method loads information only from the specified sdef file. If in reading scriptability information an exception is `raised` because of parsing errors, it handles the exception by printing a line of information to the console.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `loadSuiteWithDictionary:fromBundle:` (page 1433)

Related Sample Code
SimpleCarbonAppleScript

Declared In
`NSScriptSuiteRegistry.h`

Instance Methods

aeteResource:

Returns an `NSData` object that contains data in `'aete'` resource format describing the scriptability information currently known to the application.

```
- (NSData *)aeteResource:(NSString *)languageName
```

Discussion
This method is typically invoked to implement the `get aete` Apple event for an application that provides scriptability information in the script suite format. The *`languageName`* argument is the name of a language for which a localized resource directory (such as `English.lproj`) exists. This language indication specifies the set of `.scriptTerminology` files to be used to generate the data. `NSScriptSuiteRegistry` does not create an `'aete'` resource unless this method is called.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `appleEventCodeForSuite:` (page 1430)

Declared In
`NSScriptSuiteRegistry.h`

appleEventCodeForSuite:

Returns the Apple event code associated with the suite named *`suiteName`*, such as `'core'` for the Core suite.

```
- (FourCharCode)appleEventCodeForSuite:(NSString *)suiteName
```

Availability
Available in Mac OS X v10.0 and later.

See Also
- `suiteForAppleEventCode:` (page 1434)

Declared In
NSScriptSuiteRegistry.h

bundleForSuite:

Returns the bundle containing the suite-definition property list (extension `.scriptSuite`) identified by *suiteName*.

```
- (NSBundle *)bundleForSuite:(NSString *)suiteName
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptSuiteRegistry.h

classDescriptionsInSuite:

Returns the class descriptions contained in the suite identified by *suiteName*.

```
- (NSDictionary *)classDescriptionsInSuite:(NSString *)suiteName
```

Discussion
Each class description (instance of `NSScriptClassDescription`) in the returned dictionary is identified by class name.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `classDescriptionWithAppleEventCode:` (page 1431)
- `registerClassDescription:` (page 1433)

Declared In
NSScriptSuiteRegistry.h

classDescriptionWithAppleEventCode:

Returns the class description associated with the given four-character Apple event code, *code*.

```
- (NSScriptClassDescription *)classDescriptionWithAppleEventCode:(FourCharCode)code
```

Discussion
Overriding behavior is important here. Multiple classes can have the same code if the classes have an uninterrupted linear inheritance from one another. For example, if class B is a subclass of A and class C is a subclass of B, and all three classes have the same four-character Apple event code, then this method returns the class description for class C.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `classDescriptionsInSuite:` (page 1431)
- `registerClassDescription:` (page 1433)

Declared In
`NSScriptSuiteRegistry.h`

commandDescriptionsInSuite:

Returns the command descriptions contained in the suite identified by *suiteName*.

```
- (NSDictionary *)commandDescriptionsInSuite:(NSString *)suiteName
```

Discussion
Each command description (instance of `NSScriptCommandDescription`) in the returned dictionary is identified by command name.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `commandDescriptionWithAppleEventClass:andAppleEventCode:` (page 1432)
- `registerCommandDescription:` (page 1434)

Declared In
`NSScriptSuiteRegistry.h`

commandDescriptionWithAppleEventClass:andAppleEventCode:

Returns the command description identified by a suite's four-character Apple event code of the class (*eventClass*) and the four-character Apple event code of the command (*commandCode*).

```
- (NSScriptCommandDescription
    *)commandDescriptionWithAppleEventClass:(FourCharCode)eventClass
    andAppleEventCode:(FourCharCode)commandCode
```

Availability
Available in Mac OS X v10.0 and later.

See Also
- `commandDescriptionsInSuite:` (page 1432)
- `registerCommandDescription:` (page 1434)

Declared In
`NSScriptSuiteRegistry.h`

loadSuitesFromBundle:

Loads the suite definitions in bundle *aBundle*, invoking `loadSuiteWithDictionary:fromBundle:` (page 1433) for each suite found.

```
- (void)loadSuitesFromBundle:(NSBundle *)aBundle
```

Discussion
If errors occur while method is parsing a suite-definition file, the method logs error messages to the console.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptSuiteRegistry.h

loadSuiteWithDictionary:fromBundle:

Loads the suite definition encapsulated in *dictionary*; previously, this suite definition was parsed from a .scriptSuite property list contained in a framework or in *bundle*.

```
- (void)loadSuiteWithDictionary:(NSDictionary *)dictionary fromBundle:(NSBundle
    *)bundle
```

Discussion
The method extracts information from the dictionary and caches it in various internal collection objects. If keys are missing or values are of the wrong type, it logs messages to the console. It also registers class descriptions and command descriptions. In registering a class description, it invokes the NSClassDescription class method registerClassDescription:forClass: (page 259). In registering a command description, it arranges for the Apple event translator to handle incoming Apple events that represent the defined commands.

This method is invoked when the shared instance is initialized and when bundles are loaded at runtime. Prior to invoking it, NSScriptSuiteRegistry creates the dictionary argument from the .scriptSuite property list. If you invoke this method in your code, you should try to do it before the application receives its first Apple event.

Availability
Available in Mac OS X v10.0 and later.

See Also
- loadSuitesFromBundle: (page 1432)
- registerClassDescription: (page 1433)
- registerCommandDescription: (page 1434)
+ sharedScriptSuiteRegistry (page 1429)

Declared In
NSScriptSuiteRegistry.h

registerClassDescription:

Registers class description *classDescription* for use by Cocoa's built-in scripting support by storing it in a per-suite internal dictionary under the class name.

```
- (void)registerClassDescription:(NSScriptClassDescription *)classDescription
```

Availability
Available in Mac OS X v10.0 and later.

See Also
- `loadSuiteWithDictionary:fromBundle:` (page 1433)
- `registerCommandDescription:` (page 1434)

Declared In
`NSScriptSuiteRegistry.h`

registerCommandDescription:

Registers command description *commandDesc* for use by Cocoa's built-in scripting support by storing it in a per-suite internal dictionary under the command name.

```
- (void)registerCommandDescription:(NSScriptCommandDescription *)commandDesc
```

Discussion
Also registers with the single, shared instance of `NSAppleEventManager` to handle incoming Apple events that should be handled by the command.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `loadSuiteWithDictionary:fromBundle:` (page 1433)
- `registerClassDescription:` (page 1433)

Declared In
`NSScriptSuiteRegistry.h`

suiteForAppleEventCode:

Returns the name of the suite definition associated with the given four-character Apple event code, *code*.

```
- (NSString *)suiteForAppleEventCode:(FourCharCode)code
```

Availability
Available in Mac OS X v10.0 and later.

See Also
- `suiteNames` (page 1434)

Declared In
`NSScriptSuiteRegistry.h`

suiteNames

Returns the names of the suite definitions currently loaded by the application.

```
- (NSArray *)suiteNames
```

Availability
Available in Mac OS X v10.0 and later.

See Also
- `suiteForAppleEventCode:` (page 1434)

Declared In
`NSScriptSuiteRegistry.h`

NSScriptWhoseTest Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptWhoseTests.h
Companion guide	Cocoa Scripting Guide

Overview

`NSScriptWhoseTest` is an abstract class whose sole method is `isTrue` (page 1438). Two concrete subclasses of `NSScriptWhoseTest` generate objects representing Boolean expressions comparing one object with another and objects representing multiple Boolean expressions connected by logical operators (`OR`, `AND`, `NOT`). These classes are, respectively, `NSSpecifierTest` and `NSLogicalTest`. In evaluating itself, an `NSWhoseSpecifier` invokes the `isTrue` (page 1438) method of its "test" object.

You shouldn't need to subclass `NSScriptWhoseTest`, and you should rarely need to subclass one of its subclasses.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 2034)
- `initWithCoder:` (page 2034)

Tasks

Evaluating a Test

- `isTrue` (page 1438)
  Returns a Boolean value that indicates whether the test represented by the receiver evaluates to `YES`.

Instance Methods

isTrue

Returns a Boolean value that indicates whether the test represented by the receiver evaluates to `YES`.

```
- (BOOL)isTrue
```

Return Value
`YES` if the test represented by the receiver evaluates to `YES`, otherwise `NO`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptWhoseTests.h`

NSSerializer Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSSerialization.h
Availability	Deprecated in Mac OS X v10.2.
Companion guide	Archives and Serializations Programming Guide for Cocoa

Overview

NSSerializer is obsolete and has been deprecated. Instead use `NSPropertyListSerialization`.

The `NSSerializer` class provides a mechanism for creating an abstract representation of a property list. (In Cocoa, property lists are defined to be—and to contain—objects of these classes: `NSDictionary`, `NSArray`, `NSString`, `NSData`.) The `NSSerializer` class stores this representation in an NSData object using an architecture-independent format, so that property lists can be used with distributed applications. NSSerializer's companion class `NSDeserializer` declares methods that take the representation and recreate the property list in memory.

The `NSSerializer` class object provides the interface to the serialization process; you don't create instances of `NSSerializer`. You might subclass `NSSerializer` to modify the representation it creates, for example, to encrypt the data or add authentication information.

Other types of data besides property lists can be serialized using `serializeDataAt:ofObjCType:context:` and `deserializeDataAt:ofObjCType:atCursor:context:`, declared by `NSData` and `NSMutableData`, allowing these types to be represented in an architecture-independent format. Furthermore, the `NSObjCTypeSerializationCallBack` protocol allows you to serialize and deserialize objects that are not property lists.

Tasks

Serializing a Property List

`+ serializePropertyList:` (page 1440) Deprecated in Mac OS X v10.2

Creates a data object, serializes *aPropertyList* into it, and returns the data object.

`+ serializePropertyList:intoData:` (page 1440) Deprecated in Mac OS X v10.2
Serializes the property list *aPropertyList* into the mutable data object *mdata*.

Class Methods

serializePropertyList:

Creates a data object, serializes *aPropertyList* into it, and returns the data object. (Deprecated in Mac OS X v10.2.)

```
+ (NSData *)serializePropertyList:(id)aPropertyList
```

Discussion
aPropertyList must be a kind of `NSData`, `NSString`, `NSArray`, or `NSDictionary` object.

Availability
Deprecated in Mac OS X v10.2.

Declared In
`NSSerialization.h`

serializePropertyList:intoData:

Serializes the property list *aPropertyList* into the mutable data object *mdata*. (Deprecated in Mac OS X v10.2.)

```
+ (void)serializePropertyList:(id)aPropertyList intoData:(NSMutableData *)mdata
```

Discussion
aPropertyList must be a kind of `NSData`, `NSString`, `NSArray`, or `NSDictionary` object. The property list is appended to *mdata*.

Availability
Deprecated in Mac OS X v10.2.

Declared In
`NSSerialization.h`

NSSet Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSMutableCopying NSFastEnumeration NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSSet.h
Companion guide	Collections Programming Topics for Cocoa
Related sample code	Core Data HTML Store CoreRecipes CustomAtomicStoreSubclass NewsReader Sketch-112

Overview

The `NSSet`, `NSMutableSet`, and `NSCountedSet` classes declare the programmatic interface to an object that manages a set of objects. `NSSet` provides support for the mathematical concept of a set. A set, both in its mathematical sense and in the implementation of `NSSet`, is an unordered collection of distinct elements. The `NSMutableSet` (a subclass of `NSSet`) and `NSCountedSet` (a subclass of `NSMutableSet`) classes are provided for sets whose contents may be altered.

`NSSet` and `NSMutableSet` are part of a class cluster, so sets are not actual instances of `NSSet` or `NSMutableSet`. Rather, the instances belong to one of their private subclasses. (For convenience, we use the term set to refer to any one of these instances without specifying its exact class membership.) Although a set's class is private, its interface is public, as declared by the abstract superclasses `NSSet` and `NSMutableSet`. Note that `NSCountedSet` is not part of the class cluster; it is a concrete subclass of `NSMutableSet`.

`NSSet` declares the programmatic interface for static sets of objects. You establish a static set's entries when it's created, and thereafter the entries can't be modified. `NSMutableSet`, on the other hand, declares a programmatic interface for dynamic sets of objects. A dynamic—or mutable—set allows the addition and deletion of entries at any time, automatically allocating memory as needed.

You can use sets as an alternative to arrays when the order of elements isn't important and performance in testing whether an object is contained in the set is a consideration—while arrays are ordered, testing for membership is slower than with sets.

Objects in a set must respond to the `NSObject` protocol methods `hash` (page 2101) and `isEqual:` (page 2101)—see the `NSObject` protocol for more information.

Note that if mutable objects are stored in a set, either the `hash` method of the objects shouldn't depend on the internal state of the mutable objects or the mutable objects shouldn't be modified while they're in the set (note that it can be difficult to know whether or not a given object is in a collection).

Objects added to a set are not copied; rather, an object receives a `retain` message before it's added to a set.

Typically, you create a temporary set by sending one of the `set...` methods to the `NSSet` class object. These methods return an `NSSet` object containing the elements (if any) you pass in as arguments. The `set` (page 1445) method is a "convenience" method to create an empty mutable set.

The set classes adopt the `NSCopying` and `NSMutableCopying` protocols, making it convenient to convert a set of one type to the other.

`NSSet` provides methods for querying the elements of the set. `allObjects` (page 1449) returns an array containing the objects in a set. `anyObject` (page 1450) returns some object in the set. `count` (page 1451) returns the number of objects currently in the set. `member:` (page 1458) returns the object in the set that is equal to a specified object. Additionally, `intersectsSet:` (page 1455) tests for set intersection, `isEqualToSet:` (page 1456) tests for set equality, and `isSubsetOfSet:` (page 1456) tests for one set being a subset of another.

The `objectEnumerator` (page 1458) method provides for traversing elements of the set one by one. For better performance on Mac OS X v10.5 and later, you can also use the Objective-C fast enumeration feature (see Fast Enumeration).

`NSSet`'s `makeObjectsPerformSelector:` (page 1457) and `makeObjectsPerformSelector:withObject:` (page 1457) methods provides for sending messages to individual objects in the set.

`NSSet` is "toll-free bridged" with its Core Foundation counterpart, *CFSet Reference*. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSSet *` parameter, you can pass a `CFSetRef`, and in a function where you see a `CFSetRef` parameter, you can pass an `NSSet` instance (you cast one type to the other to suppress compiler warnings). See Interchangeable Data Types for more information on toll-free bridging.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 2034)
- `initWithCoder:` (page 2034)

NSCopying
- `copyWithZone:` (page 2042)

NSMutableCopying
- `mutableCopyWithZone:` (page 2094)

Tasks

Creating a Set

`+ set` (page 1445)
Creates and returns an empty set.

`+ setWithArray:` (page 1445)
Creates and returns a set containing a uniqued collection of those objects contained in a given array.

`+ setWithObject:` (page 1446)
Creates and returns a set that contains a single given object.

`+ setWithObjects:` (page 1447)
Creates and returns a set containing the objects in a given argument list.

`+ setWithObjects:count:` (page 1447)
Creates and returns a set containing a specified number of objects from a given C array of objects.

`+ setWithSet:` (page 1448)
Creates and returns a set containing the objects from another set.

`- setByAddingObject:` (page 1459)
Returns a new set formed by adding a given object to the collection defined by the receiver.

`- setByAddingObjectsFromSet:` (page 1461)
Returns a new set formed by adding the objects in a given set to the collection defined by the receiver.

`- setByAddingObjectsFromArray:` (page 1460)
Returns a new set formed by adding the objects in a given array to the collection defined by the receiver.

Initializing a Set

`- initWithArray:` (page 1452)
Initializes a newly allocated set with the objects that are contained in a given array.

`- initWithObjects:` (page 1453)
Initializes a newly allocated set with members taken from the specified list of objects.

`- initWithObjects:count:` (page 1454)
Initializes a newly allocated set with a specified number of objects from a given C array of objects.

`- initWithSet:` (page 1454)
Initializes a newly allocated set and adds to it objects from another given set.

`- initWithSet:copyItems:` (page 1455)
Initializes a newly allocated set and adds to it members of another given set.

Counting Entries

`- count` (page 1451)
Returns the number of members in the receiver.

Accessing Set Members

- `allObjects` (page 1449)
  Returns an array containing the receiver's members, or an empty array if the receiver has no members.
- `anyObject` (page 1450)
  Returns one of the objects in the receiver, or `nil` if the receiver contains no objects.
- `containsObject:` (page 1450)
  Returns a Boolean value that indicates whether a given object is present in the receiver.
- `filteredSetUsingPredicate:` (page 1452)
  Evaluates a given predicate against each object in the receiver and returns a new set containing the objects for which the predicate returns true.
- `makeObjectsPerformSelector:` (page 1457)
  Sends to each object in the receiver a message specified by a given selector.
- `makeObjectsPerformSelector:withObject:` (page 1457)
  Sends to each object in the receiver a message specified by a given selector.
- `member:` (page 1458)
  Determines whether the receiver contains an object equal to a given object, and returns that object if it is present.
- `objectEnumerator` (page 1458)
  Returns an enumerator object that lets you access each object in the receiver.

Comparing Sets

- `isSubsetOfSet:` (page 1456)
  Returns a Boolean value that indicates whether every object in the receiver is also present in another given set.
- `intersectsSet:` (page 1455)
  Returns a Boolean value that indicates whether at least one object in the receiver is also present in another given set.
- `isEqualToSet:` (page 1456)
  Compares the receiver to another set.
- `valueForKey:` (page 1461)
  Return a set containing the results of invoking `valueForKey:` on each of the receiver's members.
- `setValue:forKey:` (page 1461)
  Invokes `setValue:forKey:` on each of the receiver's members.

Key-Value Observing

- `addObserver:forKeyPath:options:context:` (page 1449)
  Raises an exception.
- `removeObserver:forKeyPath:` (page 1459)
  Raises an exception.

Describing a Set

- `description` (page 1451)
  Returns a string that represents the contents of the receiver, formatted as a property list.
- `descriptionWithLocale:` (page 1451)
  Returns a string that represents the contents of the receiver, formatted as a property list.

Class Methods

set

Creates and returns an empty set.

```
+ (id)set
```

Return Value
A new empty set.

Discussion
This method is declared primarily for the use of mutable subclasses of `NSSet`.

Availability
Available in Mac OS X v10.0 and later.

See Also
+ `setWithArray:` (page 1445)
+ `setWithObject:` (page 1446)
+ `setWithObjects:` (page 1447)
- `setByAddingObject:` (page 1459)
- `setByAddingObjectsFromSet:` (page 1461)
- `setByAddingObjectsFromArray:` (page 1460)

Declared In
`NSSet.h`

setWithArray:

Creates and returns a set containing a uniqued collection of those objects contained in a given array.

```
+ (id)setWithArray:(NSArray *)anArray
```

Parameters
anArray
An array containing the objects to add to the new set. If the same object appears more than once in *anArray*, it is added only once to the returned set. Each object receives a `retain` (page 2108) message as it is added to the set.

Return Value
A new set containing a uniqued collection of those objects contained in *anArray*.

Availability
Available in Mac OS X v10.0 and later.

See Also
+ set (page 1445)
+ setWithObject: (page 1446)
+ setWithObjects: (page 1447)
- setByAddingObject: (page 1459)
- setByAddingObjectsFromSet: (page 1461)
- setByAddingObjectsFromArray: (page 1460)

Related Sample Code
CoreRecipes
NewsReader

Declared In
NSSet.h

setWithObject:

Creates and returns a set that contains a single given object.

```
+ (id)setWithObject:(id)anObject
```

Parameters

anObject

The object to add to the new set. *anObject* receives a retain (page 2108) message after being added to the set.

Return Value
A new set that contains a single member, *anObject*.

Availability
Available in Mac OS X v10.0 and later.

See Also
+ set (page 1445)
+ setWithArray: (page 1445)
+ setWithObjects: (page 1447)
- setByAddingObject: (page 1459)
- setByAddingObjectsFromSet: (page 1461)
- setByAddingObjectsFromArray: (page 1460)

Related Sample Code
Core Data HTML Store

Declared In
NSSet.h

setWithObjects:

Creates and returns a set containing the objects in a given argument list.

```
+ (id)setWithObjects:(id)anObject,  ...
```

Parameters

anObject

The first object to add to the new set.

anObject, ...

A comma-separated list of objects, ending with `nil`, to add to the new set. If the same object appears more than once in the list of objects, it is added only once to the returned set. Each object receives a `retain` (page 2108) message as it is added to the set.

Return Value

A new set containing the objects in the argument list.

Discussion

As an example, the following code excerpt creates a set containing three different types of elements (assuming `aPath` exits):

```
NSSet *mySet;
NSData *someData = [NSData dataWithContentsOfFile:aPath];
NSValue *aValue = [NSNumber numberWithInteger:5];
NSString *aString = @"a string";

mySet = [NSSet setWithObjects:someData, aValue, aString, nil];
```

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ set` (page 1445)
`+ setWithArray:` (page 1445)
`+ setWithObject:` (page 1446)
`- setByAddingObject:` (page 1459)
`- setByAddingObjectsFromSet:` (page 1461)
`- setByAddingObjectsFromArray:` (page 1460)

Declared In

`NSSet.h`

setWithObjects:count:

Creates and returns a set containing a specified number of objects from a given C array of objects.

```
+ (id)setWithObjects:(id *)objects  count:(NSUInteger)count
```

Parameters

objects

A C array of objects to add to the new set. If the same object appears more than once in *objects*, it is added only once to the returned set. Each object receives a `retain` (page 2108) message as it is added to the set.

count

The number of objects from *objects* to add to the new set.

Return Value
A new set containing *count* objects from the list of objects specified by *objects*.

Availability
Available in Mac OS X v10.0 and later.

See Also
+ set (page 1445)
+ setWithArray: (page 1445)
+ setWithObject: (page 1446)
+ setWithObjects: (page 1447)
- setByAddingObject: (page 1459)
- setByAddingObjectsFromSet: (page 1461)
- setByAddingObjectsFromArray: (page 1460)

Declared In
NSSet.h

setWithSet:

Creates and returns a set containing the objects from another set.

```
+ (id)setWithSet:(NSSet *)aSet
```

Parameters
aSet

A set containing the objects to add to the new set. Each object receives a retain (page 2108) message as it is added to the new set.

Return Value
A new set containing the objects from *aSet*.

Availability
Available in Mac OS X v10.0 and later.

See Also
+ set (page 1445)
+ setWithArray: (page 1445)
+ setWithObject: (page 1446)
+ setWithObjects: (page 1447)
- setByAddingObject: (page 1459)
- setByAddingObjectsFromSet: (page 1461)
- setByAddingObjectsFromArray: (page 1460)

Declared In
NSSet.h

Instance Methods

addObserver:forKeyPath:options:context:

Raises an exception.

```
- (void)addObserver:(NSObject *)observer forKeyPath:(NSString *)keyPath
    options:(NSKeyValueObservingOptions)options context:(void *)context
```

Parameters

observer

The object to register for KVO notifications. The observer must implement the key-value observing method `observeValueForKeyPath:ofObject:change:context:` (page 2081).

keyPath

The key path, relative to the receiver, of the property to observe. This value must not be `nil`.

options

A combination of the `NSKeyValueObservingOptions` (page 2086) values that specifies what is included in observation notifications. For possible values, see `NSKeyValueObservingOptions`.

context

Arbitrary data that is passed to *observer* in `observeValueForKeyPath:ofObject:change:context:` (page 2081).

Special Considerations

`NSSet` objects are not observable, so this method raises an exception when invoked on an `NSSet` object. Instead of observing a set, observe the unordered to-many relationship for which the set is the collection of related objects.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- removeObserver:forKeyPath:` (page 1459)

Declared In

`NSKeyValueObserving.h`

allObjects

Returns an array containing the receiver's members, or an empty array if the receiver has no members.

```
- (NSArray *)allObjects
```

Return Value

An array containing the receiver's members, or an empty array if the receiver has no members. The order of the objects in the array isn't defined.

Availability

Available in Mac OS X v10.0 and later.

See Also
- `anyObject` (page 1450)
- `objectEnumerator` (page 1458)

Related Sample Code
CoreRecipes
Sketch-112

Declared In
`NSSet.h`

anyObject

Returns one of the objects in the receiver, or `nil` if the receiver contains no objects.

```
- (id)anyObject
```

Return Value
One of the objects in the receiver, or `nil` if the receiver contains no objects. The object returned is chosen at the receiver's convenience—the selection is not guaranteed to be random.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `allObjects` (page 1449)
- `objectEnumerator` (page 1458)

Related Sample Code
Core Data HTML Store

Declared In
`NSSet.h`

containsObject:

Returns a Boolean value that indicates whether a given object is present in the receiver.

```
- (BOOL)containsObject:(id)anObject
```

Parameters
anObject
The object for which to test membership of the receiver.

Return Value
`YES` if *anObject* is present in the receiver, otherwise `NO`.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `member:` (page 1458)

Declared In
NSSet.h

count

Returns the number of members in the receiver.

```
- (NSUInteger)count
```

Return Value
The number of members in the receiver.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CoreRecipes

Declared In
NSSet.h

description

Returns a string that represents the contents of the receiver, formatted as a property list.

```
- (NSString *)description
```

Return Value
A string that represents the contents of the receiver, formatted as a property list.

Availability
Available in Mac OS X v10.0 and later.

See Also
- descriptionWithLocale: (page 1451)

Declared In
NSSet.h

descriptionWithLocale:

Returns a string that represents the contents of the receiver, formatted as a property list.

```
- (NSString *)descriptionWithLocale:(id)locale
```

Parameters
locale
In Mac OS X v10.4 and earlier, this must be a dictionary that specifies options used for formatting each of the receiver's members. In Mac OS X v10.5 and later, you can use an NSLocale object. If you do not want the receiver's members to be formatted, specify nil.

Return Value
A string that represents the contents of the receiver, formatted as a property list.

Discussion
This method sends each of the receiver's members `descriptionWithLocale:` with *`locale`* passed as the sole parameter. If the receiver's members do not respond to `descriptionWithLocale:`, this method sends `description` (page 2100) instead.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- description` (page 1451)

Declared In
`NSSet.h`

filteredSetUsingPredicate:

Evaluates a given predicate against each object in the receiver and returns a new set containing the objects for which the predicate returns true.

```
- (NSSet *)filteredSetUsingPredicate:(NSPredicate *)predicate
```

Parameters
`predicate`
A predicate.

Return Value
A new set containing the objects in the receiver for which *`predicate`* returns true.

Discussion
The following example illustrates the use of this method.

```
NSSet *sourceSet =
    [NSSet setWithObjects:@"One", @"Two", @"Three", @"Four", nil];
NSPredicate *predicate =
    [NSPredicate predicateWithFormat:@"SELF beginswith 'T'"];
NSSet *filteredSet =
    [sourceSet filteredSetUsingPredicate:predicate];
// filteredSet contains (Two, Three)
```

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSPredicate.h`

initWithArray:

Initializes a newly allocated set with the objects that are contained in a given array.

```
- (id)initWithArray:(NSArray *)array
```

Parameters

array

An array of objects to add to the new set. If the same object appears more than once in *array*, it is represented only once in the returned set. Each object receives a `retain` (page 2108) message as it is added to the set.

Return Value

An initialized object, which might be different than the original receiver.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `initWithObjects:` (page 1453)
- `initWithObjects:count:` (page 1454)
- `initWithSet:` (page 1454)
- `initWithSet:copyItems:` (page 1455)
+ `setWithArray:` (page 1445)

Declared In

`NSSet.h`

initWithObjects:

Initializes a newly allocated set with members taken from the specified list of objects.

```
- (id)initWithObjects:(id)firstObj, ...
```

Parameters

anObject

The first object to add to the new set.

firstObj, ...

A comma-separated list of objects, ending with `nil`, to add to the new set. If the same object appears more than once in the list, it is represented only once in the returned set. Each object receives a `retain` (page 2108) message as it is added to the set

Return Value

An initialized object, which might be different than the original receiver.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `initWithArray:` (page 1452)
- `initWithObjects:count:` (page 1454)
- `initWithSet:` (page 1454)
- `initWithSet:copyItems:` (page 1455)
+ `setWithObjects:` (page 1447)

Declared In

`NSSet.h`

initWithObjects:count:

Initializes a newly allocated set with a specified number of objects from a given C array of objects.

```
- (id)initWithObjects:(id *)objects  count:(NSUInteger)count
```

Parameters

objects

A C array of objects to add to the new set. If the same object appears more than once in *objects*, it is added only once to the returned set. Each object receives a `retain` (page 2108) message as it is added to the set.

count

The number of objects from *objects* to add to the new set.

Return Value
An initialized object, which might be different than the original receiver.

Discussion
This method is the designated initializer for `NSSet`.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `initWithArray:` (page 1452)
- `initWithObjects:` (page 1453)
- `initWithSet:` (page 1454)
- `initWithSet:copyItems:` (page 1455)
+ `setWithObjects:count:` (page 1447)

Declared In
`NSSet.h`

initWithSet:

Initializes a newly allocated set and adds to it objects from another given set.

```
- (id)initWithSet:(NSSet *)otherSet
```

Parameters

otherSet

A set containing objects to add to the receiver. Each object is retained as it is added to the receiver.

Return Value
An initialized object, which might be different than the original receiver.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `initWithArray:` (page 1452)
- `initWithObjects:` (page 1453)
- `initWithObjects:count:` (page 1454)

- initWithSet:copyItems: (page 1455)
+ setWithSet: (page 1448)

Declared In
NSSet.h

initWithSet:copyItems:

Initializes a newly allocated set and adds to it members of another given set.

```
- (id)initWithSet:(NSSet *)otherSet copyItems:(BOOL)flag
```

Parameters

otherSet
A set containing objects to add to the new set.

flag
If YES, the members of *otherSet* are copied, and the copies are added to the receiver. If NO, the members of *otherSet* are added to the receiver and retained.

Return Value
An initialized object that contains the members of *otherSet*.

This method returns an initialized object, which might be different than the original receiver.

Discussion
Note that, if *flag* is YES, copyWithZone: (page 2042) is invoked to make copies—thus, the receiver's new member objects may be immutable, even though their counterparts in *otherSet* were mutable. Also, members must conform to the NSCopying protocol)

Availability
Available in Mac OS X v10.0 and later.

See Also
- initWithArray: (page 1452)
- initWithObjects: (page 1453)
- initWithObjects:count: (page 1454)
- initWithSet: (page 1454)
+ setWithSet: (page 1448)

Declared In
NSSet.h

intersectsSet:

Returns a Boolean value that indicates whether at least one object in the receiver is also present in another given set.

```
- (BOOL)intersectsSet:(NSSet *)otherSet
```

Parameters

otherSet

The set with which to compare the receiver.

Return Value

YES if at least one object in the receiver is also present in *otherSet*, otherwise NO.

Availability

Available in Mac OS X v10.0 and later.

See Also

- isEqualToSet: (page 1456)
- isSubsetOfSet: (page 1456)

Declared In

NSSet.h

isEqualToSet:

Compares the receiver to another set.

```
- (BOOL)isEqualToSet:(NSSet *)otherSet
```

Parameters

otherSet

The set with which to compare the receiver.

Return Value

YES if the contents of *otherSet* are equal to the contents of the receiver, otherwise NO.

Discussion

Two sets have equal contents if they each have the same number of members and if each member of one set is present in the other.

Availability

Available in Mac OS X v10.0 and later.

See Also

- intersectsSet: (page 1455)
- isEqual: (page 2101) (NSObject protocol)
- isSubsetOfSet: (page 1456)

Declared In

NSSet.h

isSubsetOfSet:

Returns a Boolean value that indicates whether every object in the receiver is also present in another given set.

```
- (BOOL)isSubsetOfSet:(NSSet *)otherSet
```

Parameters

otherSet

The set with which to compare the receiver.

Return Value

YES if every object in the receiver is also present in *otherSet*, otherwise NO.

Availability

Available in Mac OS X v10.0 and later.

See Also

- intersectsSet: (page 1455)
- isEqualToSet: (page 1456)

Declared In

NSSet.h

makeObjectsPerformSelector:

Sends to each object in the receiver a message specified by a given selector.

```
- (void)makeObjectsPerformSelector:(SEL)aSelector
```

Parameters

aSelector

A selector that specifies the message to send to the members of the receiver. The method must not take any arguments. It should not have the side effect of modifying the receiver. This value must not be NULL.

Discussion

The message specified by *aSelector* is sent once to each member of the receiver. This method raises an NSInvalidArgumentException if *aSelector* is NULL.

Availability

Available in Mac OS X v10.0 and later.

See Also

- makeObjectsPerformSelector:withObject: (page 1457)

Declared In

NSSet.h

makeObjectsPerformSelector:withObject:

Sends to each object in the receiver a message specified by a given selector.

```
- (void)makeObjectsPerformSelector:(SEL)aSelector withObject:(id)anObject
```

Parameters

aSelector

A selector that specifies the message to send to the receiver's members. The method must take a single argument of type id. The method should not, as a side effect, modify the receiver. The value must not be NULL.

anObject
The object to pass as an argument to the method specified by *aSelector*.

Discussion
The message specified by *aSelector* is sent, with *anObject* as the argument, once to each member of the receiver. This method raises an `NSInvalidArgumentException` if *aSelector* is `NULL`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- makeObjectsPerformSelector:` (page 1457)

Declared In
`NSSet.h`

member:

Determines whether the receiver contains an object equal to a given object, and returns that object if it is present.

```
- (id)member:(id)anObject
```

Parameters
anObject
The object for which to test for membership of the receiver.

Return Value
If the receiver contains an object equal to *anObject* (as determined by `isEqual:` (page 2101)) then that object (typically this will be *anObject*), otherwise `nil`.

Discussion
If you override `isEqual:`, you must also override the `hash` method for the `member:` method to work on a set of objects of your class.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSSet.h`

objectEnumerator

Returns an enumerator object that lets you access each object in the receiver.

```
- (NSEnumerator *)objectEnumerator
```

Return Value
An enumerator object that lets you access each object in the receiver.

Discussion
The following code fragment illustrates how you can use this method.

```
NSEnumerator *enumerator = [mySet objectEnumerator];
```

```
id value;

while ((value = [enumerator nextObject])) {
    /* code that acts on the set's values */
}
```

When this method is used with mutable subclasses of `NSSet`, your code shouldn't modify the receiver during enumeration. If you intend to modify the receiver, use the `allObjects` (page 1449) method to create a "snapshot" of the set's members. Enumerate the snapshot, but make your modifications to the original set.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- nextObject` (page 558) (`NSEnumerator`)

Related Sample Code
CoreRecipes

Declared In
`NSSet.h`

removeObserver:forKeyPath:

Raises an exception.

```
- (void)removeObserver:(NSObject *)observer forKeyPath:(NSString *)keyPath
```

Parameters

observer
The object to remove as an observer.

keyPath
A key-path, relative to the receiver, for which *observer* is registered to receive KVO change notifications. This value must not be `nil`.

Special Considerations

`NSSet` objects are not observable, so this method raises an exception when invoked on an `NSSet` object. Instead of observing a set, observe the unordered to-many relationship for which the set is the collection of related objects.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- addObserver:forKeyPath:options:context:` (page 1449)

Declared In
`NSKeyValueObserving.h`

setByAddingObject:

Returns a new set formed by adding a given object to the collection defined by the receiver.

```
- (NSSet *)setByAddingObject:(id)anObject
```

Parameters

anObject

The object to add to the collection defined by the receiver.

Return Value

A new set formed by adding *anObject* to the collection defined by the receiver.

Availability

Available in Mac OS X v10.5 and later.

See Also

+ set (page 1445)
+ setWithArray: (page 1445)
+ setWithObject: (page 1446)
+ setWithObjects: (page 1447)
- setByAddingObjectsFromSet: (page 1461)
- setByAddingObjectsFromArray: (page 1460)

Declared In

NSSet.h

setByAddingObjectsFromArray:

Returns a new set formed by adding the objects in a given array to the collection defined by the receiver.

```
- (NSSet *)setByAddingObjectsFromArray:(NSArray *)other
```

Parameters

other

The array of objects to add to the collection defined by the receiver.

Return Value

A new set formed by adding the objects in *other* to the collection defined by the receiver.

Availability

Available in Mac OS X v10.5 and later.

See Also

+ set (page 1445)
+ setWithArray: (page 1445)
+ setWithObject: (page 1446)
+ setWithObjects: (page 1447)
- setByAddingObject: (page 1459)
- setByAddingObjectsFromSet: (page 1461)

Declared In

NSSet.h

setByAddingObjectsFromSet:

Returns a new set formed by adding the objects in a given set to the collection defined by the receiver.

```
- (NSSet *)setByAddingObjectsFromSet:(NSSet *)other
```

Parameters

other

The set of objects to add to the collection defined by the receiver.

Return Value

A new set formed by adding the objects in *other* to the collection defined by the receiver.

Availability

Available in Mac OS X v10.5 and later.

See Also

+ set (page 1445)
+ setWithArray: (page 1445)
+ setWithObject: (page 1446)
+ setWithObjects: (page 1447)
- setByAddingObject: (page 1459)
- setByAddingObjectsFromSet: (page 1461)

Declared In

NSSet.h

setValue:forKey:

Invokes setValue:forKey: on each of the receiver's members.

```
- (void)setValue:(id)value forKey:(NSString *)key
```

Parameters

value

The value for the property identified by *key*.

key

The name of one of the properties of the receiver's members.

Availability

Available in Mac OS X v10.4 and later.

See Also

- valueForKey: (page 1461)

Declared In

NSKeyValueCoding.h

valueForKey:

Return a set containing the results of invoking valueForKey: on each of the receiver's members.

```
- (id)valueForKey:(NSString *)key
```

Parameters

key

The name of one of the properties of the receiver's members.

Return Value

A set containing the results of invoking `valueForKey:` (with the argument *key*) on each of the receiver's members.

Discussion

The returned set might not have the same number of members as the receiver. The returned set will not contain any elements corresponding to instances of `valueForKey:` returning `nil` (note that this is in contrast with `NSArray`'s implementation, which may put `NSNull` values in the arrays it returns).

Availability

Available in Mac OS X v10.4 and later.

See Also

`- setValue:forKey:` (page 1461)

Declared In

`NSKeyValueCoding.h`

NSSetCommand Class Reference

Inherits from	NSScriptCommand : NSObject
Conforms to	NSCoding (NSScriptCommand) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptStandardSuiteCommands.h
Companion guide	Cocoa Scripting Guide

Overview

An instance of `NSSetCommand` sets one or more attributes or relationships to one or more values; for example, it may set the (x, y) coordinates for a window's position or set the name of a document.

`NSSetCommand` is part of Cocoa's built-in scripting support. It works automatically to support the `set` command through key-value coding. Most applications don't need to subclass `NSSetCommand` or call its methods.

`NSSetCommand` uses available scripting class descriptions to determine whether it should set a value for an attribute (or property), or set a value for all elements (to-many objects). For the latter, it invokes `replaceValueAtIndex:inPropertyWithKey:withValue:` (page 2119); for the former, it invokes `setValue:forKey:` (page 2064) (or, if the receiver overrides `takeValue:forKey:` (page 2068), it invokes that method, to support backward binary compatibility.)

For information on working with `set` commands, see Getting and Setting Properties and Elements in *Cocoa Scripting Guide*.

Tasks

Working with Specifiers

- `keySpecifier` (page 1464)
  Returns a specifier that identifies the attribute or relationship that is to be set for the receiver of the `set` AppleScript command.

- setReceiversSpecifier: (page 1464)
  Sets the receiver's object specifier.

Instance Methods

keySpecifier

Returns a specifier that identifies the attribute or relationship that is to be set for the receiver of the set AppleScript command.

```
- (NSScriptObjectSpecifier *)keySpecifier
```

Return Value
A specifier that identifies the attribute or relationship that is to be set for the receiver of the set AppleScript command.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptStandardSuiteCommands.h

setReceiversSpecifier:

Sets the receiver's object specifier.

```
- (void)setReceiversSpecifier:(NSScriptObjectSpecifier *)receiversRef
```

Parameters
receiversRef
  The receiver's object specifier.

Discussion
When the command is executed, it sets attributes or relationships in the objects specified by *receiversRef*.

This method overrides setReceiversSpecifier: (page 1390) in NSScriptCommand. It performs the same function as the overridden method, with a critical difference: it causes the container specifier part of the passed-in object specifier to become the receiver specifier of the command, and the key part of the passed-in object specifier to become the key specifier. If, for example, *receiversRef* is a specifier for the color of the third rectangle, the receiver specifier is the third rectangle, while the key specifier is the color.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptStandardSuiteCommands.h

NSSocketPort Class Reference

Inherits from	NSPort : NSObject
Conforms to	NSCoding (NSPort) NSCopying (NSPort) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSPort.h
Companion guide	Distributed Objects Programming Topics

Overview

`NSSocketPort` is a subclass of `NSPort` that represents a BSD socket. An `NSSocketPort` object can be used as an endpoint for distributed object connections. Companion classes, `NSMachPort` and `NSMessagePort`, allow for local (on the same machine) communication only. The `NSSocketPort` class allows for both local and remote communication, but may be more expensive than the others for the local case.

Note: The `NSSocketPort` class conforms to the `NSCoding` protocol, but only supports coding by an `NSPortCoder`. `NSPort` and its other subclasses do not support archiving.

Tasks

Creating Instances

- `init` (page 1466)
  Initializes the receiver as a local TCP/IP socket of type `SOCK_STREAM`.
- `initWithTCPPort:` (page 1469)
  Initializes the receiver as a local TCP/IP socket of type `SOCK_STREAM`, listening on a specified port number.
- `initWithProtocolFamily:socketType:protocol:address:` (page 1468)
  Initializes the receiver as a local socket with the provided arguments.
- `initWithProtocolFamily:socketType:protocol:socket:` (page 1469)
  Initializes the receiver with a previously created local socket.

- `initRemoteWithTCPPort:host:` (page 1467)
  Initializes the receiver as a TCP/IP socket of type `SOCK_STREAM` that can connect to a remote host on a specified port.
- `initRemoteWithProtocolFamily:socketType:protocol:address:` (page 1467)
  Initializes the receiver as a remote socket with the provided arguments.

Getting Information

- `address` (page 1466)
  Returns the receiver's socket address structure.
- `protocol` (page 1470)
  Returns the protocol that the receiver uses for communication.
- `protocolFamily` (page 1470)
  Returns the protocol family that the receiver uses for communication.
- `socket` (page 1470)
  Returns the receiver's native socket identifier on the platform.
- `socketType` (page 1470)
  Returns the receiver's socket type.

Instance Methods

address

Returns the receiver's socket address structure.

```
- (NSData *)address
```

Return Value
The receiver's socket address structure stored inside an `NSData` object.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `initWithProtocolFamily:socketType:protocol:address:` (page 1468)
- `initRemoteWithProtocolFamily:socketType:protocol:address:` (page 1467)

Declared In
`NSPort.h`

init

Initializes the receiver as a local TCP/IP socket of type `SOCK_STREAM`.

```
- (id)init
```

Return Value
An initialized local TCP/IP socket port of type `SOCK_STREAM`.

Discussion
The port number is selected by the system.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `initWithTCPPort:` (page 1469)
- `initWithProtocolFamily:socketType:protocol:address:` (page 1468)

Declared In
`NSPort.h`

initRemoteWithProtocolFamily:socketType:protocol:address:

Initializes the receiver as a remote socket with the provided arguments.

```
- (id)initRemoteWithProtocolFamily:(int)family socketType:(int)type
    protocol:(int)protocol address:(NSData *)address
```

Parameters

family
The protocol family for the socket port.

type
The type of socket.

protocol
The specific protocol to use from the the protocol family.

address
The family-specific socket address for the receiver copied into an `NSData` object.

Discussion
A connection is not opened to the remote address until data is sent.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `initRemoteWithTCPPort:host:` (page 1467)

Declared In
`NSPort.h`

initRemoteWithTCPPort:host:

Initializes the receiver as a TCP/IP socket of type `SOCK_STREAM` that can connect to a remote host on a specified port.

```
- (id)initRemoteWithTCPPort:(unsigned short)port host:(NSString *)hostName
```

Parameters

port

The port to connect to.

hostName

The host name to connect to. *hostName* may be either a host name or an IPv4-style address.

Return Value

A TCP/IP socket port of type SOCK_STREAM that can connect to the remote host *hostName* on port *port*.

Discussion

A connection is not opened to the remote host until data is sent.

Availability

Available in Mac OS X v10.0 and later.

See Also

- initRemoteWithProtocolFamily:socketType:protocol:address: (page 1467)

Declared In

NSPort.h

initWithProtocolFamily:socketType:protocol:address:

Initializes the receiver as a local socket with the provided arguments.

```
- (id)initWithProtocolFamily:(int)family socketType:(int)type protocol:(int)protocol
    address:(NSData *)address
```

Parameters

family

The protocol family for the socket port.

type

The type of socket.

protocol

The specific protocol to use from the the protocol family.

address

The family-specific socket address for the receiver copied into an NSData object.

Return Value

A local socket port initialized with the provided arguments.

Discussion

The receiver must be added to a run loop before it can accept connections or receive messages. Incoming messages are passed to the receiver's delegate method handlePortMessage: (page 1255).

To create a standard TCP/IP socket, use initWithTCPPort: (page 1469).

Availability

Available in Mac OS X v10.0 and later.

See Also

- init (page 1466)
- initWithProtocolFamily:socketType:protocol:socket: (page 1469)

Declared In
NSPort.h

initWithProtocolFamily:socketType:protocol:socket:

Initializes the receiver with a previously created local socket.

```
- (id)initWithProtocolFamily:(int)family socketType:(int)type protocol:(int)protocol
    socket:(NSSocketNativeHandle)sock
```

Parameters

family
The protocol family for the provided socket.

type
The type of the provided socket.

protocol
The specific protocol the provided socket uses.

sock
The previously created socket.

Return Value
A local socket port initialized with the provided socket.

Availability
Available in Mac OS X v10.0 and later.

See Also
- initWithProtocolFamily:socketType:protocol:address: (page 1468)

Declared In
NSPort.h

initWithTCPPort:

Initializes the receiver as a local TCP/IP socket of type SOCK_STREAM, listening on a specified port number.

```
- (id)initWithTCPPort:(unsigned short)port
```

Parameters

port
The port number for the newly created socket port to listen on. If *port* is 0, the system will assign a port number.

Return Value
An initialized local TCP/IP socket of type SOCK_STREAM, listening on port *port*.

Availability
Available in Mac OS X v10.0 and later.

See Also
- init (page 1466)
- initWithProtocolFamily:socketType:protocol:address: (page 1468)

Declared In
NSPort.h

protocol

Returns the protocol that the receiver uses for communication.

```
- (int)protocol
```

Return Value
The protocol the receiver uses for communication.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSPort.h

protocolFamily

Returns the protocol family that the receiver uses for communication.

```
- (int)protocolFamily
```

Return Value
The protocol family the receiver uses for communication.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSPort.h

socket

Returns the receiver's native socket identifier on the platform.

```
- (NSSocketNativeHandle)socket
```

Return Value
The native socket identifier on the platform. For Mac OS X, this is an integer file descriptor.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSPort.h

socketType

Returns the receiver's socket type.

```
- (int)socketType
```

Return Value
The receiver's socket type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSPort.h`

NSSocketPortNameServer Class Reference

Inherits from	NSPortNameServer : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSPortNameServer.h
Companion guide	Distributed Objects Programming Topics

Overview

This port name server takes and returns instances of NSSocketPort.

Port removal functionality is supported by the `removePortForName:` (page 1478) method and should be used to remove invalid socket ports.

Unlike the other port name servers, NSSocketPortNameServer can operate over a network. By registering your socket ports, you make them available to other computers on the local network without hard-coding the TCP port numbers. Clients just need to know the name of the port.

NSPortNameServer is implemented using `NSNetService` and registers ports in the local network domain. The registered name of a port must be unique within the local domain, not just the local host. The name server only supports TCP/IP (either IPv4 or IPv6) sockets.

Note: Prior to Mac OS X v10.2, NSSocketPortNameServer was inoperable.

Tasks

Getting the Server Object

`+ sharedInstance` (page 1474)
Returns the shared socket port name server.

Looking Up Ports

- `portForName:` (page 1475)
  Looks up and returns the port registered under the specified name on the local host.
- `portForName:host:` (page 1475)
  Looks up and returns the port registered under the specified name on a specified host.
- `portForName:host:nameServerPortNumber:` (page 1476)
  Looks up and returns the port registered under the specified name on a specified host.

Registering and Removing Ports

- `registerPort:name:` (page 1477)
  Registers a given port as a network service with the specified name in the local domain.
- `registerPort:name:nameServerPortNumber:` (page 1477)
  Registers a given port as a network service with the specified name in the local domain.
- `removePortForName:` (page 1478)
  Unregisters the port for a given name on the local host.

Configuring the Default Port Number

- `defaultNameServerPortNumber` (page 1475)
  Returns the port number used to contact the name server.
- `setDefaultNameServerPortNumber:` (page 1478)
  Sets the default port number used to contact the name server.

Class Methods

sharedInstance

Returns the shared socket port name server.

```
+ (id)sharedInstance
```

Return Value
The single instance of `NSSocketPortNameServer` with which you register and look up `NSSocketPort` objects.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSPortNameServer.h`

Instance Methods

defaultNameServerPortNumber

Returns the port number used to contact the name server.

```
- (uint16_t)defaultNameServerPortNumber
```

Return Value
The port number used to contact the name server. This value is currently ignored.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setDefaultNameServerPortNumber:` (page 1478)

Declared In
`NSPortNameServer.h`

portForName:

Looks up and returns the port registered under the specified name on the local host.

```
- (NSPort *)portForName:(NSString *)portName
```

Parameters
portName
The name of the desired port.

Return Value
The port associated with *portName* on the local host. Returns `nil` if no such port exists.

Discussion
Invokes `portForName:host:nameServerPortNumber:` (page 1476) with `nil` as the host name and 0 as the name server port number.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSPortNameServer.h`

portForName:host:

Looks up and returns the port registered under the specified name on a specified host.

```
- (NSPort *)portForName:(NSString *)portName host:(NSString *)hostName
```

Parameters

portName

The name of the desired port.

hostName

The name of the host. *hostName* is an Internet domain name (for example, "sales.anycorp.com"). If *hostName* is nil or empty, the local host is checked.

Return Value

The port associated with *portName* on the host *hostName*. Returns nil if no such port exists.

Discussion

Invokes portForName:host:nameServerPortNumber: (page 1476) with 0 as the name server port number.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSPortNameServer.h

portForName:host:nameServerPortNumber:

Looks up and returns the port registered under the specified name on a specified host.

```
- (NSPort *)portForName:(NSString *)portName host:(NSString *)hostName
    nameServerPortNumber:(uint16_t)portNumber
```

Parameters

portName

The name of the desired port.

hostName

The name of the host. *hostName* is an Internet domain name (for example, "sales.anycorp.com") or IP address (IPv4 or IPv6). If *hostName* is nil or empty, the local host is checked. If *hostName* is @"*", all hosts on the local network are checked.

portNumber

The *portNumber* parameter is ignored.

Return Value

The port associated with *portName* on the host *hostName*. Returns nil if no such port exists.

Availability

Available in Mac OS X v10.0 and later.

See Also

- portForName: (page 1475)
- portForName:host: (page 1475)
- registerPort:name:nameServerPortNumber: (page 1477)

Declared In

NSPortNameServer.h

registerPort:name:

Registers a given port as a network service with the specified name in the local domain.

```
- (BOOL)registerPort:(NSPort *)port name:(NSString *)portName
```

Parameters

port

The port to make available.

portName

The name for the port.

Return Value

`YES` if successful, `NO` otherwise.

Discussion

Invokes `registerPort:name:nameServerPortNumber:` (page 1477) with 0 as the name server port number.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`NSPortNameServer.h`

registerPort:name:nameServerPortNumber:

Registers a given port as a network service with the specified name in the local domain.

```
- (BOOL)registerPort:(NSPort *)port name:(NSString *)portName
    nameServerPortNumber:(uint16_t)portNumber
```

Parameters

port

The port to make available.

portName

The name for the port.

portNumber

The *portNumber* parameter is ignored.

Return Value

`YES` if successful, `NO` otherwise.

Special Considerations

If your application has already registered a port under the name *portName*, this method replaces it with *port*.

If the local domain already has a port named *portName* registered, this method could return `YES` before the name collision is detected. To detect a potential name collision, you can invoke `portForName:host:` (page 1475) with a `host` argument of `@"*"` to test if *portName* is already taken. This, however, leaves a race condition wherein another process can register a port under *portName* after `portForName:host:` (page 1475) returns but before you register *port*. If this is an unacceptable risk for your application, you can also invoke `portForName:host:` (page 1475) some finite time after registering your port to test if you get the same port back.

Availability
Available in Mac OS X v10.0 and later.

See Also
- portForName:host:nameServerPortNumber: (page 1476)

Declared In
NSPortNameServer.h

removePortForName:

Unregisters the port for a given name on the local host.

```
- (BOOL)removePortForName:(NSString *)portName
```

Parameters

portName
The name of the port to unregister.

Return Value
YES if successful, otherwise NO.

Discussion
If the operation is successful, the port can no longer be looked up using the name *portName*. Other applications that already have a reference to the port can continue to use it until it becomes invalid.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSPortNameServer.h

setDefaultNameServerPortNumber:

Sets the default port number used to contact the name server.

```
- (void)setDefaultNameServerPortNumber:(uint16_t)portNumber
```

Parameters

portNumber
The new port number used to contact the name server. This value is currently ignored.

Availability
Available in Mac OS X v10.0 and later.

See Also
- defaultNameServerPortNumber (page 1475)

Declared In
NSPortNameServer.h

NSSortDescriptor Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.3 and later.
Declared in	Foundation/NSSortDescriptor.h
Companion guide	Sort Descriptor Programming Topics
Related sample code	CoreRecipes GridCalendar iSpend NSOperationSample SpotlightFortunes

Overview

An instance of `NSSortDescriptor` describes a basis for ordering objects by specifying the property to use to compare the objects, the method to use to compare the properties, and whether the comparison should be ascending or descending. Instances of `NSSortDescriptor` are immutable.

You construct an instance of `NSSortDescriptor` by specifying the key path of the property to be compared, the order of the sort (ascending or descending), and (optionally) a selector to use to perform the comparison. The three-argument constructor allows you to specify other comparison selectors such as `caseInsensitiveCompare:` and `localizedCompare:`. Sorting raises an exception if the objects to be sorted do not respond to the sort descriptor's comparison selector.

Note: Many of the descriptions of `NSSortDescriptor` methods refer to "property key". This, briefly, is a string (key) that identifies a property (an attribute or relationship) of an object. You can find a discussion of this terminology in "Object Modeling" in *Cocoa Fundamentals Guide* and in *Key-Value Coding Programming Guide*.

There are a number of situations in which you can use sort descriptors, for example:

- To sort an array (an instance of `NSArray` or `NSMutableArray`—see `sortedArrayUsingDescriptors:` and `sortUsingDescriptors:`)

- To directly compare two objects (see compareObject:toObject: (page 1481))
- To specify how the elements in a table view should be arranged (see `sortDescriptors`)
- To specify how the elements managed by an array controller should be arranged (see `sortDescriptors`)
- If you are using Core Data, to specify the ordering of objects returned from a fetch request (see `sortDescriptors`)

Adopted Protocols

NSCoding
- `- encodeWithCoder:` (page 2034)
- `- initWithCoder:` (page 2034)

NSCopying
- `- copyWithZone:` (page 2042)

Tasks

Initializing a Sort Descriptor

`- initWithKey:ascending:` (page 1482)
Returns an `NSSortDescriptor` object initialized with a given property key path and sort order, and with the default comparison selector.

`- initWithKey:ascending:selector:` (page 1482)
Returns an `NSSortDescriptor` object initialized with a given property key path, sort order, and comparison selector.

Getting Information About a Sort Descriptor

`- ascending` (page 1481)
Returns a Boolean value that indicates whether the receiver specifies sorting in ascending order.

`- key` (page 1483)
Returns the receiver's property key path.

`- selector` (page 1484)
Returns the selector the receiver specifies to use when comparing objects.

Using Sort Descriptors

`- compareObject:toObject:` (page 1481)
Returns an `NSComparisonResult` value that indicates the ordering of two given objects.

- reversedSortDescriptor (page 1483)
  Returns a copy of the receiver with the sort order reversed.

Instance Methods

ascending

Returns a Boolean value that indicates whether the receiver specifies sorting in ascending order.

```
- (BOOL)ascending
```

Return Value
YES if the receiver specifies sorting in ascending order, otherwise NO.

Availability
Available in Mac OS X v10.3 and later.

Declared In
NSSortDescriptor.h

compareObject:toObject:

Returns an NSComparisonResult value that indicates the ordering of two given objects.

```
- (NSComparisonResult)compareObject:(id)object1 toObject:(id)object2
```

Parameters

object1
  The object to compare with *object2*. This object must have a property accessible using the key-path specified by key (page 1483).

  This value must not be nil. If the value is nil, the behavior is undefined and may change in future versions of Mac OS X.

object2
  The object to compare with *object1*. This object must have a property accessible using the key-path specified by key (page 1483).

  This value must not be nil. If the value is nil, the behavior is undefined and may change in future versions of Mac OS X.

Return Value
NSOrderedAscending if *object1* is less than *object2*, NSOrderedDescending if *object1* is greater than *object2*, or NSOrderedSame if *object1* is equal to *object2*.

Discussion
The ordering is determined by comparing, using the selector specified selector (page 1484), the values of the properties specified by key (page 1483) of *object1* and *object2*.

Availability
Available in Mac OS X v10.3 and later.

Declared In
NSSortDescriptor.h

initWithKey:ascending:

Returns an NSSortDescriptor object initialized with a given property key path and sort order, and with the default comparison selector.

```
- (id)initWithKey:(NSString *)keyPath ascending:(BOOL)ascending
```

Parameters

keyPath

The property key to use when performing a comparison. In the comparison, the property is accessed using key-value coding (see *Key-Value Coding Programming Guide*).

ascending

YES if the receiver specifies sorting in ascending order, otherwise NO.

Return Value
An NSSortDescriptor object initialized with the property key path specified by *keyPath*, sort order specified by *ascending*, and the default comparison selector (compare:).

Availability
Available in Mac OS X v10.3 and later.

See Also
- initWithKey:ascending:selector: (page 1482)

Related Sample Code
CoreRecipes
NSOperationSample
PredicateEditorSample
SimpleCalendar
SpotlightFortunes

Declared In
NSSortDescriptor.h

initWithKey:ascending:selector:

Returns an NSSortDescriptor object initialized with a given property key path, sort order, and comparison selector.

```
- (id)initWithKey:(NSString *)keyPath ascending:(BOOL)ascending
    selector:(SEL)selector
```

Parameters

keyPath

The property key to use when performing a comparison. In the comparison, the property is accessed using key-value coding (see *Key-Value Coding Programming Guide*).

ascending

YES if the receiver specifies sorting in ascending order, otherwise NO.

selector

The method to use when comparing the properties of objects, for example `caseInsensitiveCompare:` or `localizedCompare:`. The selector must specify a method implemented by the value of the property identified by *keyPath*. The selector used for the comparison is passed a single parameter, the object to compare against `self`, and must return the appropriate `NSComparisonResult` constant. The selector must have the same method signature as:

```
- (NSComparisonResult)localizedCompare:(NSString *)aString
```

Return Value
An `NSSortDescriptor` object initialized with the property key path specified by *keyPath*, sort order specified by *ascending*, and the selector specified by *selector*.

Availability
Available in Mac OS X v10.3 and later.

See Also
`- initWithKey:ascending:` (page 1482)

Related Sample Code
GridCalendar

Declared In
`NSSortDescriptor.h`

key

Returns the receiver's property key path.

```
- (NSString *)key
```

Return Value
The receiver's property key path.

Discussion
This key path specifies the property that is compared during sorting.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
iSpend

Declared In
`NSSortDescriptor.h`

reversedSortDescriptor

Returns a copy of the receiver with the sort order reversed.

```
- (id)reversedSortDescriptor
```

Return Value
A copy of the receiver with the sort order reversed

Availability
Available in Mac OS X v10.3 and later.

Declared In
`NSSortDescriptor.h`

selector

Returns the selector the receiver specifies to use when comparing objects.

```
- (SEL)selector
```

Return Value
The selector the receiver specifies to use when comparing objects.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`NSSortDescriptor.h`

NSSpecifierTest Class Reference

Inherits from	NSScriptWhoseTest : NSObject
Conforms to	NSCoding (NSScriptWhoseTest) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptWhoseTests.h
Companion guide	Cocoa Scripting Guide

Overview

Instances of this class represent a Boolean expression; they evaluate an object specifier and compare the resulting object to another object using a given comparison method. For more information on `NSSpecifierTest`, see the method description for its sole public method, its initializer, `initWithObjectSpecifier:comparisonOperator:testObject:` (page 1486).

When an `NSSpecifierTest` object is properly initialized, it holds two objects:

- A "value" or "test" object used as the basis of the comparison; this object can be a regular object or object specifier (such as "blue" in "words whose color is blue").
- An object specifier evaluating to the container ("words").

The instance also encapsulates a selector identifying the method performing this comparison. The informal protocol `NSComparisonMethods` defines a set of comparison methods useful for this purpose, while `NSScriptingComparisonMethods` describes additional methods you may need to use for scripting.

The test object is compared, using the selector, against each object in the container. Specifiers in these tests usually have `containerIsObjectBeingTested` (page 1415) invoked on their topmost container.

You should rarely need to subclass `NSSpecifierTest`.

Tasks

Initializing a Specifier Test

`- initWithObjectSpecifier:comparisonOperator:testObject:` (page 1486)
Returns a specifier test initialized to evaluate a test object against an object specified by an object specifier using a given comparison operation.

Instance Methods

initWithObjectSpecifier:comparisonOperator:testObject:

Returns a specifier test initialized to evaluate a test object against an object specified by an object specifier using a given comparison operation.

```
- (id)initWithObjectSpecifier:(NSScriptObjectSpecifier *)obj1
    comparisonOperator:(NSTestComparisonOperation)compOp testObject:(id)obj2
```

Parameters

obj1
An object specifier.

compOp
The comparison operation.

obj2
The object against which to evaluate the object specified by *obj1*.

Return Value
A specifier test initialized to evaluate (*obj2*) against an object specified by *obj1* using the comparison operation *compOp*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptWhoseTests.h`

Constants

NSTestComparisonOperation

These are passed to initWithObjectSpecifier:comparisonOperator:testObject: (page 1486) to specify the comparison operator.

```
typedef enum {
    NSEqualToComparison = 0,
    NSLessThanOrEqualToComparison,
    NSLessThanComparison,
    NSGreaterThanOrEqualToComparison,
    NSGreaterThanComparison,
    NSBeginsWithComparison,
    NSEndsWithComparison,
    NSContainsComparison
} NSTestComparisonOperation;
```

Constants

`NSEqualToComparison`

Binary comparison operator that results in `YES` if the two objects are equal.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptWhoseTests.h`.

`NSLessThanOrEqualToComparison`

Binary comparison operator that results in `YES` if the value of the test object is equal to or less than the value of the other object.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptWhoseTests.h`.

`NSLessThanComparison`

Binary comparison operator that results in `YES` if the value of the test object is less than the value of the other object.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptWhoseTests.h`.

`NSGreaterThanOrEqualToComparison`

Binary comparison operator that results in `YES` if the value of the test object is greater than or equal to the value of the other object.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptWhoseTests.h`.

`NSGreaterThanComparison`

Binary comparison operator that results in `YES` if the value of the test object is greater than the value of the other object.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptWhoseTests.h`.

`NSBeginsWithComparison`

Binary containment operator that results in `YES` if the test object is a list or string that matches the beginning of the other object (which is also a list or string).

Available in Mac OS X v10.0 and later.

Declared in `NSScriptWhoseTests.h`.

`NSEndsWithComparison`

Binary containment operator that results in `YES` if the test object is a list or string that matches the end of the other object (which is also a list or string).

Available in Mac OS X v10.0 and later.

Declared in `NSScriptWhoseTests.h`.

`NSContainsComparison`

Binary containment operator that results in `YES` if the test object is a list or string that matches the other object (which is also a list or string) at any location.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptWhoseTests.h`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptWhoseTests.h`

NSSpellServer Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSSpellServer.h
Companion guide	Spell Checking

Overview

The `NSSpellServer` class gives you a way to make your application's spell checker available as a **spelling service** available to any application.

A **service provider** is an application that declares its availability in a standard way, so that any other applications that wish to use it can do so. If you build a spelling checker that makes use of the `NSSpellServer` class and list it as an available service, then users of any application that makes use of NSSpellChecker or includes a Services menu will see your spelling checker as one of the available dictionaries.

Tasks

Configuring Spelling Servers

- `setDelegate:` (page 1492)
  Assigns the specified delegate to the receiver.
- `delegate` (page 1490)
  Returns the receiver's delegate.

Providing Spelling Services

- `registerLanguage:byVendor:` (page 1491)
  Notifies the receiver of a language your spelling checker can check.
- `run` (page 1491)
  Causes the receiver to start listening for spell-checking requests.

Managing the Spell-Checking Process

`- isWordInUserDictionaries:caseSensitive:` (page 1490)
Indicates whether a given word is in the user's list of learned words or the document's list of words to ignore.

`- spellServer:didForgetWord:inLanguage:` (page 1493) *delegate method*
Notifies the delegate that the sender has removed the specified word from the user's list of acceptable words in the specified language.

`- spellServer:didLearnWord:inLanguage:` (page 1493) *delegate method*
Notifies the delegate that the sender has added the specified word to the user's list of acceptable words in the specified language.

`- spellServer:findMisspelledWordInString:language:wordCount:countOnly:` (page 1494) *delegate method*
Asks the delegate to search for a misspelled word in a given string, using the specified language, and marking the first misspelled word found by returning its range within the string.

`- spellServer:suggestCompletionsForPartialWordRange:inString:language:` (page 1494) *delegate method*
This delegate method returns an array of possible word completions from the spell checker, based on a partially completed string and a given range.

`- spellServer:suggestGuessesForWord:inLanguage:` (page 1495) *delegate method*
Gives the delegate the opportunity to suggest guesses to the sender for the correct spelling of the given misspelled word in the specified language.

`- spellServer:checkGrammarInString:language:details:` (page 1492) *delegate method*
Gives the delegate the opportunity to customize the grammatical analysis of a given string.

Instance Methods

delegate

Returns the receiver's delegate.

```
- (id)delegate
```

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setDelegate:` (page 1492)

Declared In
`NSSpellServer.h`

isWordInUserDictionaries:caseSensitive:

Indicates whether a given word is in the user's list of learned words or the document's list of words to ignore.

```
- (BOOL)isWordInUserDictionaries:(NSString *)word caseSensitive:(BOOL)caseSensitive
```

Parameters

word

The word to compare with those in the user dictionaries.

caseSensitive

Specifies whether the comparison is case sensitive.

Return Value

A Boolean value indicating whether the word is in the user dictionaries. If `YES`, the word is acceptable to the user.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSSpellServer.h`

registerLanguage:byVendor:

Notifies the receiver of a language your spelling checker can check.

```
- (BOOL)registerLanguage:(NSString *)language byVendor:(NSString *)vendor
```

Parameters

language

A string specifying the English name of a language on Apple's list of languages.

vendor

A string that identifies the vendor (to distinguish your spelling checker from those that others may offer for the same language).

Return Value

Returns `YES` if the language is registered, `NO` if for some reason it can't be registered.

Discussion

If your spelling checker supports more than one language, it should invoke this method once for each language. Registering a language-vendor combination causes it to appear in the Spelling panel's pop-up menu of spelling checkers.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSSpellServer.h`

run

Causes the receiver to start listening for spell-checking requests.

```
- (void)run
```

Discussion

This method starts a loop that never returns; you need to set the `NSSpellServer` object's delegate before sending this message.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `setDelegate:` (page 1492)

Declared In
`NSSpellServer.h`

setDelegate:

Assigns the specified delegate to the receiver.

```
- (void)setDelegate:(id)anObject
```

Parameters

anObject
The delegate assigned to the receiver.

Discussion
Because the delegate is where the real work is done, this step is essential before telling the `NSSpellServer` object to run.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `delegate` (page 1490)
- `run` (page 1491)

Declared In
`NSSpellServer.h`

Delegate Methods

spellServer:checkGrammarInString:language:details:

Gives the delegate the opportunity to customize the grammatical analysis of a given string.

```
- (NSRange)spellServer:(NSSpellServer *)sender checkGrammarInString:(NSString
    *)string language:(NSString *)language details:(NSArray **)outDetails
```

Parameters

sender
Spell server satisfying a grammatical analysis request.

string
String to analyze.

language
Language use in *string*. When `nil`, the language selected in the Spelling panel is used.

outDetails
On output, dictionaries describing grammar-analysis details within the flagged grammatical unit. See the `NSSpellServer` class for information about these dictionaries.

Return Value
Location of the first flagged grammatical unit within *string*.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSSpellServer.h`

spellServer:didForgetWord:inLanguage:

Notifies the delegate that the sender has removed the specified word from the user's list of acceptable words in the specified language.

```
- (void)spellServer:(NSSpellServer *)sender didForgetWord:(NSString *)word
    inLanguage:(NSString *)language
```

Parameters
sender
The `NSSpellServer` object that removed the word.

word
The word that was removed.

language
The language of the removed word.

Discussion
If your delegate maintains a similar auxiliary word list, you may wish to edit the list accordingly.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSSpellServer.h`

spellServer:didLearnWord:inLanguage:

Notifies the delegate that the sender has added the specified word to the user's list of acceptable words in the specified language.

```
- (void)spellServer:(NSSpellServer *)sender didLearnWord:(NSString *)word
    inLanguage:(NSString *)language
```

Parameters
sender
The `NSSpellServer` object that added the word.

word
The word that was added.

language
The language of the added word.

Discussion
If your delegate maintains a similar auxiliary word list, you may wish to edit the list accordingly.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSSpellServer.h

spellServer:findMisspelledWordInString:language:wordCount:countOnly:

Asks the delegate to search for a misspelled word in a given string, using the specified language, and marking the first misspelled word found by returning its range within the string.

```
- (NSRange)spellServer:(NSSpellServer *)sender findMisspelledWordInString:(NSString
    *)stringToCheck language:(NSString *)language wordCount:(int32_t *)wordCount
    countOnly:(BOOL)countOnly
```

Parameters
sender
The NSSpellServer object that sent this message.

stringToCheck
The string to search for the misspelled word.

language
The language to use for the search.

wordCount
On output, returns by reference the number of words from the beginning of the string object until the misspelled word (or the end of string).

countOnly
If YES, the method only counts the words in the string object and does not spell checking.

Return Value
The range of the misspelled word within the given string.

Discussion
Send isWordInUserDictionaries:caseSensitive: (page 1490) to the spelling server to determine if the word exists in the user's language dictionaries.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSSpellServer.h

spellServer:suggestCompletionsForPartialWordRange:inString:language:

This delegate method returns an array of possible word completions from the spell checker, based on a partially completed string and a given range.

```
- (NSArray *)spellServer:(NSSpellServer *)sender
    suggestCompletionsForPartialWordRange:(NSRange)range inString:(NSString *)string
     language:(NSString *)language
```

Parameters

sender

The `NSSpellServer` object that sent this message.

range

The range of the partially completed word.

string

The string containing the partial word range.

language

The language to use for the completion.

Return Value

An array of `NSString` objects indicating possible completions.

Discussion

See `completionsForPartialWordRange:inString:language:inSpellDocumentWithTag:` in `NSSpellChecker` for more information.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`NSSpellServer.h`

spellServer:suggestGuessesForWord:inLanguage:

Gives the delegate the opportunity to suggest guesses to the sender for the correct spelling of the given misspelled word in the specified language.

```
- (NSArray *)spellServer:(NSSpellServer *)sender suggestGuessesForWord:(NSString
    *)word inLanguage:(NSString *)language
```

Parameters

sender

The `NSSpellServer` object that sent this message.

word

The misspelled word.

language

The language to use for the guesses.

Return Value

An array of `NSString` objects indicating possible correct spellings.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSSpellServer.h`

Constants

Grammatical-Analysis Details

These constants are use as the keys in the outDetails dictionaries returned by `spellServer:checkGrammarInString:language:details:` (page 1492) and `NSSpellChecker -checkGrammarOfString:startingAt:language:wrap:inSpellingDocumentWithTag:details:`.

```
FOUNDATION_EXPORT NSString *const NSGrammarRange;
FOUNDATION_EXPORT NSString *const NSGrammarUserDescription;
FOUNDATION_EXPORT NSString *const NSGrammarCorrections;
```

Constants

`NSGrammarRange`

`NSGrammarUserDescription`

`NSGrammarCorrections`

Declared In

`NSSpellServer.h`

NSStream Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.3 and later.
Declared in	Foundation/NSStream.h
Companion guide	Stream Programming Guide for Cocoa
Related sample code	CocoaEcho CocoaHTTPServer CocoaSOAP

Overview

`NSStream` is an abstract class for objects representing streams. Its interface is common to all Cocoa stream classes, including its concrete subclasses `NSInputStream` and `NSOutputStream`.

`NSStream` objects provide an easy way to read and write data to and from a variety of media in a device-independent way. You can create stream objects for data located in memory, in a file, or on a network (using sockets), and you can use stream objects without loading all of the data into memory at once.

By default, `NSStream` instances that are not file-based are non-seekable, one-way streams (although custom seekable subclasses are possible). Once the data has been provided or consumed, the data cannot be retrieved from the stream.

Subclassing Notes

`NSStream` is an abstract class, incapable of instantiation and intended to be subclassed. It publishes a programmatic interface that all subclasses must adopt and provide implementations for. The two Apple-provided concrete subclasses of `NSStream`, `NSInputStream` and `NSOutputStream`, are suitable for most purposes. However, there might be situations when you want a peer subclass to `NSInputStream` and `NSOutputStream`. For example, you might want a class that implements a full-duplex (two-way) stream, or a class whose instances are capable of seeking through a stream.

Methods to Override

All subclasses must fully implement the following methods, which are presented in functional pairs:

- `open` (page 1501) and `close` (page 1500)

  Implement `open` to open the stream for reading or writing and make the stream available to the client directly or, if the stream object is scheduled on a run loop, to the delegate. Implement `close` to close the stream and remove the stream object from the run loop, if necessary. A closed stream should still be able to accept new properties and report its current properties. Once a stream is closed, it cannot be reopened.

- `delegate` (page 1500) and `setDelegate:` (page 1503)

  Return and set the delegate. By a default, a stream object must be its own delegate; so a `setDelegate:` message with an argument of `nil` should restore this delegate. Do not retain the delegate to prevent retain cycles.

  To learn about delegates and delegation, read "Delegates and Data Sources" in *Cocoa Fundamentals Guide*.

- `scheduleInRunLoop:forMode:` (page 1502) and `removeFromRunLoop:forMode:` (page 1502)

  Implement `scheduleInRunLoop:forMode:` to schedule the stream object on the specified run loop for the specified mode. Implement `removeFromRunLoop:forMode:` to remove the object from the run loop. See the documentation of the `NSRunLoop` class for details. Once the stream object for an open stream is scheduled on a run loop, it is the responsibility of the subclass as it processes stream data to send `stream:handleEvent:` (page 1504) messages to its delegate.

- `propertyForKey:` (page 1501) and `setProperty:forKey:` (page 1503)

  Implement these methods to return and set, respectively, the property value for the specified key. You may add custom properties, but be sure to handle all properties defined by `NSStream` as well.

- `streamStatus` (page 1504) and `streamError` (page 1504)

  Implement `streamStatus` to return the current status of the stream as a `NSStreamStatus` constant; you may define new `NSStreamStatus` constants, but be sure to handle the `NSStream`-defined constants properly. Implement`streamError` to return an `NSError` object representing the current error. You might decide to return a custom `NSError` object that can provide complete and localized information about the error.

Tasks

Creating Streams

`+ getStreamsToHost:port:inputStream:outputStream:` (page 1499)
Creates and returns by reference an `NSInputStream` object and `NSOutputStream` object for a socket connection with a given host on a given port.

Configuring Streams

- `propertyForKey:` (page 1501)
  Returns the receiver's property for a given key.
- `setProperty:forKey:` (page 1503)
  Attempts to set the value of a given property of the receiver and returns a Boolean value that indicates whether the value is accepted by the receiver.
- `delegate` (page 1500)
  Returns the receiver's delegate.
- `setDelegate:` (page 1503)
  Sets the receiver's delegate.

Using Streams

- `open` (page 1501)
  Opens the receiving stream.
- `close` (page 1500)
  Closes the receiver.
- `stream:handleEvent:` (page 1504) *delegate method*
  The delegate receives this message when a given event has occurred on a given stream.

Managing Run Loops

- `scheduleInRunLoop:forMode:` (page 1502)
  Schedules the receiver on a given run loop in a given mode.
- `removeFromRunLoop:forMode:` (page 1502)
  Removes the receiver from a given run loop running in a given mode.

Getting Stream Information

- `streamStatus` (page 1504)
  Returns the receiver's status.
- `streamError` (page 1504)
  Returns an `NSError` object representing the stream error.

Class Methods

getStreamsToHost:port:inputStream:outputStream:

Creates and returns by reference an `NSInputStream` object and `NSOutputStream` object for a socket connection with a given host on a given port.

```
+ (void)getStreamsToHost:(NSHost *)host port:(NSInteger)port
    inputStream:(NSInputStream **)inputStream outputStream:(NSOutputStream
    **)outputStream
```

Parameters

host

The host to which to connect.

port

The port to connect to on *host*.

inputStream

Upon return, contains the input stream. If `nil` is passed, the stream object is not created.

outputStream

Upon return, contains the output stream. If `nil` is passed, the stream object is not created.

Discussion
If neither *port* nor *host* is properly specified, no socket connection is made.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`NSStream.h`

Instance Methods

close

Closes the receiver.

```
- (void)close
```

Discussion
Closing the stream terminates the flow of bytes and releases system resources that were reserved for the stream when it was opened. If the stream has been scheduled on a run loop, closing the stream implicitly removes the stream from the run loop. A stream that is closed can still be queried for its properties.

Availability
Available in Mac OS X v10.3 and later.

See Also
- open (page 1501)

Declared In
`NSStream.h`

delegate

Returns the receiver's delegate.

```
- (id)delegate
```

Return Value
The receiver's delegate.

Discussion
By default, a stream is its own delegate, and subclasses of `NSInputStream` and `NSOutputStream` must maintain this contract.

Availability
Available in Mac OS X v10.3 and later.

See Also
`- setDelegate:` (page 1503)

Declared In
`NSStream.h`

open

Opens the receiving stream.

```
- (void)open
```

Discussion
A stream must be created before it can be opened. Once opened, a stream cannot be closed and reopened.

Availability
Available in Mac OS X v10.3 and later.

See Also
`- close` (page 1500)

Declared In
`NSStream.h`

propertyForKey:

Returns the receiver's property for a given key.

```
- (id)propertyForKey:(NSString *)key
```

Parameters

key

The key for one of the receiver's properties. See "Constants" (page 1505) for a description of the available property-key constants and associated values.

Return Value
The receiver's property for the key *key*.

Availability
Available in Mac OS X v10.3 and later.

See Also
`- setProperty:forKey:` (page 1503)

Declared In
NSStream.h

removeFromRunLoop:forMode:

Removes the receiver from a given run loop running in a given mode.

```
- (void)removeFromRunLoop:(NSRunLoop *)aRunLoop forMode:(NSString *)mode
```

Parameters

aRunLoop
The run loop on which the receiver was scheduled.

mode
The mode for the run loop.

Availability
Available in Mac OS X v10.3 and later.

See Also
- scheduleInRunLoop:forMode: (page 1502)

Declared In
NSStream.h

scheduleInRunLoop:forMode:

Schedules the receiver on a given run loop in a given mode.

```
- (void)scheduleInRunLoop:(NSRunLoop *)aRunLoop forMode:(NSString *)mode
```

Parameters

aRunLoop
The run loop on which to schedule the receiver.

mode
The mode for the run loop.

Discussion
Unless the client is polling the stream, it is responsible for ensuring that the stream is scheduled on at least one run loop and that at least one of the run loops on which the stream is scheduled is being run.

Availability
Available in Mac OS X v10.3 and later.

See Also
- removeFromRunLoop:forMode: (page 1502)

Declared In
NSStream.h

setDelegate:

Sets the receiver's delegate.

```
- (void)setDelegate:(id)delegate
```

Parameters

delegate

The delegate for the receiver.

Discussion

By default, a stream is its own delegate, and subclasses of `NSInputStream` and `NSOutputStream` must maintain this contract. If you override this method in a subclass, passing `nil` must restore the receiver as its own delegate. Delegates are not retained.

To learn about delegates and delegation, read "Delegates and Data Sources" in *Cocoa Fundamentals Guide*.

Availability

Available in Mac OS X v10.3 and later.

See Also

`- delegate` (page 1500)

Declared In

`NSStream.h`

setProperty:forKey:

Attempts to set the value of a given property of the receiver and returns a Boolean value that indicates whether the value is accepted by the receiver.

```
- (BOOL)setProperty:(id)property forKey:(NSString *)key
```

Parameters

property

The value for *key*.

key

The key for one of the receiver's properties. See "Constants" (page 1505) for a description of the available property-key constants and expected values.

Return Value

`YES` if the value is accepted by the receiver, otherwise `NO`.

Availability

Available in Mac OS X v10.3 and later.

See Also

`- propertyForKey:` (page 1501)

Declared In

`NSStream.h`

streamError

Returns an `NSError` object representing the stream error.

```
- (NSError *)streamError
```

Return Value
An `NSError` object representing the stream error, or `nil` if no error has been encountered.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`NSStream.h`

streamStatus

Returns the receiver's status.

```
- (NSStreamStatus)streamStatus
```

Return Value
The receiver's status.

Discussion
See "Constants" (page 1505) for a description of the available `NSStreamStatus` constants.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`NSStream.h`

Delegate Methods

stream:handleEvent:

The delegate receives this message when a given event has occurred on a given stream.

```
- (void)stream:(NSStream *)theStream handleEvent:(NSStreamEvent)streamEvent
```

Parameters

theStream
The stream on which *streamEvent* occurred.

streamEvent
The stream event that occurred,

Discussion
The delegate receives this message only if *theStream* is scheduled on a run loop. The message is sent on the stream object's thread. The delegate should examine *streamEvent* to determine the appropriate action it should take.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`NSStream.h`

Constants

NSStreamStatus

The type declared for the constants listed in "Stream Status Constants" (page 1505).

```
typedef NSUInteger NSStreamStatus;
```

Availability
Available in Mac OS X v10.3 and later.

Declared In
`NSStream.h`

Stream Status Constants

These constants indicate the current status of a stream. They are returned by `streamStatus` (page 1504).

```
typedef enum {
    NSStreamStatusNotOpen = 0,
    NSStreamStatusOpening = 1,
    NSStreamStatusOpen = 2,
    NSStreamStatusReading = 3,
    NSStreamStatusWriting = 4,
    NSStreamStatusAtEnd = 5,
    NSStreamStatusClosed = 6,
    NSStreamStatusError = 7
};
```

Constants

`NSStreamStatusNotOpen`
The stream is not open for reading or writing. This status is returned before the underlying call to open a stream but after it's been created.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

`NSStreamStatusOpening`
The stream is in the process of being opened for reading or for writing. For network streams, this status might include the time after the stream was opened, but while network DNS resolution is happening.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

`NSStreamStatusOpen`
The stream is open, but no reading or writing is occurring.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

`NSStreamStatusReading`
Data is being read from the stream. This status would be returned if code on another thread were to call `streamStatus` (page 1504) on the stream while a `read:maxLength:` (page 767) call (`NSInputStream`) was in progress.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

`NSStreamStatusWriting`
Data is being written to the stream. This status would be returned if code on another thread were to call `streamStatus` (page 1504) on the stream while a `write:maxLength:` (page 1222) call (`NSOutputStream`) was in progress.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

`NSStreamStatusAtEnd`
There is no more data to read, or no more data can be written to the stream. When this status is returned, the stream is in a "non-blocking" mode and no data are available.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

`NSStreamStatusClosed`
The stream is closed (`close` (page 1500) has been called on it).

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

`NSStreamStatusError`
The remote end of the connection can't be contacted, or the connection has been severed for some other reason.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

Declared In
`NSStream.h`

NSStreamEvent

The type declared for the constants listed in "Stream Event Constants" (page 1507).

```
typedef NSUInteger NSStreamEvent;
```

Availability
Available in Mac OS X v10.3 and later.

Declared In
`NSStream.h`

Stream Event Constants

One or more of these constants may be sent to the delegate as a bit field in the second parameter of `stream:handleEvent:` (page 1504).

```
typedef enum {
    NSStreamEventNone = 0,
    NSStreamEventOpenCompleted = 1 << 0,
    NSStreamEventHasBytesAvailable = 1 << 1,
    NSStreamEventHasSpaceAvailable = 1 << 2,
    NSStreamEventErrorOccurred = 1 << 3,
    NSStreamEventEndEncountered = 1 << 4
};
```

Constants

`NSStreamEventNone`
No event has occurred.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

`NSStreamEventOpenCompleted`
The open has completed successfully.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

`NSStreamEventHasBytesAvailable`
The stream has bytes to be read.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

`NSStreamEventHasSpaceAvailable`
The stream can accept bytes for writing.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

`NSStreamEventErrorOccurred`
An error has occurred on the stream.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

`NSStreamEventEndEncountered`
The end of the stream has been reached.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

Declared In

`NSStream.h`

NSStream Property Keys

`NSStream` defines these string constants as keys for accessing stream properties using `propertyForKey:` (page 1501) and setting properties with `setProperty:forKey:` (page 1503):

```
extern NSString * const NSStreamSocketSecurityLevelKey  ;
extern NSString * const NSStreamSocketSecurityLevelNone ;
extern NSString * const NSStreamSocketSecurityLevelSSLv2 ;
extern NSString * const NSStreamSocketSecurityLevelSSLv3 ;
extern NSString * const NSStreamSocketSecurityLevelTLSv1 ;
extern NSString * const NSStreamSocketSecurityLevelNegotiatedSSL;
extern NSString * const NSStreamSOCKSProxyConfigurationKey  ;
extern NSString * const NSStreamSOCKSProxyHostKey  ;
extern NSString * const NSStreamSOCKSProxyPortKey  ;
extern NSString * const NSStreamSOCKSProxyVersionKey  ;
extern NSString * const NSStreamSOCKSProxyUserKey  ;
extern NSString * const NSStreamSOCKSProxyPasswordKey ;
extern NSString * const NSStreamSOCKSProxyVersion4  ;
extern NSString * const NSStreamSOCKSProxyVersion5  ;
extern NSString * const NSStreamDataWrittenToMemoryStreamKey ;
extern NSString * const NSStreamFileCurrentOffsetKey  ;
```

Constants

`NSStreamSocketSecurityLevelKey`

The security level of the target stream. May be one of the following values: `NSStreamSocketSecurityLevelNone`, `NSStreamSocketSecurityLevelSSLv2`, `NSStreamSocketSecurityLevelSSLv3`, `NSStreamSocketSecurityLevelTLSv1`, or `NSStreamSocketSecurityLevelNegotiatedSSL`.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

`NSStreamSOCKSProxyConfigurationKey`

Value is an `NSDictionary` object containing SOCKS proxy configuration information.

The dictionary returned from the System Configuration framework for SOCKS proxies usually suffices.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

`NSStreamDataWrittenToMemoryStreamKey`

Value is an `NSData` instance containing the data written to a memory stream.

Use this property when you have an output-stream object instantiated to collect written data in memory. The value of this property is read-only.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

`NSStreamFileCurrentOffsetKey`

Value is an `NSNumber` object containing the current absolute offset of the stream.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

Declared In

`NSStream.h`

NSStream Error Domains

`NSStream` defines these string constants to represent error domains that can be returned by `streamError` (page 1504):

```
extern NSString * const NSStreamSocketSSLErrorDomain  ;
extern NSString * const NSStreamSOCKSErrorDomain  ;
```

Constants

`NSStreamSocketSSLErrorDomain`

The error domain used by `NSError` when reporting SSL errors.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

`NSStreamSOCKSErrorDomain`

The error domain used by `NSError` when reporting SOCKS errors.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

Declared In

`NSStream.h`

Secure-Socket Layer (SSL) Security Level

`NSStream` defines these string constants for specifying the secure-socket layer (SSL) security level.

```
NSString * const NSStreamSocketSecurityLevelNone;
NSString * const NSStreamSocketSecurityLevelSSLv2;
NSString * const NSStreamSocketSecurityLevelSSLv3;
NSString * const NSStreamSocketSecurityLevelTLSv1;
NSString * const NSStreamSocketSecurityLevelNegotiatedSSL
```

Constants

`NSStreamSocketSecurityLevelNone`

Specifies that no security level be set for a socket stream.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

`NSStreamSocketSecurityLevelSSLv2`

Specifies that SSL version 2 be set as the security protocol for a socket stream.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

`NSStreamSocketSecurityLevelSSLv3`

Specifies that SSL version 3 be set as the security protocol for a socket stream.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

`NSStreamSocketSecurityLevelTLSv1`

Specifies that TLS version 1 be set as the security protocol for a socket stream.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

`NSStreamSocketSecurityLevelNegotiatedSSL`

Specifies that the highest level security protocol that can be negotiated be set as the security protocol for a socket stream.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

Discussion
You access and set these values using the `NSStreamSocketSecurityLevelKey` property key.

Declared In
`NSStream.h`

SOCKS Proxy Configuration Values

`NSStream` defines these string constants for use as keys to specify SOCKS proxy configuration values in an `NSDictionary` object.

```
NSString * const NSStreamSOCKSProxyHostKey;
NSString * const NSStreamSOCKSProxyPortKey;
NSString * const NSStreamSOCKSProxyVersionKey;
NSString * const NSStreamSOCKSProxyUserKey;
NSString * const NSStreamSOCKSProxyPasswordKey;
NSString * const NSStreamSOCKSProxyVersion4;
NSString * const NSStreamSOCKSProxyVersion5
```

Constants

`NSStreamSOCKSProxyHostKey`
Value is an `NSString` object that represents the SOCKS proxy host.
Available in Mac OS X v10.3 and later.
Declared in `NSStream.h`.

`NSStreamSOCKSProxyPortKey`
Value is an `NSNumber` object containing an integer that represents the port on which the proxy listens.
Available in Mac OS X v10.3 and later.
Declared in `NSStream.h`.

`NSStreamSOCKSProxyVersionKey`
Value is either `NSStreamSOCKSProxyVersion4` or `NSStreamSOCKSProxyVersion5`.
If this key is not present, `NSStreamSOCKSProxyVersion5` is used by default.
Available in Mac OS X v10.3 and later.
Declared in `NSStream.h`.

`NSStreamSOCKSProxyUserKey`
Value is an `NSString` object containing the user's name.
Available in Mac OS X v10.3 and later.
Declared in `NSStream.h`.

`NSStreamSOCKSProxyPasswordKey`
Value is an `NSString` object containing the user's password.
Available in Mac OS X v10.3 and later.
Declared in `NSStream.h`.

`NSStreamSOCKSProxyVersion4`
Possible value for `NSStreamSOCKSProxyVersionKey`.
Available in Mac OS X v10.3 and later.
Declared in `NSStream.h`.

`NSStreamSOCKSProxyVersion5`

Possible value for `NSStreamSOCKSProxyVersionKey`.

Available in Mac OS X v10.3 and later.

Declared in `NSStream.h`.

Discussion

You set the dictionary object as the current SOCKS proxy configuration using the `NSStreamSOCKSProxyConfigurationKey` key

Declared In

`NSStream.h`

NSString Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSMutableCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSString.h Foundation/NSPathUtilities.h Foundation/NSURL.h
Companion guides	String Programming Guide for Cocoa Property List Programming Guide
Related sample code	Dicey GLSLShowpiece Quartz Composer WWDC 2005 TextEdit StickiesExample TextEditPlus

Overview

The `NSString` class declares the programmatic interface for an object that manages immutable strings. (An immutable string is a text string that is defined when it is created and subsequently cannot be changed. `NSString` is implemented to represent an array of Unicode characters (in other words, a text string).

The mutable subclass of `NSString` is `NSMutableString`.

The `NSString` class has two primitive methods—`length` (page 1580) and `characterAtIndex:` (page 1540)—that provide the basis for all other methods in its interface. The `length` (page 1580) method returns the total number of Unicode characters in the string. `characterAtIndex:` (page 1540) gives access to each character in the string by index, with index values starting at `0`.

`NSString` declares methods for finding and comparing strings. It also declares methods for reading numeric values from strings, for combining strings in various ways, and for converting a string to different forms (such as encoding and case changes).

The Application Kit also uses `NSParagraphStyle` and its subclass `NSMutableParagraphStyle` to encapsulate the paragraph or ruler attributes used by the `NSAttributedString` classes. Additionally, methods to support string drawing are described in NSString Application Kit Additions Reference, found in the Application Kit.

`NSString` is "toll-free bridged" with its Core Foundation counterpart, CFString (see `CFStringRef`). This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSString *` parameter, you can pass a `CFStringRef`, and in a function where you see a `CFStringRef` parameter, you can pass an `NSString` instance (you cast one type to the other to suppress compiler warnings). This also applies to your concrete subclasses of `NSString`. See Interchangeable Data Types for more information on toll-free bridging.

String Objects

`NSString` objects represent character strings in frameworks. Representing strings as objects allows you to use strings wherever you use other objects. It also provides the benefits of encapsulation, so that string objects can use whatever encoding and storage are needed for efficiency while simply appearing as arrays of characters. The cluster's two public classes, `NSString` and `NSMutableString`, declare the programmatic interface for non-editable and editable strings, respectively.

Note: An immutable string is a text string that is defined when it is created and subsequently cannot be changed. An immutable string is implemented as an array of Unicode characters (in other words, a text string). To create and manage an immutable string, use the `NSString` class. To construct and manage a string that can be changed after it has been created, use `NSMutableString`.

The objects you create using `NSString` and `NSMutableString` are referred to as string objects (or, when no confusion will result, merely as strings). The term C string refers to the standard `char *` type. Because of the nature of class clusters, string objects aren't actual instances of the `NSString` or `NSMutableString` classes but of one of their private subclasses. Although a string object's class is private, its interface is public, as declared by these abstract superclasses, `NSString` and `NSMutableString`. The string classes adopt the `NSCopying` and `NSMutableCopying` protocols, making it convenient to convert a string of one type to the other.

Understanding characters

A string object presents itself as an array of Unicode characters (Unicode is a registered trademark of Unicode, Inc.). You can determine how many characters a string object contains with the `length` (page 1580) method and can retrieve a specific character with the `characterAtIndex:` (page 1540) method. These two "primitive" methods provide basic access to a string object.

Most use of strings, however, is at a higher level, with the strings being treated as single entities: You compare strings against one another, search them for substrings, combine them into new strings, and so on. If you need to access string objects character by character, you must understand the Unicode character encoding, specifically issues related to composed character sequences. For details see *The Unicode Standard, Version 4.0* (The Unicode Consortium, Boston: Addison-Wesley, 2003, ISBN 0-321-18578-1) and the Unicode Consortium web site: http://www.unicode.org/. See also Characters and Grapheme Clusters in *String Programming Guide for Cocoa*.

Interpreting UTF-16-encoded data

When creating an `NSString` object from a UTF-16-encoded string (or a byte stream interpreted as UTF-16), if the byte order is not otherwise specified, `NSString` assumes that the UTF-16 characters are big-endian, unless there is a BOM (byte-order mark), in which case the BOM dictates the byte order. When creating an `NSString` object from an array of Unicode characters, the returned string is always native-endian, since the array always contains Unicode characters in native byte order.

Distributed objects

Over distributed-object connections, mutable string objects are passed by-reference and immutable string objects are passed by-copy.

Subclassing Notes

It is possible to subclass `NSString` (and `NSMutableString`), but doing so requires providing storage facilities for the string (which is not inherited by subclasses) and implementing two primitive methods. The abstract `NSString` and `NSMutableString` classes are the public interface of a class cluster consisting mostly of private, concrete classes that create and return a string object appropriate for a given situation. Making your own concrete subclass of this cluster imposes certain requirements (discussed in "Methods to Override" (page 1515)).

Make sure your reasons for subclassing `NSString` are valid. Instances of your subclass should represent a string and not something else. Thus the only attributes the subclass should have are the length of the character buffer it's managing and access to individual characters in the buffer. Valid reasons for making a subclass of `NSString` include providing a different backing store (perhaps for better performance) or implementing some aspect of object behavior differently, such as memory management. If your purpose is to add non-essential attributes or metadata to your subclass of `NSString`, a better alternative would be object composition (see "Alternatives to Subclassing" (page 1516)). Cocoa already provides an example of this with the `NSAttributedString` class.

Methods to Override

Any subclass of `NSString` *must* override the primitive instance methods `length` (page 1580) and `characterAtIndex:` (page 1540). These methods must operate on the backing store that you provide for the characters of the string. For this backing store you can use a static array, a dynamically allocated buffer, a standard `NSString` object, or some other data type or mechanism. You may also choose to override, partially or fully, any other `NSString` method for which you want to provide an alternative implementation. For example, for better performance it is recommended that you override `getCharacters:range:` (page 1555) and give it a faster implementation.

You might want to implement an initializer for your subclass that is suited to the backing store that the subclass is managing. The `NSString` class does not have a designated initializer, so your initializer need only invoke the `init` (page 1178) method of `super`. The `NSString` class adopts the `NSCopying`, `NSMutableCopying`, and `NSCoding` protocols; if you want instances of your own custom subclass created from copying or coding, override the methods in these protocols.

Note that you shouldn't override the `hash` (page 1561) method.

Alternatives to Subclassing

Often a better and easier alternative to making a subclass of `NSString`—or of any other abstract, public class of a class cluster, for that matter—is object composition. This is especially the case when your intent is to add to the subclass metadata or some other attribute that is not essential to a string object. In object composition, you would have an `NSString` object as one instance variable of your custom class (typically a subclass of `NSObject`) and one or more instance variables that store the metadata that you want for the custom object. Then just design your subclass interface to include accessor methods for the embedded string object and the metadata.

If the behavior you want to add supplements that of the existing class, you could write a category on `NSString`. Keep in mind, however, that this category will be in effect for all instances of `NSString` that you use, and this might have unintended consequences.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 2034)
- `initWithCoder:` (page 2034)

NSCopying
- `copyWithZone:` (page 2042)

NSMutableCopying
- `mutableCopyWithZone:` (page 2094)

Tasks

Creating and Initializing Strings

`+ string` (page 1529)
Returns an empty string.

`- init` (page 1563)
Returns an initialized `NSString` object that contains no characters.

`- initWithBytes:length:encoding:` (page 1563)
Returns an initialized `NSString` object containing a given number of bytes from a given C array of bytes in a given encoding.

`- initWithBytesNoCopy:length:encoding:freeWhenDone:` (page 1564)
Returns an initialized `NSString` object that contains a given number of bytes from a given C array of bytes in a given encoding, and optionally frees the array on deallocation.

`- initWithCharacters:length:` (page 1565)
Returns an initialized `NSString` object that contains a given number of characters from a given C array of Unicode characters.

- `initWithCharactersNoCopy:length:freeWhenDone:` (page 1565)
  Returns an initialized `NSString` object that contains a given number of characters from a given C array of Unicode characters.
- `initWithString:` (page 1575)
  Returns an `NSString` object initialized by copying the characters from another given string.
- `initWithCString:encoding:` (page 1570)
  Returns an `NSString` object initialized using the characters in a given C array, interpreted according to a given encoding.
- `initWithUTF8String:` (page 1576)
  Returns an `NSString` object initialized by copying the characters a given C array of UTF8-encoded bytes.
- `initWithFormat:` (page 1572)
  Returns an *`NSString`* object initialized by using a given format string as a template into which the remaining argument values are substituted.
- `initWithFormat:arguments:` (page 1573)
  Returns an *`NSString`* object initialized by using a given format string as a template into which the remaining argument values are substituted according to the user's default locale.
- `initWithFormat:locale:` (page 1574)
  Returns an *`NSString`* object initialized by using a given format string as a template into which the remaining argument values are substituted according to given locale information.
- `initWithFormat:locale:arguments:` (page 1574)
  Returns an *`NSString`* object initialized by using a given format string as a template into which the remaining argument values are substituted according to given locale information.
- `initWithData:encoding:` (page 1572)
  Returns an `NSString` object initialized by converting given data into Unicode characters using a given encoding.
+ `stringWithFormat:` (page 1536)
  Returns a string created by using a given format string as a template into which the remaining argument values are substituted.
+ `localizedStringWithFormat:` (page 1528)
  Returns a string created by using a given format string as a template into which the remaining argument values are substituted according to the user's default locale.
+ `stringWithCharacters:length:` (page 1530)
  Returns a string containing a given number of characters taken from a given C array of Unicode characters.
+ `stringWithString:` (page 1537)
  Returns a string created by copying the characters from another given string.
+ `stringWithCString:encoding:` (page 1535)
  Returns a string containing the bytes in a given C array, interpreted according to a given encoding.
+ `stringWithUTF8String:` (page 1537)
  Returns a string created by copying the data from a given C array of UTF8-encoded bytes.
+ `stringWithCString:` (page 1534) Deprecated in Mac OS X v10.4
  Creates a new string using a given C-string. (Deprecated. Use `stringWithCString:encoding:` (page 1535) instead.)

+ stringWithCString:length: (page 1535) Deprecated in Mac OS X v10.4
Returns a string containing the characters in a given C-string. (Deprecated. Use stringWithCString:encoding: (page 1535) instead.)

- initWithCString: (page 1569) Deprecated in Mac OS X v10.4
Initializes the receiver, a newly allocated NSString object, by converting the data in a given C-string from the default C-string encoding into the Unicode character encoding. (Deprecated. Use initWithCString:encoding: (page 1570) instead.)

- initWithCString:length: (page 1570) Deprecated in Mac OS X v10.4
Initializes the receiver, a newly allocated NSString object, by converting the data in a given C-string from the default C-string encoding into the Unicode character encoding. (Deprecated. Use initWithCString:encoding: (page 1570) instead.)

- initWithCStringNoCopy:length:freeWhenDone: (page 1571) Deprecated in Mac OS X v10.4
Initializes the receiver, a newly allocated NSString object, by converting the data in a given C-string from the default C-string encoding into the Unicode character encoding. (Deprecated. Use initWithBytesNoCopy:length:encoding:freeWhenDone: (page 1564) instead.)

Creating and Initializing a String from a File

+ stringWithContentsOfFile:encoding:error: (page 1531)
Returns a string created by reading data from the file at a given path interpreted using a given encoding.

- initWithContentsOfFile:encoding:error: (page 1566)
Returns an NSString object initialized by reading data from the file at a given path using a given encoding.

+ stringWithContentsOfFile:usedEncoding:error: (page 1532)
Returns a string created by reading data from the file at a given path and returns by reference the encoding used to interpret the file.

- initWithContentsOfFile:usedEncoding:error: (page 1567)
Returns an NSString object initialized by reading data from the file at a given path and returns by reference the encoding used to interpret the characters.

+ stringWithContentsOfFile: (page 1530) Deprecated in Mac OS X v10.4
Returns a string created by reading data from the file named by a given path. (Deprecated. Use stringWithContentsOfFile:encoding:error: (page 1531) or stringWithContentsOfFile:usedEncoding:error: (page 1532) instead.)

- initWithContentsOfFile: (page 1566) Deprecated in Mac OS X v10.4
Initializes the receiver, a newly allocated NSString object, by reading data from the file named by *path*. (Deprecated. Use initWithContentsOfFile:encoding:error: (page 1566) or initWithContentsOfFile:usedEncoding:error: (page 1567) instead.)

Creating and Initializing a String from an URL

+ stringWithContentsOfURL:encoding:error: (page 1533)
Returns a string created by reading data from a given URL interpreted using a given encoding.

- initWithContentsOfURL:encoding:error: (page 1568)
Returns an NSString object initialized by reading data from a given URL interpreted using a given encoding.

+ stringWithContentsOfURL:usedEncoding:error: (page 1534)
Returns a string created by reading data from a given URL and returns by reference the encoding used to interpret the data.

- initWithContentsOfURL:usedEncoding:error: (page 1569)
Returns an NSString object initialized by reading data from a given URL and returns by reference the encoding used to interpret the data.

+ stringWithContentsOfURL: (page 1532) Deprecated in Mac OS X v10.4
Returns a string created by reading data from the file named by a given URL. (Deprecated. Use stringWithContentsOfURL:encoding:error: (page 1533) or stringWithContentsOfURL:usedEncoding:error: (page 1534) instead.)

- initWithContentsOfURL: (page 1568) Deprecated in Mac OS X v10.4
Initializes the receiver, a newly allocated NSString object, by reading data from the location named by a given URL. (Deprecated. Use initWithContentsOfURL:encoding:error: (page 1568) or initWithContentsOfURL:usedEncoding:error: (page 1569) instead.)

Writing to a File or URL

- writeToFile:atomically:encoding:error: (page 1613)
Writes the contents of the receiver to a file at a given path using a given encoding.

- writeToURL:atomically:encoding:error: (page 1614)
Writes the contents of the receiver to the URL specified by *url* using the specified encoding.

- writeToFile:atomically: (page 1612) Deprecated in Mac OS X v10.4
Writes the contents of the receiver to the file specified by a given path. (Deprecated. Use writeToFile:atomically:encoding:error: (page 1613) instead.)

- writeToURL:atomically: (page 1614) Deprecated in Mac OS X v10.4
Writes the contents of the receiver to the location specified by a given URL. (Deprecated. Use writeToURL:atomically:encoding:error: (page 1614) instead.)

Getting a String's Length

- length (page 1580)
Returns the number of Unicode characters in the receiver.

- lengthOfBytesUsingEncoding: (page 1580)
Returns the number of bytes required to store the receiver in a given encoding.

- maximumLengthOfBytesUsingEncoding: (page 1584)
Returns the maximum number of bytes needed to store the receiver in a given encoding.

Getting Characters and Bytes

- characterAtIndex: (page 1540)
Returns the character at a given array position.

- getCharacters: (page 1555)
Copies all characters from the receiver into a given buffer.

- `getCharacters:range:` (page 1555)
  Copies characters from a given range in the receiver into a given buffer.
- `getBytes:maxLength:usedLength:encoding:options:range:remainingRange:` (page 1554)
  Gets a given range of characters as bytes in a specified encoding.

Getting C Strings

- `cStringUsingEncoding:` (page 1549)
  Returns a representation of the receiver as a C string using a given encoding.
- `getCString:maxLength:encoding:` (page 1557)
  Converts the receiver's content to a given encoding and stores them in a buffer.
- `UTF8String` (page 1612)
  Returns a null-terminated UTF8 representation of the receiver.
- `cString` (page 1547) Deprecated in Mac OS X v10.4
  Returns a representation of the receiver as a C string in the default C-string encoding. (Deprecated. Use `cStringUsingEncoding:` (page 1549) or `UTF8String` (page 1612) instead.)
- `cStringLength` (page 1548) Deprecated in Mac OS X v10.4
  Returns the length in `char`-sized units of the receiver's C-string representation in the default C-string encoding. (Deprecated. Use lengthOfBytesUsingEncoding: (page 1580) or maximumLengthOfBytesUsingEncoding: (page 1584) instead.)
- `getCString:` (page 1556) Deprecated in Mac OS X v10.4
  Invokes `getCString:maxLength:range:remainingRange:` (page 1558) with `NSMaximumStringLength` as the maximum length, the receiver's entire extent as the range, and `NULL` for the remaining range. (Deprecated. Use cStringUsingEncoding: (page 1549) or `dataUsingEncoding:allowLossyConversion:` (page 1550) instead.)
- `getCString:maxLength:` (page 1557) Deprecated in Mac OS X v10.4
  Invokes `getCString:maxLength:range:remainingRange:` (page 1558) with *`maxLength`* as the maximum length in `char`-sized units, the receiver's entire extent as the range, and `NULL` for the remaining range. (Deprecated. Use getCString:maxLength:encoding: (page 1557) instead.)
- `getCString:maxLength:range:remainingRange:` (page 1558) Deprecated in Mac OS X v10.4
  Converts the receiver's content to the default C-string encoding and stores them in a given buffer. (Deprecated. Use getCString:maxLength:encoding: (page 1557) instead.)
- `lossyCString` (page 1583) Deprecated in Mac OS X v10.4
  Returns a representation of the receiver as a C string in the default C-string encoding, possibly losing information in converting to that encoding. (Deprecated. Use cStringUsingEncoding: (page 1549) or `dataUsingEncoding:allowLossyConversion:` (page 1550) instead.)

Combining Strings

- `stringByAppendingFormat:` (page 1597)
  Returns a string made by appending to the receiver a string constructed from a given format string and the following arguments.
- `stringByAppendingString:` (page 1599)
  Returns a new string made by appending a given string to the receiver.

- `stringByPaddingToLength:withString:startingAtIndex:` (page 1603)
  Returns a new string formed from the receiver by either removing characters from the end, or by appending as many occurrences as necessary of a given pad string.

Dividing Strings

- `componentsSeparatedByString:` (page 1547)
  Returns an array containing substrings from the receiver that have been divided by a given separator.
- `componentsSeparatedByCharactersInSet:` (page 1546)
  Returns an array containing substrings from the receiver that have been divided by characters in a given set.
- `stringByTrimmingCharactersInSet:` (page 1608)
  Returns a new string made by removing from both ends of the receiver characters contained in a given character set.
- `substringFromIndex:` (page 1609)
  Returns a new string containing the characters of the receiver from the one at a given index to the end.
- `substringWithRange:` (page 1611)
  Returns a string object containing the characters of the receiver that lie within a given range.
- `substringToIndex:` (page 1610)
  Returns a new string containing the characters of the receiver up to, but not including, the one at a given index.

Finding Characters and Substrings

- `rangeOfCharacterFromSet:` (page 1589)
  Finds and returns the range in the receiver of the first character from a given character set.
- `rangeOfCharacterFromSet:options:` (page 1590)
  Finds and returns the range in the receiver of the first character, using given options, from a given character set.
- `rangeOfCharacterFromSet:options:range:` (page 1590)
  Finds and returns the range in the receiver of the first character from a given character set found in a given range with given options.
- `rangeOfString:` (page 1592)
  Finds and returns the range of the first occurrence of a given string within the receiver.
- `rangeOfString:options:` (page 1593)
  Finds and returns the range of the first occurrence of a given string within the receiver, subject to given options.
- `rangeOfString:options:range:` (page 1594)
  Finds and returns the range of the first occurrence of a given string, within the given range of the receiver, subject to given options.
- `rangeOfString:options:range:locale:` (page 1595)
  Finds and returns the range of the first occurrence of a given string within a given range of the receiver, subject to given options, using the specified locale, if any.

Replacing Substrings

- `stringByReplacingOccurrencesOfString:withString:` (page 1605)
  Returns a new string in which all occurrences of a target string in the receiver are replaced by another given string.
- `stringByReplacingOccurrencesOfString:withString:options:range:` (page 1605)
  Returns a new string in which all occurrences of a target string in a specified range of the receiver are replaced by another given string.
- `stringByReplacingCharactersInRange:withString:` (page 1604)
  Returns a new string in which the characters in a specified range of the receiver are replaced by a given string.

Determining Line and Paragraph Ranges

- `getLineStart:end:contentsEnd:forRange:` (page 1560)
  Returns by reference the beginning of the first line and the end of the last line touched by the given range.
- `lineRangeForRange:` (page 1581)
  Returns the range of characters representing the line or lines containing a given range.
- `getParagraphStart:end:contentsEnd:forRange:` (page 1561)
  Returns by reference the beginning of the first paragraph and the end of the last paragraph touched by the given range.
- `paragraphRangeForRange:` (page 1585)
  Returns the range of characters representing the paragraph or paragraphs containing a given range.

Determining Composed Character Sequences

- `rangeOfComposedCharacterSequenceAtIndex:` (page 1591)
  Returns the range in the receiver of the composed character sequence located at a given index.
- `rangeOfComposedCharacterSequencesForRange:` (page 1592)
  Returns the range in the receiver of the composed character sequence in a given range.

Converting String Contents Into a Property List

- `propertyList` (page 1588)
  Parses the receiver as a text representation of a property list, returning an `NSString`, `NSData`, `NSArray`, or `NSDictionary` object, according to the topmost element.
- `propertyListFromStringsFileFormat` (page 1588)
  Returns a dictionary object initialized with the keys and values found in the receiver.

Identifying and Comparing Strings

- `caseInsensitiveCompare:` (page 1540)
  Returns the result of invoking `compare:options:` (page 1542) with `NSCaseInsensitiveSearch` as the only option.
- `localizedCaseInsensitiveCompare:` (page 1582)
  Returns an `NSComparisonResult` value that indicates the lexical ordering of the receiver and a given string using a case-insensitive, localized, comparison.
- `compare:` (page 1542)
  Returns the result of invoking `compare:options:range:` (page 1543) with no options and the receiver's full extent as the range.
- `localizedCompare:` (page 1582)
  Returns an `NSComparisonResult` value that indicates the lexical ordering of the receiver and another given string using a localized comparison.
- `compare:options:` (page 1542)
  Returns the result of invoking `compare:options:range:` (page 1543) with a given mask as the options and the receiver's full extent as the range.
- `compare:options:range:` (page 1543)
  Returns the result of invoking `compare:options:range:locale:` (page 1544) with a `nil` locale.
- `compare:options:range:locale:` (page 1544)
  Returns an `NSComparisonResult` value that indicates the lexical ordering of a specified range within the receiver and a given string.
- `hasPrefix:` (page 1562)
  Returns a Boolean value that indicates whether a given string matches the beginning characters of the receiver.
- `hasSuffix:` (page 1562)
  Returns a Boolean value that indicates whether a given string matches the ending characters of the receiver.
- `isEqualToString:` (page 1578)
  Returns a Boolean value that indicates whether a given string is equal to the receiver using an literal Unicode-based comparison.
- `hash` (page 1561)
  Returns an unsigned integer that can be used as a hash table address.

Folding Strings

- `stringByFoldingWithOptions:locale:` (page 1603)
  Returns a string with the given character folding options applied.

Getting a Shared Prefix

- `commonPrefixWithString:options:` (page 1541)
  Returns a string containing prefix the receiver and a given string have in common.

Changing Case

- `capitalizedString` (page 1539)
  Returns a capitalized representation of the receiver.
- `lowercaseString` (page 1584)
  Returns lowercased representation of the receiver.
- `uppercaseString` (page 1611)
  Returns an uppercased representation of the receiver.

Getting Strings with Mapping

- `decomposedStringWithCanonicalMapping` (page 1551)
  Returns a string made by normalizing the receiver's contents using Form D.
- `decomposedStringWithCompatibilityMapping` (page 1551)
  Returns a string made by normalizing the receiver's contents using Form KD.
- `precomposedStringWithCanonicalMapping` (page 1587)
  Returns a string made by normalizing the receiver's contents using Form C.
- `precomposedStringWithCompatibilityMapping` (page 1588)
  Returns a string made by normalizing the receiver's contents using Form KC.

Getting Numeric Values

- `doubleValue` (page 1552)
  Returns the floating-point value of the receiver's text as a `double`.
- `floatValue` (page 1553)
  Returns the floating-point value of the receiver's text as a `float`.
- `intValue` (page 1577)
  Returns the integer value of the receiver's text.
- `integerValue` (page 1576)
  Returns the `NSInteger` value of the receiver's text.
- `longLongValue` (page 1583)
  Returns the `long long` value of the receiver's text.
- `boolValue` (page 1538)
  Returns the Boolean value of the receiver's text.

Working with Encodings

+ `availableStringEncodings` (page 1526)
  Returns a zero-terminated list of the encodings string objects support in the application's environment.
+ `defaultCStringEncoding` (page 1527)
  Returns the C-string encoding assumed for any method accepting a C string as an argument.
+ `localizedNameOfStringEncoding:` (page 1527)
  Returns a human-readable string giving the name of a given encoding.

- `canBeConvertedToEncoding:` (page 1539)
  Returns a Boolean value that indicates whether the receiver can be converted to a given encoding without loss of information.
- `dataUsingEncoding:` (page 1549)
  Returns an `NSData` object containing a representation of the receiver encoded using a given encoding.
- `dataUsingEncoding:allowLossyConversion:` (page 1550)
  Returns an `NSData` object containing a representation of the receiver encoded using a given encoding.
- `description` (page 1551)
  Returns the receiver.
- `fastestEncoding` (page 1552)
  Returns the fastest encoding to which the receiver may be converted without loss of information.
- `smallestEncoding` (page 1595)
  Returns the smallest encoding to which the receiver can be converted without loss of information.

Working with Paths

+ `pathWithComponents:` (page 1529)
  Returns a string built from the strings in a given array by concatenating them with a path separator between each pair.
- `pathComponents` (page 1585)
  Returns an array of `NSString` objects containing, in order, each path component of the receiver.
- `completePathIntoString:caseSensitive:matchesIntoArray:filterTypes:` (page 1545)
  Interprets the receiver as a path in the file system and attempts to perform filename completion, returning a numeric value that indicates whether a match was possible, and by reference the longest path that matches the receiver.
- `fileSystemRepresentation` (page 1553)
  Returns a file system-specific representation of the receiver.
- `getFileSystemRepresentation:maxLength:` (page 1559)
  Interprets the receiver as a system-independent path and fills a buffer with a C-string in a format and encoding suitable for use with file-system calls.
- `isAbsolutePath` (page 1578)
  Returning a Boolean value that indicates whether the receiver represents an absolute path.
- `lastPathComponent` (page 1579)
  Returns the last path component of the receiver.
- `pathExtension` (page 1586)
  Interprets the receiver as a path and returns the receiver's extension, if any.
- `stringByAbbreviatingWithTildeInPath` (page 1596)
  Returns a new string representing the receiver as a path with a tilde (~) substituted for the full path to the current user's home directory.
- `stringByAppendingPathComponent:` (page 1598)
  Returns a new string made by appending to the receiver a given string.
- `stringByAppendingPathExtension:` (page 1598)
  Returns a new string made by appending to the receiver an extension separator followed by a given extension.

- `stringByDeletingLastPathComponent` (page 1600)
  Returns a new string made by deleting the last path component from the receiver, along with any final path separator.
- `stringByDeletingPathExtension` (page 1601)
  Returns a new string made by deleting the extension (if any, and only the last) from the receiver.
- `stringByExpandingTildeInPath` (page 1602)
  Returns a new string made by expanding the initial component of the receiver to its full path value.
- `stringByResolvingSymlinksInPath` (page 1606)
  Returns a new string made from the receiver by resolving all symbolic links and standardizing path.
- `stringByStandardizingPath` (page 1607)
  Returns a new string made by removing extraneous path components from the receiver.
- `stringsByAppendingPaths:` (page 1609)
  Returns an array of strings made by separately appending to the receiver each string in in a given array.

Working with URLs

- `stringByAddingPercentEscapesUsingEncoding:` (page 1596)
  Returns a representation of the receiver using a given encoding to determine the percent escapes necessary to convert the receiver into a legal URL string.
- `stringByReplacingPercentEscapesUsingEncoding:` (page 1606)
  Returns a new string made by replacing in the receiver all percent escapes with the matching characters as determined by a given encoding.

Class Methods

availableStringEncodings

Returns a zero-terminated list of the encodings string objects support in the application's environment.

```
+ (const NSStringEncoding *)availableStringEncodings
```

Return Value
A zero-terminated list of the encodings string objects support in the application's environment.

Discussion
Among the more commonly used encodings are:

```
NSASCIIStringEncoding
NSUnicodeStringEncoding
NSISOLatin1StringEncoding
NSISOLatin2StringEncoding
NSSymbolStringEncoding
```

See the "Constants" (page 1615) section for a larger list and descriptions of many supported encodings. In addition to those encodings listed here, you can also use the encodings defined for CFString in Core Foundation; you just need to call the `CFStringConvertEncodingToNSStringEncoding` function to convert them to a usable format.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ localizedNameOfStringEncoding:` (page 1527)

Declared In
`NSString.h`

defaultCStringEncoding

Returns the C-string encoding assumed for any method accepting a C string as an argument.

```
+ (NSStringEncoding)defaultCStringEncoding
```

Return Value
The C-string encoding assumed for any method accepting a C string as an argument.

Discussion
This method returns a user-dependent encoding who value is derived from user's default language and potentially other factors. You might sometimes need to use this encoding when interpreting user documents with unknown encodings, in the absence of other hints, but in general this encoding should be used rarely, if at all. Note that some potential values might result in unexpected encoding conversions of even fairly straightforward `NSString` content—for example, punctuation characters with a bidirectional encoding.

Methods that accept a C string as an argument use `...CString...` in the keywords for such arguments: for example, `stringWithCString:` (page 1534)—note, though, that these are deprecated. The default C-string encoding is determined from system information and can't be changed programmatically for an individual process. See "String Encodings" (page 1619) for a full list of supported encodings.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTKitCreateMovie
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSString.h`

localizedNameOfStringEncoding:

Returns a human-readable string giving the name of a given encoding.

```
+ (NSString *)localizedNameOfStringEncoding:(NSStringEncoding)encoding
```

Parameters

encoding

A string encoding.

Return Value

A human-readable string giving the name of *encoding* in the current locale's language.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

NSFontAttributeExplorer

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In

`NSString.h`

localizedStringWithFormat:

Returns a string created by using a given format string as a template into which the remaining argument values are substituted according to the user's default locale.

```
+ (id)localizedStringWithFormat:(NSString *)format ...
```

Parameters

format

A format string. See Formatting String Objects for examples of how to use this method, and String Format Specifiers for a list of format specifiers. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *format* is `nil`.

...

A comma-separated list of arguments to substitute into *format*.

Return Value

A string created by using *format* as a template into which the following argument values are substituted according to the formatting information to the user's default locale.

Discussion

This method is equivalent to using `initWithFormat:locale:` (page 1574) and passing `[[NSUserDefaults standardUserDefaults] dictionaryRepresentation]` as the locale argument.

As an example of formatting, this method replaces the decimal according to the locale in `%f` and `%d` substitutions, and calls `descriptionWithLocale:` instead of `description` where necessary.

This code excerpt creates a string from another string and a `float`:

```
NSString *myString = [NSString localizedStringWithFormat:@"%@:  %f\n", @"Cost",
 1234.56];
```

The resulting string has the value "`Cost: 1234.560000\n`" if the locale is `en_US`, and "`Cost: 1234,560000\n`" if the locale is `fr_FR`.

See Formatting String Objects for more information.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ stringWithFormat:` (page 1536)
`- initWithFormat:locale:` (page 1574)

Related Sample Code
FilterDemo
GridCalendar

Declared In
`NSString.h`

pathWithComponents:

Returns a string built from the strings in a given array by concatenating them with a path separator between each pair.

```
+ (NSString *)pathWithComponents:(NSArray *)components
```

Parameters

components

An array of `NSString` objects representing a file path. To create an absolute path, use a slash mark ("/") as the first component. To include a trailing path divider, use an empty string as the last component.

Return Value
A string built from the strings in *components* by concatenating them (in the order they appear in the array) with a path separator between each pair.

Discussion
This method doesn't clean up the path created; use `stringByStandardizingPath` (page 1607) to resolve empty components, references to the parent directory, and so on.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- pathComponents` (page 1585)

Declared In
`NSPathUtilities.h`

string

Returns an empty string.

```
+ (id)string
```

Return Value
An empty string.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `init` (page 1563)

Declared In
`NSString.h`

stringWithCharacters:length:

Returns a string containing a given number of characters taken from a given C array of Unicode characters.

```
+ (id)stringWithCharacters:(const unichar *)chars length:(NSUInteger)length
```

Parameters

chars

A C array of Unicode characters; the value must not be `NULL`.

Important: Raises an exception if *chars* is `NULL`, even if *length* is `0`.

length

The number of characters to use from *chars*.

Return Value
A string containing *length* Unicode characters taken (starting with the first) from *chars*.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `initWithCharacters:length:` (page 1565)

Related Sample Code
CrossEvents
PDFKitLinker2
QCCocoaComponent
SharedMemory

Declared In
`NSString.h`

stringWithContentsOfFile:

Returns a string created by reading data from the file named by a given path. (Deprecated in Mac OS X v10.4. Use `stringWithContentsOfFile:encoding:error:` (page 1531) or stringWithContentsOfFile:usedEncoding:error: (page 1532) instead.)

```
+ (id)stringWithContentsOfFile:(NSString *)path
```

Discussion
If the contents begin with a Unicode byte-order mark (U+FEFF or U+FFFE), interprets the contents as Unicode characters. If the contents begin with a UTF-8 byte-order mark (EFBBBF), interprets the contents as UTF-8. Otherwise, interprets the contents as data in the default C string encoding. Since the default C string encoding will vary with the user's configuration, do not depend on this method unless you are using Unicode or UTF-8 or you can verify the default C string encoding. Returns `nil` if the file can't be opened.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
`+ stringWithContentsOfFile:encoding:error:` (page 1531)
`+ stringWithContentsOfFile:usedEncoding:error:` (page 1532)

Related Sample Code
CIAnnotation
GLSLShowpiece
NURBSSurfaceVertexProg
SpecialPictureProtocol
SurfaceVertexProgram

Declared In
`NSString.h`

stringWithContentsOfFile:encoding:error:

Returns a string created by reading data from the file at a given path interpreted using a given encoding.

```
+ (id)stringWithContentsOfFile:(NSString *)path encoding:(NSStringEncoding)enc
    error:(NSError **)error
```

Parameters

path
A path to a file.

enc
The encoding of the file at *path*.

error
If an error occurs, upon returns contains an `NSError` object that describes the problem. If you are not interested in possible errors, pass in `NULL`.

Return Value
A string created by reading data from the file named by *path* using the encoding, *enc*. If the file can't be opened or there is an encoding error, returns `nil`.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- initWithContentsOfFile:encoding:error:` (page 1566)

Related Sample Code
JSPong
LSMSmartCategorizer

Declared In
NSString.h

stringWithContentsOfFile:usedEncoding:error:

Returns a string created by reading data from the file at a given path and returns by reference the encoding used to interpret the file.

```
+ (id)stringWithContentsOfFile:(NSString *)path usedEncoding:(NSStringEncoding
    *)enc error:(NSError **)error
```

Parameters

path
A path to a file.

enc
Upon return, if the file is read successfully, contains the encoding used to interpret the file at *path*.

error
If an error occurs, upon returns contains an NSError object that describes the problem. If you are not interested in possible errors, you may pass in NULL.

Return Value
A string created by reading data from the file named by *path*. If the file can't be opened or there is an encoding error, returns nil.

Discussion
This method attempts to determine the encoding of the file at *path*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- initWithContentsOfFile:encoding:error: (page 1566)

Declared In
NSString.h

stringWithContentsOfURL:

Returns a string created by reading data from the file named by a given URL. (Deprecated in Mac OS X v10.4. Use stringWithContentsOfURL:encoding:error: (page 1533) or stringWithContentsOfURL:usedEncoding:error: (page 1534) instead.)

```
+ (id)stringWithContentsOfURL:(NSURL *)aURL
```

Discussion
If the contents begin with a byte-order mark (U+FEFF or U+FFFE), interprets the contents as Unicode characters. If the contents begin with a UTF-8 byte-order mark (EFBBBF), interprets the contents as UTF-8. Otherwise interprets the contents as data in the default C string encoding. Since the default C string encoding will vary with the user's configuration, do not depend on this method unless you are using Unicode or UTF-8 or you can verify the default C string encoding. Returns `nil` if the location can't be opened.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
`+ stringWithContentsOfURL:encoding:error:` (page 1533)
`+ stringWithContentsOfURL:usedEncoding:error:` (page 1534)

Declared In
`NSString.h`

stringWithContentsOfURL:encoding:error:

Returns a string created by reading data from a given URL interpreted using a given encoding.

```
+ (id)stringWithContentsOfURL:(NSURL *)url encoding:(NSStringEncoding)enc
    error:(NSError **)error
```

Parameters

url
The URL to read.

enc
The encoding of the data at *url*.

error
If an error occurs, upon returns contains an `NSError` object that describes the problem. If you are not interested in possible errors, you may pass in `NULL`.

Return Value
A string created by reading data from *URL* using the encoding, *enc*. If the URL can't be opened or there is an encoding error, returns `nil`.

Availability
Available in Mac OS X v10.4 and later.

See Also
`+ stringWithContentsOfURL:usedEncoding:error:` (page 1534)
`- initWithContentsOfURL:encoding:error:` (page 1568)

Declared In
`NSString.h`

stringWithContentsOfURL:usedEncoding:error:

Returns a string created by reading data from a given URL and returns by reference the encoding used to interpret the data.

```
+ (id)stringWithContentsOfURL:(NSURL *)url usedEncoding:(NSStringEncoding *)enc
    error:(NSError **)error
```

Parameters

url

The URL from which to read data.

enc

Upon return, if *url* is read successfully, contains the encoding used to interpret the data.

error

If an error occurs, upon returns contains an `NSError` object that describes the problem. If you are not interested in possible errors, you may pass in `NULL`.

Return Value

A string created by reading data from *url*. If the URL can't be opened or there is an encoding error, returns `nil`.

Discussion

This method attempts to determine the encoding at *url*.

Availability

Available in Mac OS X v10.4 and later.

See Also

`+ stringWithContentsOfURL:encoding:error:` (page 1533)

`- initWithContentsOfURL:usedEncoding:error:` (page 1569)

Declared In

`NSString.h`

stringWithCString:

Creates a new string using a given C-string. (Deprecated in Mac OS X v10.4. Use `stringWithCString:encoding:` (page 1535) instead.)

```
+ (id)stringWithCString:(const char *)cString
```

Discussion

cString should contain data in the default C string encoding. If the argument passed to `stringWithCString:` is not a zero-terminated C-string, the results are undefined.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

See Also

`+ stringWithCString:encoding:` (page 1535)

Related Sample Code

Quartz EB

Simon
SurfaceVertexProgram
Vertex Optimization
Video Hardware Info

Declared In
NSString.h

stringWithCString:encoding:

Returns a string containing the bytes in a given C array, interpreted according to a given encoding.

```
+ (id)stringWithCString:(const char *)cString encoding:(NSStringEncoding)enc
```

Parameters

cString
A C array of bytes. The array must end with a NULL character; intermediate NULL characters are not allowed.

enc
The encoding of *cString*.

Return Value
A string containing the characters described in *cString*.

Discussion
If *cString* is not a NULL-terminated C string, or *encoding* does not match the actual encoding, the results are undefined.

Availability
Available in Mac OS X v10.4 and later.

See Also
- initWithCString:encoding: (page 1570)

Related Sample Code
CAPlayThrough
QTKitCreateMovie
QTMetadataEditor
SMARTQuery
VideoHardwareInfo

Declared In
NSString.h

stringWithCString:length:

Returns a string containing the characters in a given C-string. (Deprecated in Mac OS X v10.4. Use stringWithCString:encoding: (page 1535) instead.)

```
+ (id)stringWithCString:(const char *)cString length:(NSUInteger)length
```

Discussion
cString must not be NULL. *cString* should contain characters in the default C-string encoding. This method converts *length* * `sizeof(char)` bytes from *cString* and doesn't stop short at a `NULL` character.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
`+ stringWithCString:encoding:` (page 1535)

Related Sample Code
CapabilitiesSample
CocoaSpeechSynthesisExample
EnhancedDataBurn
Fiendishthngs
SGDevices

Declared In
`NSString.h`

stringWithFormat:

Returns a string created by using a given format string as a template into which the remaining argument values are substituted.

```
+ (id)stringWithFormat:(NSString *)format, ...
```

Parameters

format

A format string. See Formatting String Objects for examples of how to use this method, and String Format Specifiers for a list of format specifiers. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *format* is `nil`.

`...`

A comma-separated list of arguments to substitute into *format*.

Return Value
A string created by using *format* as a template into which the remaining argument values are substituted according to the canonical locale.

Discussion
This method is similar to `localizedStringWithFormat:` (page 1528), but using the canonical locale to format numbers. This is useful, for example, if you want to produce "non-localized" formatting which needs to be written out to files and parsed back later.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- initWithFormat:` (page 1572)

+ localizedStringWithFormat: (page 1528)

Related Sample Code
CoreRecipes
Fiendishthngs
LSMSmartCategorizer
MyPhoto
Quartz Composer WWDC 2005 TextEdit

Declared In
NSString.h

stringWithString:

Returns a string created by copying the characters from another given string.

```
+ (id)stringWithString:(NSString *)aString
```

Parameters

aString

The string from which to copy characters. This value must not be nil.

Important: Raises an NSInvalidArgumentException if *aString* is nil.

Return Value
A string created by copying the characters from *aString*.

Availability
Available in Mac OS X v10.0 and later.

See Also
- initWithString: (page 1575)

Related Sample Code
OpenGL Screensaver
QTMetadataEditor
SimpleScriptingProperties
SurfaceVertexProgram
TimelineToTC

Declared In
NSString.h

stringWithUTF8String:

Returns a string created by copying the data from a given C array of UTF8-encoded bytes.

```
+ (id)stringWithUTF8String:(const char *)bytes
```

Parameters

bytes

A NULL-terminated C array of bytes in UTF8 encoding.

Important: Raises an exception if *bytes* is NULL.

Return Value
A string created by copying the data from *bytes*.

Availability
Available in Mac OS X v10.0 and later.

See Also
- initWithString: (page 1575)

Related Sample Code
DockTile
DynamicProperties
MyPhoto
QTMetadataEditor
StickiesExample

Declared In
NSString.h

Instance Methods

boolValue

Returns the Boolean value of the receiver's text.

```
- (BOOL)boolValue
```

Return Value
The Boolean value of the receiver's text. Returns YES on encountering one of "Y", "y", "T", "t", or a digit 1-9—the method ignores any trailing characters. Returns NO if the receiver doesn't begin with a valid decimal text representation of a number.

Discussion
The method assumes a decimal representation and skips whitespace at the beginning of the string. It also skips initial whitespace characters, or optional -/+ sign followed by zeroes.

Availability
Available in Mac OS X v10.5 and later.

See Also
- integerValue (page 1576)
- scanInt: (page 1353) (NSScanner)

Declared In
NSString.h

canBeConvertedToEncoding:

Returns a Boolean value that indicates whether the receiver can be converted to a given encoding without loss of information.

```
- (BOOL)canBeConvertedToEncoding:(NSStringEncoding)encoding
```

Parameters

encoding
A string encoding.

Return Value
YES if the receiver can be converted to *encoding* without loss of information. Returns NO if characters would have to be changed or deleted in the process of changing encodings.

Discussion
If you plan to actually convert a string, the dataUsingEncoding:... methods return nil on failure, so you can avoid the overhead of invoking this method yourself by simply trying to convert the string.

Availability
Available in Mac OS X v10.0 and later.

See Also
- dataUsingEncoding:allowLossyConversion: (page 1550)

Declared In
NSString.h

capitalizedString

Returns a capitalized representation of the receiver.

```
- (NSString *)capitalizedString
```

Return Value
A string with the first character from each word in the receiver changed to its corresponding uppercase value, and all remaining characters set to their corresponding lowercase values.

Discussion
A "word" here is any sequence of characters delimited by spaces, tabs, or line terminators (listed under getLineStart:end:contentsEnd:forRange: (page 1560)). Other common word delimiters such as hyphens and other punctuation aren't considered, so this method may not generally produce the desired results for multiword strings.

Case transformations aren't guaranteed to be symmetrical or to produce strings of the same lengths as the originals. See lowercaseString (page 1584) for an example.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `lowercaseString` (page 1584)
- `uppercaseString` (page 1611)

Related Sample Code
Mountains
StickiesExample

Declared In
`NSString.h`

caseInsensitiveCompare:

Returns the result of invoking `compare:options:` (page 1542) with `NSCaseInsensitiveSearch` as the only option.

```
- (NSComparisonResult)caseInsensitiveCompare:(NSString *)aString
```

Parameters

aString

The string with which to compare the receiver.

This value must not be `nil`. If this value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

Return Value
The result of invoking `compare:options:` (page 1542) with `NSCaseInsensitiveSearch` as the only option.

Discussion
If you are comparing strings to present to the end-user, you should typically use `localizedCaseInsensitiveCompare:` (page 1582) instead.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `localizedCaseInsensitiveCompare:` (page 1582)
- `compare:options:` (page 1542)

Related Sample Code
IdentitySample
People

Declared In
`NSString.h`

characterAtIndex:

Returns the character at a given array position.

```
- (unichar)characterAtIndex:(NSUInteger)index
```

Parameters

index

The index of the character to retrieve. The index value must not lie outside the bounds of the receiver.

Return Value

The character at the array position given by *index*.

Discussion

Raises an `NSRangeException` if *index* lies beyond the end of the receiver.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `getCharacters:` (page 1555)
- `getCharacters:range:` (page 1555)

Related Sample Code

CubePuzzle
DerivedProperty
NSGLImage
NSOpenGL Fullscreen
PDFKitLinker2

Declared In

`NSString.h`

commonPrefixWithString:options:

Returns a string containing prefix the receiver and a given string have in common.

```
- (NSString *)commonPrefixWithString:(NSString *)aString
    options:(NSStringCompareOptions)mask
```

Parameters

aString

The string with which to compare the receiver.

mask

Options for the comparison. The following search options may be specified by combining them with the C bitwise `OR` operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`. See *String Programming Guide for Cocoa* for details on these options.

Return Value

A string containing characters the receiver and *aString* have in common, starting from the beginning of each up to the first characters that aren't equivalent.

Discussion

The returned string is based on the characters of the receiver. For example, if the receiver is "Ma¨dchen" and *aString* is "Mädchenschule", the string returned is "Ma¨dchen", not "Mädchen".

Availability

Available in Mac OS X v10.0 and later.

See Also
- `hasPrefix:` (page 1562)

Declared In
`NSString.h`

compare:

Returns the result of invoking `compare:options:range:` (page 1543) with no options and the receiver's full extent as the range.

```
- (NSComparisonResult)compare:(NSString *)aString
```

Parameters

aString

The string with which to compare the receiver.

This value must not be `nil`. If this value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

Return Value
The result of invoking `compare:options:range:` (page 1543) with no options and the receiver's full extent as the range.

Discussion
If you are comparing strings to present to the end-user, you should typically use `localizedCompare:` (page 1582) or `localizedCaseInsensitiveCompare:` (page 1582) instead.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `localizedCompare:` (page 1582)
- `localizedCaseInsensitiveCompare:` (page 1582)
- `compare:options:` (page 1542)
- `caseInsensitiveCompare:` (page 1540)
- `isEqualToString:` (page 1578)

Related Sample Code
QTCoreVideo102
QTCoreVideo103
QTCoreVideo201
QTCoreVideo202
QTCoreVideo301

Declared In
`NSString.h`

compare:options:

Returns the result of invoking `compare:options:range:` (page 1543) with a given mask as the options and the receiver's full extent as the range.

```
- (NSComparisonResult)compare:(NSString *)aString
    options:(NSStringCompareOptions)mask
```

Parameters

aString

The string with which to compare the receiver.

This value must not be `nil`. If this value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

mask

Options for the search—you can combine any of the following using a C bitwise OR operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSNumericSearch`. See *String Programming Guide for Cocoa* for details on these options.

Return Value

The result of invoking `compare:options:range:` (page 1543) with a given mask as the options and the receiver's full extent as the range.

Discussion

If you are comparing strings to present to the end-user, you should typically use `localizedCompare:` (page 1582) or `localizedCaseInsensitiveCompare:` (page 1582) instead, or use `compare:options:range:locale:` (page 1544) and pass the user's locale.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `localizedCompare:` (page 1582)
- `localizedCaseInsensitiveCompare:` (page 1582)
- `compare:options:range:locale:` (page 1544)
- `caseInsensitiveCompare:` (page 1540)
- `isEqualToString:` (page 1578)

Declared In

`NSString.h`

compare:options:range:

Returns the result of invoking `compare:options:range:locale:` (page 1544) with a `nil` locale.

```
- (NSComparisonResult)compare:(NSString *)aString
    options:(NSStringCompareOptions)mask range:(NSRange)range
```

Parameters

aString

The string with which to compare the range of the receiver specified by *range*.

This value must not be `nil`. If this value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

mask

Options for the search—you can combine any of the following using a C bitwise OR operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSNumericSearch`.

See *String Programming Guide for Cocoa* for details on these options.

range

The range of the receiver over which to perform the comparison. The range must not exceed the bounds of the receiver.

Important: Raises an `NSRangeException` if *range* exceeds the bounds of the receiver.

Return Value
The result of invoking `compare:options:range:locale:` (page 1544) with a `nil` locale.

Discussion
If you are comparing strings to present to the end-user, you should typically use `compare:options:range:locale:` (page 1544) instead and pass the user's locale (`currentLocale` (page 821) [`NSLocale`]).

Availability
Available in Mac OS X v10.0 and later.

See Also
- `localizedCompare:` (page 1582)
- `localizedCaseInsensitiveCompare:` (page 1582)
- `compare:options:` (page 1542)
- `caseInsensitiveCompare:` (page 1540)
- `isEqualToString:` (page 1578)

Declared In
`NSString.h`

compare:options:range:locale:

Returns an `NSComparisonResult` value that indicates the lexical ordering of a specified range within the receiver and a given string.

```
- (NSComparisonResult)compare:(NSString *)aString
    options:(NSStringCompareOptions)mask range:(NSRange)range locale:(id)locale
```

Parameters

aString

The string with which to compare the range of the receiver specified by *range*.

This value must not be `nil`. If this value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

mask

Options for the search—you can combine any of the following using a C bitwise OR operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSNumericSearch`.

See *String Programming Guide for Cocoa* for details on these options.

range

The range of the receiver over which to perform the comparison. The range must not exceed the bounds of the receiver.

Important: Raises an `NSRangeException` if *range* exceeds the bounds of the receiver.

locale

An instance of `NSLocale`. If this value not `nil` and is not an instance of `NSLocale`, uses the current locale instead. If you are comparing strings to present to the end-user, you should typically pass the user's locale (`currentLocale` (page 821) [`NSLocale`]).

The locale argument affects both equality and ordering algorithms. For example, in some locales, accented characters are ordered immediately after the base; other locales order them after "z".

Return Value

`NSOrderedAscending` if the substring of the receiver given by *range* precedes *aString* in lexical ordering for the locale given in *dict*, `NSOrderedSame` if the substring of the receiver and *aString* are equivalent in lexical value, and `NSOrderedDescending` if the substring of the receiver follows *aString*.

Special Considerations

Prior to Mac OS X v10.5, the *locale* argument was an instance of `NSDictionary`. On Mac OS X v10.5 and later, if you pass an instance of `NSDictionary` the current locale is used instead.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `localizedCompare:` (page 1582)
- `localizedCaseInsensitiveCompare:` (page 1582)
- `caseInsensitiveCompare:` (page 1540)
- `compare:` (page 1542)
- `compare:options:` (page 1542)
- `compare:options:range:` (page 1543)
- `isEqualToString:` (page 1578)

Declared In

`NSString.h`

completePathIntoString:caseSensitive:matchesIntoArray:filterTypes:

Interprets the receiver as a path in the file system and attempts to perform filename completion, returning a numeric value that indicates whether a match was possible, and by reference the longest path that matches the receiver.

```
- (NSUInteger)completePathIntoString:(NSString **)outputName caseSensitive:(BOOL)flag
    matchesIntoArray:(NSArray **)outputArray filterTypes:(NSArray *)filterTypes
```

Parameters

outputName

Upon return, contains the longest path that matches the receiver.

flag
If YES, the methods considers case for possible completions.

outputArray
Upon return, contains all matching filenames.

filterTypes
An array of NSString objects specifying path extensions to consider for completion. only paths whose extensions (not including the extension separator) match one of those strings.

Return Value
0 if no matches are found and 1 if exactly one match is found. In the case of multiple matches, returns the actual number of matching paths if *outputArray* is provided, or simply a positive value if *outputArray* is NULL.

Discussion
You can check for the existence of matches without retrieving by passing NULL as *outputArray*.

Note that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSPathUtilities.h

componentsSeparatedByCharactersInSet:

Returns an array containing substrings from the receiver that have been divided by characters in a given set.

```
- (NSArray *)componentsSeparatedByCharactersInSet:(NSCharacterSet *)separator
```

Parameters
separator
A character set containing the characters to to use to split the receiver. Must not be nil.

Return Value
An NSArray object containing substrings from the receiver that have been divided by characters in *separator*.

Discussion
The substrings in the array appear in the order they did in the receiver. Adjacent occurrences of the separator characters produce empty strings in the result. Similarly, if the string begins or ends with separator characters, the first or last substring, respectively, is empty.

Availability
Available in Mac OS X v10.5 and later.

See Also
- componentsSeparatedByString: (page 1547)
- stringByTrimmingCharactersInSet: (page 1608)

Declared In
NSString.h

componentsSeparatedByString:

Returns an array containing substrings from the receiver that have been divided by a given separator.

```
- (NSArray *)componentsSeparatedByString:(NSString *)separator
```

Parameters

separator

The separator string.

Return Value

An `NSArray` object containing substrings from the receiver that have been divided by *separator*.

Discussion

The substrings in the array appear in the order they did in the receiver. Adjacent occurrences of the separator string produce empty strings in the result. Similarly, if the string begins or ends with the separator, the first or last substring, respectively, is empty. For example, this code fragment:

```
NSString *list = @"Norman, Stanley, Fletcher";
NSArray *listItems = [list componentsSeparatedByString:@", "];
```

produces an array `{ @"Norman", @"Stanley", @"Fletcher" }`.

If *list* begins with a comma and space—for example, `", Norman, Stanley, Fletcher"`—the array has these contents: `{ @"", @"Norman", @"Stanley", @"Fletcher" }`

If *list* has no separators—for example, `"Norman"`—the array contains the string itself, in this case `{ @"Norman" }`.

Availability

Available in Mac OS X v10.0 and later.

See Also

`componentsJoinedByString:` (page 118) (`NSArray`)

`- pathComponents` (page 1585)

Related Sample Code

Birthdays

ColorMatching

CoreRecipes

iSpend

QTKitMovieShuffler

Declared In

`NSString.h`

cString

Returns a representation of the receiver as a C string in the default C-string encoding. (Deprecated in Mac OS X v10.4. Use `cStringUsingEncoding:` (page 1549) or `UTF8String` (page 1612) instead.)

```
- (const char *)cString
```

Discussion
The returned C string will be automatically freed just as a returned object would be released; your code should copy the C string or use `getCString:` (page 1556) if it needs to store the C string outside of the autorelease context in which the C string is created.

Raises an `NSCharacterConversionException` if the receiver can't be represented in the default C-string encoding without loss of information. Use `canBeConvertedToEncoding:` (page 1539) if necessary to check whether a string can be losslessly converted to the default C-string encoding. If it can't, use `lossyCString` (page 1583) or `dataUsingEncoding:allowLossyConversion:` (page 1550) to get a C-string representation with some loss of information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
- `cStringUsingEncoding:` (page 1549)
- `getCString:maxLength:encoding:` (page 1557)
- `UTF8String` (page 1612)

Related Sample Code
WhackedTV

Declared In
`NSString.h`

cStringLength

Returns the length in `char`-sized units of the receiver's C-string representation in the default C-string encoding. (Deprecated in Mac OS X v10.4. Use lengthOfBytesUsingEncoding: (page 1580) or maximumLengthOfBytesUsingEncoding: (page 1584) instead.)

```
- (NSUInteger)cStringLength
```

Discussion
Raises if the receiver can't be represented in the default C-string encoding without loss of information. You can also use `canBeConvertedToEncoding:` (page 1539) to check whether a string can be losslessly converted to the default C-string encoding. If it can't, use `lossyCString` (page 1583) to get a C-string representation with some loss of information, then check its length explicitly using the ANSI function `strlen()`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
- `lengthOfBytesUsingEncoding:` (page 1580)
- `maximumLengthOfBytesUsingEncoding:` (page 1584)
- `UTF8String` (page 1612)

Declared In
`NSString.h`

cStringUsingEncoding:

Returns a representation of the receiver as a C string using a given encoding.

```
- (const char *)cStringUsingEncoding:(NSStringEncoding)encoding
```

Parameters

encoding

The encoding for the returned C string.

Return Value

A C string representation of the receiver using the encoding specified by *encoding*. Returns `NULL` if the receiver cannot be losslessly converted to *encoding*.

Discussion

The returned C string is guaranteed to be valid only until either the receiver is freed, or until the current autorelease pool is emptied, whichever occurs first. You should copy the C string or use `getCString:maxLength:encoding:` (page 1557) if it needs to store the C string beyond this time.

You can use `canBeConvertedToEncoding:` (page 1539) to check whether a string can be losslessly converted to *encoding*. If it can't, you can use `dataUsingEncoding:allowLossyConversion:` (page 1550) to get a C-string representation using *encoding*, allowing some loss of information (note that the data returned by `dataUsingEncoding:allowLossyConversion:` is not a strict C-string since it does not have a `NULL` terminator).

Availability

Available in Mac OS X v10.4 and later.

See Also

- `getCString:` (page 1556)
- `canBeConvertedToEncoding:` (page 1539)
+ `defaultCStringEncoding` (page 1527)
- `cStringLength` (page 1548)
- `getCharacters:` (page 1555)
- `UTF8String` (page 1612)

Related Sample Code

CocoaDVDPlayer

Core Data HTML Store

Declared In

`NSString.h`

dataUsingEncoding:

Returns an `NSData` object containing a representation of the receiver encoded using a given encoding.

```
- (NSData *)dataUsingEncoding:(NSStringEncoding)encoding
```

Parameters

encoding

A string encoding.

Return Value
The result of invoking `dataUsingEncoding:allowLossyConversion:` (page 1550) with `NO` as the second argument (that is, requiring lossless conversion).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
EnhancedAudioBurn
ObjectPath
QTSSConnectionMonitor
QTSSInspector
Sketch-112

Declared In
`NSString.h`

dataUsingEncoding:allowLossyConversion:

Returns an `NSData` object containing a representation of the receiver encoded using a given encoding.

```
- (NSData *)dataUsingEncoding:(NSStringEncoding)encoding
    allowLossyConversion:(BOOL)flag
```

Parameters

encoding
A string encoding.

flag
If `YES`, then allows characters to be removed or altered in conversion.

Return Value
An `NSData` object containing a representation of the receiver encoded using *encoding*. Returns `nil` if *flag* is `NO` and the receiver can't be converted without losing some information (such as accents or case).

Discussion
If *flag* is `YES` and the receiver can't be converted without losing some information, some characters may be removed or altered in conversion. For example, in converting a character from `NSUnicodeStringEncoding` to `NSASCIIStringEncoding`, the character 'Á' becomes 'A', losing the accent.

This method creates an external representation (with a byte order marker, if necessary, to indicate endianness) to ensure that the resulting `NSData` object can be written out to a file safely. The result of this method, when lossless conversion is made, is the default "plain text" format for encoding and is the recommended way to save or transmit a string object.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ availableStringEncodings` (page 1526)
`- canBeConvertedToEncoding:` (page 1539)

Related Sample Code
JavaSplashScreen

Spotlight

Declared In
NSString.h

decomposedStringWithCanonicalMapping

Returns a string made by normalizing the receiver's contents using Form D.

```
- (NSString *)decomposedStringWithCanonicalMapping
```

Return Value
A string made by normalizing the receiver's contents using the Unicode Normalization Form D.

Availability
Available in Mac OS X v10.2 and later.

See Also
- precomposedStringWithCanonicalMapping (page 1587)
- decomposedStringWithCompatibilityMapping (page 1551)

Declared In
NSString.h

decomposedStringWithCompatibilityMapping

Returns a string made by normalizing the receiver's contents using Form KD.

```
- (NSString *)decomposedStringWithCompatibilityMapping
```

Return Value
A string made by normalizing the receiver's contents using the Unicode Normalization Form KD.

Availability
Available in Mac OS X v10.2 and later.

See Also
- precomposedStringWithCompatibilityMapping (page 1588)
- decomposedStringWithCanonicalMapping (page 1551)

Declared In
NSString.h

description

Returns the receiver.

```
- (NSString *)description
```

Return Value
The receiver.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSString.h

doubleValue

Returns the floating-point value of the receiver's text as a `double`.

```
- (double)doubleValue
```

Return Value
The floating-point value of the receiver's text as a `double`. Returns `HUGE_VAL` or `-HUGE_VAL` on overflow, 0.0 on underflow. Returns `0.0` if the receiver doesn't begin with a valid text representation of a floating-point number.

Discussion
This method skips any whitespace at the beginning of the string. This method uses formatting information stored in the non-localized value; use an `NSScanner` object for localized scanning of numeric values from a string.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `floatValue` (page 1553)
- `longLongValue` (page 1583)
- `integerValue` (page 1576)
- `scanDouble:` (page 1350) (`NSScanner`)

Related Sample Code
JavaFrameEmbedding example
QTMetadataEditor
TimelineToTC
TrackBall

Declared In
NSString.h

fastestEncoding

Returns the fastest encoding to which the receiver may be converted without loss of information.

```
- (NSStringEncoding)fastestEncoding
```

Return Value
The fastest encoding to which the receiver may be converted without loss of information.

Discussion
"Fastest" applies to retrieval of characters from the string. This encoding may not be space efficient.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `smallestEncoding` (page 1595)
- `getCharacters:range:` (page 1555)

Declared In
`NSString.h`

fileSystemRepresentation

Returns a file system-specific representation of the receiver.

```
- (const char *)fileSystemRepresentation
```

Return Value
A file system-specific representation of the receiver, as described for `getFileSystemRepresentation:maxLength:` (page 1559).

Discussion
The returned C string will be automatically freed just as a returned object would be released; your code should copy the representation or use `getFileSystemRepresentation:maxLength:` (page 1559) if it needs to store the representation outside of the autorelease context in which the representation is created.

Raises an `NSCharacterConversionException` if the receiver can't be represented in the file system's encoding.

Note that this method only works with file paths (not, for example, string representations of URLs).

To convert a `char *` path (such as you might get from a C library routine) to an `NSString` object, use `NSFileManager`'s `stringWithFileSystemRepresentation:length:` (page 659) method.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
JavaSplashScreen

Declared In
`NSPathUtilities.h`

floatValue

Returns the floating-point value of the receiver's text as a `float`.

```
- (float)floatValue
```

Return Value
The floating-point value of the receiver's text as a `float`, skipping whitespace at the beginning of the string. Returns `HUGE_VAL` or `-HUGE_VAL` on overflow, `0.0` on underflow. Also returns `0.0` if the receiver doesn't begin with a valid text representation of a floating-point number.

Discussion
This method uses formatting information stored in the non-localized value; use an `NSScanner` object for localized scanning of numeric values from a string.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `doubleValue` (page 1552)
- `longLongValue` (page 1583)
- `integerValue` (page 1576)
- `scanFloat:` (page 1350) (`NSScanner`)

Related Sample Code
WhackedTV

Declared In
`NSString.h`

getBytes:maxLength:usedLength:encoding:options:range:remainingRange:

Gets a given range of characters as bytes in a specified encoding.

```
- (BOOL)getBytes:(void *)buffer maxLength:(NSUInteger)maxBufferCount
    usedLength:(NSUInteger *)usedBufferCount encoding:(NSStringEncoding)encoding
    options:(NSStringEncodingConversionOptions)options range:(NSRange)range
    remainingRange:(NSRangePointer)leftover
```

Parameters

buffer
A buffer into which to store the bytes from the receiver. The returned bytes are *not* `NULL`-terminated.

maxBufferCount
The maximum number of bytes to write to *buffer*.

usedBufferCount
The number of bytes used from *buffer*. Pass `NULL` if you do not need this value.

encoding
The encoding to use for the returned bytes.

options
A mask to specify options to use for converting the receiver's contents to *encoding* (if conversion is necessary).

range
The range of characters in the receiver to get.

leftover
The remaining range. Pass `NULL` If you do not need this value.

Return Value
`YES` if some characters were converted, otherwise `NO`.

Discussion
Conversion might stop when the buffer fills, but it might also stop when the conversion isn't possible due to the chosen encoding.

Availability
Available in Mac OS X v10.5 and later.

Declared In
NSString.h

getCharacters:

Copies all characters from the receiver into a given buffer.

```
- (void)getCharacters:(unichar *)buffer
```

Parameters

buffer

Upon return, contains the characters from the receiver. *buffer* must be large enough to contain all characters in the string ([string length]*sizeof(unichar)).

Discussion
Invokes getCharacters:range: (page 1555) with *buffer* and the entire extent of the receiver as the range.

Availability
Available in Mac OS X v10.0 and later.

See Also
- length (page 1580)

Related Sample Code
JSheets

Declared In
NSString.h

getCharacters:range:

Copies characters from a given range in the receiver into a given buffer.

```
- (void)getCharacters:(unichar *)buffer range:(NSRange)aRange
```

Parameters

buffer

Upon return, contains the characters from the receiver. *buffer* must be large enough to contain the characters in the range *aRange* (aRange.length*sizeof(unichar)).

aRange

The range of characters to retrieve. The range must not exceed the bounds of the receiver.

Important: Raises an NSRangeException if any part of *aRange* lies beyond the bounds of the receiver.

Discussion
This method does not add a NULL character.

The abstract implementation of this method uses `characterAtIndex:` (page 1540) repeatedly, correctly extracting the characters, though very inefficiently. Subclasses should override it to provide a fast implementation.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSString.h`

getCString:

Invokes `getCString:maxLength:range:remainingRange:` (page 1558) with `NSMaximumStringLength` as the maximum length, the receiver's entire extent as the range, and `NULL` for the remaining range. (Deprecated in Mac OS X v10.4. Use cStringUsingEncoding: (page 1549) or `dataUsingEncoding:allowLossyConversion:` (page 1550) instead.)

```
- (void)getCString:(char *)buffer
```

Discussion
buffer must be large enough to contain the resulting C-string plus a terminating NULL character (which this method adds—`[string cStringLength]`).

Raises an `NSCharacterConversionException` if the receiver can't be represented in the default C-string encoding without loss of information. Use `canBeConvertedToEncoding:` (page 1539) if necessary to check whether a string can be losslessly converted to the default C-string encoding. If it can't, use `lossyCString` (page 1583) or `dataUsingEncoding:allowLossyConversion:` (page 1550) to get a C-string representation with some loss of information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
- `cStringUsingEncoding:` (page 1549)
- `getCString:maxLength:encoding:` (page 1557)
- `UTF8String` (page 1612)

Related Sample Code
QTMetadataEditor
ThreadsExporter
ThreadsImporter
ThreadsImportMovie

Declared In
`NSString.h`

getCString:maxLength:

Invokes `getCString:maxLength:range:remainingRange:` (page 1558) with *maxLength* as the maximum length in `char`-sized units, the receiver's entire extent as the range, and `NULL` for the remaining range. (Deprecated in Mac OS X v10.4. Use getCString:maxLength:encoding: (page 1557) instead.)

```
- (void)getCString:(char *)buffer maxLength:(NSUInteger)maxLength
```

Discussion
buffer must be large enough to contain *maxLength* `char`s plus a terminating zero `char` (which this method adds).

Raises an `NSCharacterConversionException` if the receiver can't be represented in the default C-string encoding without loss of information. Use `canBeConvertedToEncoding:` (page 1539) if necessary to check whether a string can be losslessly converted to the default C-string encoding. If it can't, use `lossyCString` (page 1583) or `dataUsingEncoding:allowLossyConversion:` (page 1550) to get a C-string representation with some loss of information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
- `cStringUsingEncoding:` (page 1549)
- `getCString:maxLength:encoding:` (page 1557)
- `UTF8String` (page 1612)

Declared In
`NSString.h`

getCString:maxLength:encoding:

Converts the receiver's content to a given encoding and stores them in a buffer.

```
- (BOOL)getCString:(char *)buffer maxLength:(NSUInteger)maxBufferCount
    encoding:(NSStringEncoding)encoding
```

Parameters

buffer
Upon return, contains the converted C-string plus the `NULL` termination byte. The buffer must include room for *maxBufferCount* bytes.

maxBufferCount
The maximum number of bytes in the string to return in buffer (*including* the `NULL` termination byte).

encoding
The encoding for the returned C string.

Return Value
`YES` if the operation was successful, otherwise `NO`. Returns `NO` if conversion is not possible due to encoding errors or if *buffer* is too small.

Discussion
Note that in the treatment of the *maxBufferCount* argument, this method differs from the deprecated getCString:maxLength: (page 1557) method which it replaces. (The buffer should include room for *maxBufferCount* bytes; this number should accommodate the expected size of the return value plus the NULL termination byte, which this method adds.)

You can use canBeConvertedToEncoding: (page 1539) to check whether a string can be losslessly converted to *encoding*. If it can't, you can use dataUsingEncoding:allowLossyConversion: (page 1550) to get a C-string representation using *encoding*, allowing some loss of information (note that the data returned by dataUsingEncoding:allowLossyConversion: is not a strict C-string since it does not have a NULL terminator).

Availability
Available in Mac OS X v10.4 and later.

See Also
- cStringUsingEncoding: (page 1549)
- canBeConvertedToEncoding: (page 1539)
- getCharacters: (page 1555)
- UTF8String (page 1612)

Related Sample Code
QTMetadataEditor

Declared In
NSString.h

getCString:maxLength:range:remainingRange:

Converts the receiver's content to the default C-string encoding and stores them in a given buffer. (Deprecated in Mac OS X v10.4. Use getCString:maxLength:encoding: (page 1557) instead.)

```
- (void)getCString:(char *)buffer maxLength:(NSUInteger)maxLength
    range:(NSRange)aRange remainingRange:(NSRangePointer)leftoverRange
```

Discussion
buffer must be large enough to contain *maxLength* bytes plus a terminating zero character (which this method adds). Copies and converts as many characters as possible from *aRange* and stores the range of those not converted in the range given by *leftoverRange* (if it's non-nil). Raises an NSRangeException if any part of *aRange* lies beyond the end of the string.

Raises an NSCharacterConversionException if the receiver can't be represented in the default C-string encoding without loss of information. Use canBeConvertedToEncoding: (page 1539) if necessary to check whether a string can be losslessly converted to the default C-string encoding. If it can't, use lossyCString (page 1583) or dataUsingEncoding:allowLossyConversion: (page 1550) to get a C-string representation with some loss of information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
- cStringUsingEncoding: (page 1549)

- getCString:maxLength:encoding: (page 1557)
- UTF8String (page 1612)

Declared In
NSString.h

getFileSystemRepresentation:maxLength:

Interprets the receiver as a system-independent path and fills a buffer with a C-string in a format and encoding suitable for use with file-system calls.

```
- (BOOL)getFileSystemRepresentation:(char *)buffer maxLength:(NSUInteger)maxLength
```

Parameters

buffer

Upon return, contains a C-string that represent the receiver as as a system-independent path, plus the NULL termination byte. The size of *buffer* must be large enough to contain *maxLength* bytes.

maxLength

The maximum number of bytes in the string to return in *buffer* (including a terminating NULL character, which this method adds).

Return Value
YES if *buffer* is successfully filled with a file-system representation, otherwise NO (for example, if *maxLength* would be exceeded or if the receiver can't be represented in the file system's encoding).

Discussion
This method operates by replacing the abstract path and extension separator characters ('/' and '.' respectively) with their equivalents for the operating system. If the system-specific path or extension separator appears in the abstract representation, the characters it is converted to depend on the system (unless they're identical to the abstract separators).

Note that this method only works with file paths (not, for example, string representations of URLs).

The following example illustrates the use of the *maxLength* argument. The first method invocation returns failure as the file representation of the string (@"/mach_kernel") is 12 bytes long and the value passed as the *maxLength* argument (12) does not allow for the addition of a NULL termination byte.

```
char filenameBuffer[13];
BOOL success;
success = [@"/mach_kernel" getFileSystemRepresentation:filenameBuffer
maxLength:12];
// success == NO
// Changing the length to include the NULL character does work
success = [@"/mach_kernel" getFileSystemRepresentation:filenameBuffer
maxLength:13];
// success == YES
```

Availability
Available in Mac OS X v10.0 and later.

See Also
- fileSystemRepresentation (page 1553)

Declared In
NSPathUtilities.h

getLineStart:end:contentsEnd:forRange:

Returns by reference the beginning of the first line and the end of the last line touched by the given range.

```
- (void)getLineStart:(NSUInteger *)startIndex end:(NSUInteger *)lineEndIndex
    contentsEnd:(NSUInteger *)contentsEndIndex forRange:(NSRange)aRange
```

Parameters

startIndex

Upon return, contains the index of the first character of the line containing the beginning of *aRange*. Pass NULL if you do not need this value (in which case the work to compute the value isn't performed).

lineEndIndex

Upon return, contains the index of the first character past the terminator of the line containing the end of *aRange*. Pass NULL if you do not need this value (in which case the work to compute the value isn't performed).

contentsEndIndex

Upon return, contains the index of the first character of the terminator of the line containing the end of *aRange*. Pass NULL if you do not need this value (in which case the work to compute the value isn't performed).

aRange

A range within the receiver. The value must not exceed the bounds of the receiver.

Important: Raises an NSRangeException if any part of *aRange* lies beyond the end of the string.

Discussion
A line is delimited by any of these characters, the longest possible sequence being preferred to any shorter:

- U+000D (\r or CR)
- U+2028 (Unicode line separator)
- U+000A (\n or LF)
- U+2029 (Unicode paragraph separator)
- \r\n, in that order (also known as CRLF)

If *aRange* is contained with a single line, of course, the returned indexes all belong to that line. You can use the results of this method to construct ranges for lines by using the start index as the range's location and the difference between the end index and the start index as the range's length.

Availability
Available in Mac OS X v10.0 and later.

See Also
- lineRangeForRange: (page 1581)
- substringWithRange: (page 1611)

Declared In
NSString.h

getParagraphStart:end:contentsEnd:forRange:

Returns by reference the beginning of the first paragraph and the end of the last paragraph touched by the given range.

```
- (void)getParagraphStart:(NSUInteger *)startIndex end:(NSUInteger *)endIndex
    contentsEnd:(NSUInteger *)contentsEndIndex forRange:(NSRange)aRange
```

Parameters

startIndex

Upon return, contains the index of the first character of the paragraph containing the beginning of *aRange*. Pass NULL if you do not need this value (in which case the work to compute the value isn't performed).

endIndex

Upon return, contains the index of the first character past the terminator of the paragraph containing the end of *aRange*. Pass NULL if you do not need this value (in which case the work to compute the value isn't performed).

contentsEndIndex

Upon return, contains the index of the first character of the terminator of the paragraph containing the end of *aRange*. Pass NULL if you do not need this value (in which case the work to compute the value isn't performed).

aRange

A range within the receiver. The value must not exceed the bounds of the receiver.

Discussion
If *aRange* is contained with a single paragraph, of course, the returned indexes all belong to that paragraph. Similar to `getLineStart:end:contentsEnd:forRange:` (page 1560), you can use the results of this method to construct the ranges for paragraphs.

Availability
Available in Mac OS X v10.3 and later.

See Also
`- paragraphRangeForRange:` (page 1585)

Declared In
NSString.h

hash

Returns an unsigned integer that can be used as a hash table address.

```
- (NSUInteger)hash
```

Return Value
An unsigned integer that can be used as a hash table address.

Discussion
If two string objects are equal (as determined by the `isEqualToString:` (page 1578) method), they must have the same hash value. The abstract implementation of this method fulfills this requirement, so subclasses of `NSString` shouldn't override it.

You should not rely on this method returning the same hash value across releases of Mac OS X.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSString.h`

hasPrefix:

Returns a Boolean value that indicates whether a given string matches the beginning characters of the receiver.

```
- (BOOL)hasPrefix:(NSString *)aString
```

Parameters

aString
A string.

Return Value
`YES` if *aString* matches the beginning characters of the receiver, otherwise `NO`. Returns `NO` if *aString* is empty.

Discussion
This method is a convenience for comparing strings using the `NSAnchoredSearch` option. See *String Programming Guide for Cocoa* for more information.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `hasSuffix:` (page 1562)
- `compare:options:range:` (page 1543)

Related Sample Code
Reminders

Declared In
`NSString.h`

hasSuffix:

Returns a Boolean value that indicates whether a given string matches the ending characters of the receiver.

```
- (BOOL)hasSuffix:(NSString *)aString
```

Parameters

aString

A string.

Return Value

YES if *aString* matches the ending characters of the receiver, otherwise NO. Returns NO if *aString* is empty.

Discussion

This method is a convenience for comparing strings using the NSAnchoredSearch and NSBackwardsSearch options. See *String Programming Guide for Cocoa* for more information.

Availability

Available in Mac OS X v10.0 and later.

See Also

- hasPrefix: (page 1562)
- compare:options:range: (page 1543)

Declared In

NSString.h

init

Returns an initialized NSString object that contains no characters.

```
- (id)init
```

Return Value

An initialized NSString object that contains no characters. The returned object may be different from the original receiver.

Availability

Available in Mac OS X v10.0 and later.

See Also

+ string (page 1529)

Declared In

NSString.h

initWithBytes:length:encoding:

Returns an initialized NSString object containing a given number of bytes from a given C array of bytes in a given encoding.

```
- (id)initWithBytes:(const void *)bytes length:(NSUInteger)length
    encoding:(NSStringEncoding)encoding
```

Parameters

bytes

A C array of bytes in the encoding specified by *encoding*. The array must not contain NULL.

length
The number of bytes to use from *bytes*.

encoding
The character encoding of *bytes*.

Return Value
An initialized `NSString` object containing *length* bytes from *bytes* interpreted using the encoding *encoding*. The returned object may be different from the original receiver.

Availability
Available in Mac OS X v10.3 and later.

See Also
`- initWithBytesNoCopy:length:encoding:freeWhenDone:` (page 1564)

Related Sample Code
VideoHardwareInfo

Declared In
`NSString.h`

initWithBytesNoCopy:length:encoding:freeWhenDone:

Returns an initialized `NSString` object that contains a given number of bytes from a given C array of bytes in a given encoding, and optionally frees the array on deallocation.

```
- (id)initWithBytesNoCopy:(void *)bytes length:(NSUInteger)length
    encoding:(NSStringEncoding)encoding freeWhenDone:(BOOL)flag
```

Parameters

bytes
A C array of bytes in the encoding specified by *encoding*. The array must not contain `NULL`.

length
The number of bytes to use from *bytes*.

encoding
The character encoding of *bytes*.

flag
If `YES`, the receiver will free the memory when it no longer needs the data; if `NO` it won't.

Return Value
An initialized `NSString` object containing *length* bytes from *bytes* interpreted using the encoding *encoding*. The returned object may be different from the original receiver.

Special Considerations

If an error occurs during the creation of the string, then *bytes* is not freed even if *flag* is `YES`. In this case, the caller is responsible for freeing the buffer. This allows the caller to continue trying to create a string with the buffer, without having the buffer deallocated.

Availability
Available in Mac OS X v10.3 and later.

See Also
`- initWithBytes:length:encoding:` (page 1563)

Related Sample Code
QTRecorder

Declared In
`NSString.h`

initWithCharacters:length:

Returns an initialized `NSString` object that contains a given number of characters from a given C array of Unicode characters.

```
- (id)initWithCharacters:(const unichar *)characters length:(NSUInteger)length
```

Parameters

characters
A C array of Unicode characters; the value must not be `NULL`.

Important: Raises an exception if *characters* is `NULL`, even if *length* is `0`.

length
The number of characters to use from *characters*.

Return Value
An initialized `NSString` object containing *length* characters taken from *characters*. The returned object may be different from the original receiver.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ stringWithCharacters:length:` (page 1530)

Declared In
`NSString.h`

initWithCharactersNoCopy:length:freeWhenDone:

Returns an initialized `NSString` object that contains a given number of characters from a given C array of Unicode characters.

```
- (id)initWithCharactersNoCopy:(unichar *)characters length:(NSUInteger)length
    freeWhenDone:(BOOL)flag
```

Parameters

characters
A C array of Unicode characters.

length
The number of characters to use from *characters*.

flag
If `YES`, the receiver will free the memory when it no longer needs the characters; if `NO` it won't.

Return Value
An initialized `NSString` object that contains *`length`* characters from *`characters`*. The returned object may be different from the original receiver.

Special Considerations

If an error occurs during the creation of the string, then *`bytes`* is not freed even if *`flag`* is `YES`. In this case, the caller is responsible for freeing the buffer. This allows the caller to continue trying to create a string with the buffer, without having the buffer deallocated.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ stringWithCharacters:length:` (page 1530)

Declared In
`NSString.h`

initWithContentsOfFile:

Initializes the receiver, a newly allocated `NSString` object, by reading data from the file named by *`path`*. (Deprecated in Mac OS X v10.4. Use `initWithContentsOfFile:encoding:error:` (page 1566) or `initWithContentsOfFile:usedEncoding:error:` (page 1567) instead.)

```
- (id)initWithContentsOfFile:(NSString *)path
```

Discussion
Initializes the receiver, a newly allocated `NSString` object, by reading data from the file named by *`path`*. If the contents begin with a byte-order mark (`U+FEFF` or `U+FFFE`), interprets the contents as Unicode characters; otherwise interprets the contents as data in the default C string encoding. Returns an initialized object, which might be different from the original receiver, or `nil` if the file can't be opened.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
- `initWithContentsOfFile:encoding:error:` (page 1566)
- `initWithContentsOfFile:usedEncoding:error:` (page 1567)

Declared In
`NSString.h`

initWithContentsOfFile:encoding:error:

Returns an `NSString` object initialized by reading data from the file at a given path using a given encoding.

```
- (id)initWithContentsOfFile:(NSString *)path encoding:(NSStringEncoding)enc
    error:(NSError **)error
```

Parameters

path

A path to a file.

enc

The encoding of the file at *path*.

error

If an error occurs, upon return contains an `NSError` object that describes the problem. If you are not interested in possible errors, pass in `NULL`.

Return Value

An `NSString` object initialized by reading data from the file named by *path* using the encoding, *enc*. The returned object may be different from the original receiver. If the file can't be opened or there is an encoding error, returns `nil`.

Availability

Available in Mac OS X v10.4 and later.

See Also

`+ stringWithContentsOfFile:encoding:error:` (page 1531)

`- initWithContentsOfFile:usedEncoding:error:` (page 1567)

Declared In

`NSString.h`

initWithContentsOfFile:usedEncoding:error:

Returns an `NSString` object initialized by reading data from the file at a given path and returns by reference the encoding used to interpret the characters.

```
- (id)initWithContentsOfFile:(NSString *)path usedEncoding:(NSStringEncoding *)enc
    error:(NSError **)error
```

Parameters

path

A path to a file.

enc

Upon return, if the file is read successfully, contains the encoding used to interpret the file at *path*.

error

If an error occurs, upon returns contains an `NSError` object that describes the problem. If you are not interested in possible errors, pass in `NULL`.

Return Value

An `NSString` object initialized by reading data from the file named by *path*. The returned object may be different from the original receiver. If the file can't be opened or there is an encoding error, returns `nil`.

Availability

Available in Mac OS X v10.4 and later.

See Also

`+ stringWithContentsOfFile:encoding:error:` (page 1531)

`- initWithContentsOfFile:encoding:error:` (page 1566)

Declared In
NSString.h

initWithContentsOfURL:

Initializes the receiver, a newly allocated NSString object, by reading data from the location named by a given URL. (Deprecated in Mac OS X v10.4. Use initWithContentsOfURL:encoding:error: (page 1568) or initWithContentsOfURL:usedEncoding:error: (page 1569) instead.)

```
- (id)initWithContentsOfURL:(NSURL *)aURL
```

Discussion
Initializes the receiver, a newly allocated NSString object, by reading data from the location named by *aURL*. If the contents begin with a byte-order mark (U+FEFF or U+FFFE), interprets the contents as Unicode characters; otherwise interprets the contents as data in the default C string encoding. Returns an initialized object, which might be different from the original receiver, or nil if the location can't be opened.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
- initWithContentsOfURL:encoding:error: (page 1568)
- initWithContentsOfURL:usedEncoding:error: (page 1569)

Declared In
NSString.h

initWithContentsOfURL:encoding:error:

Returns an NSString object initialized by reading data from a given URL interpreted using a given encoding.

```
- (id)initWithContentsOfURL:(NSURL *)url encoding:(NSStringEncoding)enc
    error:(NSError **)error
```

Parameters

url
The URL to read.

enc
The encoding of the file at *path*.

error
If an error occurs, upon returns contains an NSError object that describes the problem. If you are not interested in possible errors, pass in NULL.

Return Value
An NSString object initialized by reading data from *url*. The returned object may be different from the original receiver. If the URL can't be opened or there is an encoding error, returns nil.

Availability
Available in Mac OS X v10.4 and later.

See Also
+ stringWithContentsOfURL:encoding:error: (page 1533)

Declared In
NSString.h

initWithContentsOfURL:usedEncoding:error:

Returns an NSString object initialized by reading data from a given URL and returns by reference the encoding used to interpret the data.

```
- (id)initWithContentsOfURL:(NSURL *)url usedEncoding:(NSStringEncoding *)enc
    error:(NSError **)error
```

Parameters

url

The URL from which to read data.

enc

Upon return, if *url* is read successfully, contains the encoding used to interpret the data.

error

If an error occurs, upon returns contains an NSError object that describes the problem. If you are not interested in possible errors, pass in NULL.

Return Value
An NSString object initialized by reading data from *url*. If *url* can't be opened or the encoding cannot be determined, returns nil. The returned initialized object might be different from the original receiver

Availability
Available in Mac OS X v10.4 and later.

See Also
+ stringWithContentsOfURL:usedEncoding:error: (page 1534)

Declared In
NSString.h

initWithCString:

Initializes the receiver, a newly allocated NSString object, by converting the data in a given C-string from the default C-string encoding into the Unicode character encoding. (Deprecated in Mac OS X v10.4. Use initWithCString:encoding: (page 1570) instead.)

```
- (id)initWithCString:(const char *)cString
```

Discussion
cString must be a zero-terminated C string in the default C string encoding, and may not be NULL. Returns an initialized object, which might be different from the original receiver.

To create an immutable string from an immutable C string buffer, do not attempt to use this method. Instead, use initWithCStringNoCopy:length:freeWhenDone: (page 1571).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
- `initWithCString:encoding:` (page 1570)

Related Sample Code
SimplePlayThru

Declared In
`NSString.h`

initWithCString:encoding:

Returns an `NSString` object initialized using the characters in a given C array, interpreted according to a given encoding.

```
- (id)initWithCString:(const char *)nullTerminatedCString
    encoding:(NSStringEncoding)encoding
```

Parameters

`nullTerminatedCString`
A C array of characters. The array must end with a `NULL` character; intermediate `NULL` characters are not allowed.

`encoding`
The encoding of *`nullTerminatedCString`*.

Return Value
An `NSString` object initialized using the characters from *`nullTerminatedCString`*. The returned object may be different from the original receiver

Discussion
If *`nullTerminatedCString`* is not a `NULL`-terminated C string, or *`encoding`* does not match the actual encoding, the results are undefined.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `+ stringWithCString:` (page 1534)
- `- initWithCStringNoCopy:length:freeWhenDone:` (page 1571)
- `+ defaultCStringEncoding` (page 1527)

Declared In
`NSString.h`

initWithCString:length:

Initializes the receiver, a newly allocated `NSString` object, by converting the data in a given C-string from the default C-string encoding into the Unicode character encoding. (Deprecated in Mac OS X v10.4. Use initWithCString:encoding: (page 1570) instead.)

```
- (id)initWithCString:(const char *)cString length:(NSUInteger)length
```

Discussion
This method converts *length* * `sizeof(char)` bytes from *cString* and doesn't stop short at a zero character. *cString* must contain bytes in the default C-string encoding and may not be `NULL`. Returns an initialized object, which might be different from the original receiver.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
`- initWithCString:encoding:` (page 1570)

Related Sample Code
CocoaSpeechSynthesisExample

Declared In
`NSString.h`

initWithCStringNoCopy:length:freeWhenDone:

Initializes the receiver, a newly allocated `NSString` object, by converting the data in a given C-string from the default C-string encoding into the Unicode character encoding. (Deprecated in Mac OS X v10.4. Use `initWithBytesNoCopy:length:encoding:freeWhenDone:` (page 1564) instead.)

```
- (id)initWithCStringNoCopy:(char *)cString length:(NSUInteger)length
    freeWhenDone:(BOOL)flag
```

Discussion
This method converts *length* * `sizeof(char)` bytes from *cString* and doesn't stop short at a zero character. *cString* must contain data in the default C-string encoding and may not be `NULL`. The receiver becomes the owner of *cString*; if *flag* is `YES` it will free the memory when it no longer needs it, but if *flag* is `NO` it won't. Returns an initialized object, which might be different from the original receiver.

You can use this method to create an immutable string from an immutable (`const char *`) C-string buffer. If you receive a warning message, you can disregard it; its purpose is simply to warn you that the C string passed as the method's first argument may be modified. If you make certain the `freeWhenDone` argument to `initWithStringNoCopy` is `NO`, the C string passed as the method's first argument cannot be modified, so you can safely use `initWithStringNoCopy` to create an immutable string from an immutable (`const char *`) C-string buffer.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
`- initWithCString:encoding:` (page 1570)

Declared In
`NSString.h`

initWithData:encoding:

Returns an `NSString` object initialized by converting given data into Unicode characters using a given encoding.

```
- (id)initWithData:(NSData *)data encoding:(NSStringEncoding)encoding
```

Parameters

data

An `NSData` object containing bytes in *encoding* and the default plain text format (that is, pure content with no attributes or other markups) for that encoding.

encoding

The encoding used by *data*.

Return Value

An `NSString` object initialized by converting the bytes in *data* into Unicode characters using *encoding*. The returned object may be different from the original receiver. Returns `nil` if the initialization fails for some reason (for example if *data* does not represent valid data for *encoding*).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AutoUpdater
EnhancedAudioBurn
GridCalendar
Moriarity
NameAndPassword

Declared In

`NSString.h`

initWithFormat:

Returns an *NSString* object initialized by using a given format string as a template into which the remaining argument values are substituted.

```
- (id)initWithFormat:(NSString *)format ...
```

Parameters

format

A format string. See Formatting String Objects for examples of how to use this method, and String Format Specifiers for a list of format specifiers. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *format* is `nil`.

...

A comma-separated list of arguments to substitute into *format*.

Return Value
An *NSString* object initialized by using *format* as a template into which the remaining argument values are substituted according to the canonical locale. The returned object may be different from the original receiver.

Discussion
Invokes `initWithFormat:locale:arguments:` (page 1574) with `nil` as the locale, hence using the canonical locale to format numbers. This is useful, for example, if you want to produce "non-localized" formatting which needs to be written out to files and parsed back later.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ stringWithFormat:` (page 1536)
`- initWithFormat:locale:arguments:` (page 1574)

Declared In
`NSString.h`

initWithFormat:arguments:

Returns an *NSString* object initialized by using a given format string as a template into which the remaining argument values are substituted according to the user's default locale.

```
- (id)initWithFormat:(NSString *)format arguments:(va_list)argList
```

Parameters

format

A format string. See Formatting String Objects for examples of how to use this method, and String Format Specifiers for a list of format specifiers. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *format* is `nil`.

argList

A list of arguments to substitute into *format*.

Return Value
An *NSString* object initialized by using *format* as a template into which the values in *argList* are substituted according to the user's default locale. The returned object may be different from the original receiver.

Discussion
Invokes `initWithFormat:locale:arguments:` (page 1574) with `nil` as the locale.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ stringWithFormat:` (page 1536)

Declared In
`NSString.h`

initWithFormat:locale:

Returns an *NSString* object initialized by using a given format string as a template into which the remaining argument values are substituted according to given locale information.

```
- (id)initWithFormat:(NSString *)format locale:(id)locale ...
```

Parameters

format

A format string. See Formatting String Objects for examples of how to use this method, and String Format Specifiers for a list of format specifiers. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *format* is `nil`.

locale

This may be an instance of `NSDictionary` containing locale information or an instance of `NSLocale`. If this value is `nil`, uses the canonical locale.

To use a dictionary containing the current user's locale, you can use `[[NSUserDefaults standardUserDefaults] dictionaryRepresentation]`.

...

A comma-separated list of arguments to substitute into *format*.

Discussion

Invokes `initWithFormat:locale:arguments:` (page 1574) with *locale* as the locale.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ localizedStringWithFormat:` (page 1528)

Declared In

`NSString.h`

initWithFormat:locale:arguments:

Returns an *NSString* object initialized by using a given format string as a template into which the remaining argument values are substituted according to given locale information.

```
- (id)initWithFormat:(NSString *)format locale:(id)locale arguments:(va_list)argList
```

Parameters

format

A format string. See Formatting String Objects for examples of how to use this method, and String Format Specifiers for a list of format specifiers. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *format* is `nil`.

locale

This may be an instance of `NSDictionary` containing locale information or an instance of `NSLocale`. If this value is `nil`, uses the canonical locale.

To use a dictionary containing the current user's locale, you can use `[[NSUserDefaults standardUserDefaults] dictionaryRepresentation]`.

argList

A list of arguments to substitute into *format*.

Return Value

An *NSString* object initialized by using *format* as a template into which values in *argList* are substituted according the locale information in *locale*. The returned object may be different from the original receiver.

Discussion

The following code fragment illustrates how to create a string from *myArgs*, which is derived from a string object with the value "Cost:" and an `int` with the value 32:

```
va_list myArgs;

NSString *myString = [[NSString alloc] initWithFormat:@"%@:  %d\n"
        locale:[[NSUserDefaults standardUserDefaults] dictionaryRepresentation]
        arguments:myArgs];
```

The resulting string has the value "`Cost: 32\n`".

See *String Programming Guide for Cocoa* for more information.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- initWithFormat:arguments:` (page 1573)

Declared In

`NSString.h`

initWithString:

Returns an `NSString` object initialized by copying the characters from another given string.

```
- (id)initWithString:(NSString *)aString
```

Parameters

aString

The string from which to copy characters. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aString* is `nil`.

Return Value

An `NSString` object initialized by copying the characters from *aString*. The returned object may be different from the original receiver.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ stringWithString:` (page 1537)

Declared In

`NSString.h`

initWithUTF8String:

Returns an `NSString` object initialized by copying the characters a given C array of UTF8-encoded bytes.

```
- (id)initWithUTF8String:(const char *)bytes
```

Parameters

bytes

A `NULL`-terminated C array of bytes in UTF-8 encoding. This value must not be `NULL`.

Important: Raises an exception if *bytes* is `NULL`.

Return Value

An `NSString` object initialized by copying the bytes from *bytes*. The returned object may be different from the original receiver.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ stringWithUTF8String:` (page 1537)

Related Sample Code

Reminders

Declared In

`NSString.h`

integerValue

Returns the `NSInteger` value of the receiver's text.

```
- (NSInteger)integerValue
```

Return Value
The `NSInteger` value of the receiver's text, assuming a decimal representation and skipping whitespace at the beginning of the string. Returns `0` if the receiver doesn't begin with a valid decimal text representation of a number.

Discussion
This method uses formatting information stored in the non-localized value; use an `NSScanner` object for localized scanning of numeric values from a string.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `doubleValue` (page 1552)
- `floatValue` (page 1553)
- `scanInt:` (page 1353) (`NSScanner`)

Related Sample Code
Core Data HTML Store

Declared In
`NSString.h`

intValue

Returns the integer value of the receiver's text.

```
- (int)intValue
```

Return Value
The integer value of the receiver's text, assuming a decimal representation and skipping whitespace at the beginning of the string. Returns `INT_MAX` or `INT_MIN` on overflow. Returns `0` if the receiver doesn't begin with a valid decimal text representation of a number.

Discussion
This method uses formatting information stored in the non-localized value; use an `NSScanner` object for localized scanning of numeric values from a string.

Special Considerations

On Mac OS X v10.5 and later, use `integerValue` (page 1576) instead.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `integerValue` (page 1576)
- `doubleValue` (page 1552)
- `floatValue` (page 1553)
- `scanInt:` (page 1353) (`NSScanner`)

Related Sample Code
AlbumToSlideshow
DatePicker
QTAudioExtractionPanel
QTMetadataEditor
WebKitDOMElementPlugIn

Declared In
`NSString.h`

isAbsolutePath

Returning a Boolean value that indicates whether the receiver represents an absolute path.

```
- (BOOL)isAbsolutePath
```

Return Value
`YES` if the receiver (if interpreted as a path) represents an absolute path, otherwise `NO` (if the receiver represents a relative path).

Discussion
See *String Programming Guide for Cocoa* for more information on paths.

Note that this method only works with file paths (not, for example, string representations of URLs). The method does not check the filesystem for the existence of the path (use `fileExistsAtPath:` (page 646) or similar methods in `NSFileManager` for that task).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSPathUtilities.h`

isEqualToString:

Returns a Boolean value that indicates whether a given string is equal to the receiver using an literal Unicode-based comparison.

```
- (BOOL)isEqualToString:(NSString *)aString
```

Parameters
aString
The string with which to compare the receiver.

Return Value
`YES` if *aString* is equivalent to the receiver (if they have the same `id` or if they are `NSOrderedSame` in a literal comparison), otherwise `NO`.

Discussion
The comparison uses the canonical representation of strings, which for a particular string is the length of the string plus the Unicode characters that make up the string. When this method compares two strings, if the individual Unicodes are the same, then the strings are equal, regardless of the backing store. “Literal” when

applied to string comparison means that various Unicode decomposition rules are not applied and Unicode characters are individually compared. So, for instance, "Ö" represented as the composed character sequence "O" and umlaut would not compare equal to "Ö" represented as one Unicode character.

Special Considerations

When you know both objects are strings, this method is a faster way to check equality than `isEqual:` (page 2101).

Availability
Available in Mac OS X v10.0 and later.

See Also
`- compare:options:range:` (page 1543)

Related Sample Code
Core Data HTML Store
NameAndAddress
People
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSString.h`

lastPathComponent

Returns the last path component of the receiver.

```
- (NSString *)lastPathComponent
```

Return Value
The last path component of the receiver.

Discussion
The following table illustrates the effect of `lastPathComponent` on a variety of different paths:

Receiver's String Value	String Returned
"/tmp/scratch.tiff"	"scratch.tiff"
"/tmp/scratch"	"scratch"
"/tmp/"	"tmp"
"scratch"	"scratch"
"/"	"/"

Note that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
EnhancedAudioBurn
OpenGLCompositorLab
Quartz Composer WWDC 2005 TextEdit
StickiesExample
TextEditPlus

Declared In
`NSPathUtilities.h`

length

Returns the number of Unicode characters in the receiver.

```
- (NSUInteger)length
```

Return Value
The number of Unicode characters in the receiver.

Discussion
The number returned includes the individual characters of composed character sequences, so you cannot use this method to determine if a string will be visible when printed or how long it will appear.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- lengthOfBytesUsingEncoding:` (page 1580)
`sizeWithAttributes:` (`NSString` Additions)

Related Sample Code
iSpend
People
Quartz Composer WWDC 2005 TextEdit
StickiesExample
VertexPerformanceTest

Declared In
`NSString.h`

lengthOfBytesUsingEncoding:

Returns the number of bytes required to store the receiver in a given encoding.

```
- (NSUInteger)lengthOfBytesUsingEncoding:(NSStringEncoding)enc
```

Parameters
enc
The encoding for which to determine the receiver's length.

Return Value
The number of bytes required to store the receiver in the encoding *enc* in a non-external representation. The length does not include space for a terminating NULL character.

Discussion
The result is exact and is returned in O(n) time.

Availability
Available in Mac OS X v10.4 and later.

See Also
- maximumLengthOfBytesUsingEncoding: (page 1584)
- length (page 1580)

Related Sample Code
Core Data HTML Store

Declared In
NSString.h

lineRangeForRange:

Returns the range of characters representing the line or lines containing a given range.

```
- (NSRange)lineRangeForRange:(NSRange)aRange
```

Parameters
aRange
A range within the receiver.

Return Value
The range of characters representing the line or lines containing *aRange*, including the line termination characters.

Availability
Available in Mac OS X v10.0 and later.

See Also
- paragraphRangeForRange: (page 1585)
- getLineStart:end:contentsEnd:forRange: (page 1560)
- substringWithRange: (page 1611)

Related Sample Code
iSpend
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
NSString.h

localizedCaseInsensitiveCompare:

Returns an `NSComparisonResult` value that indicates the lexical ordering of the receiver and a given string using a case-insensitive, localized, comparison.

```
- (NSComparisonResult)localizedCaseInsensitiveCompare:(NSString *)aString
```

Parameters

aString

The string with which to compare the receiver.

This value must not be `nil`. If this value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

Return Value

`NSOrderedAscending` the receiver precedes *aString* in lexical ordering, `NSOrderedSame` the receiver and *aString* are equivalent in lexical value, and `NSOrderedDescending` if the receiver follows *aString*.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- compare:options:range:locale:` (page 1544)

Related Sample Code

NewsReader

Declared In

`NSString.h`

localizedCompare:

Returns an `NSComparisonResult` value that indicates the lexical ordering of the receiver and another given string using a localized comparison.

```
- (NSComparisonResult)localizedCompare:(NSString *)aString
```

Parameters

aString

The string with which to compare the receiver.

This value must not be `nil`. If this value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

Return Value

`NSOrderedAscending` the receiver precedes *string* in lexical ordering, `NSOrderedSame` the receiver and *string* are equivalent in lexical value, and `NSOrderedDescending` if the receiver follows *string*.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- compare:options:range:locale:` (page 1544)

Declared In

`NSString.h`

longLongValue

Returns the `long long` value of the receiver's text.

```
- (long long)longLongValue
```

Return Value
The `long long` value of the receiver's text, assuming a decimal representation and skipping whitespace at the beginning of the string. Returns `LLONG_MAX` or `LLONG_MIN` on overflow. Returns `0` if the receiver doesn't begin with a valid decimal text representation of a number.

Discussion
This method uses formatting information stored in the non-localized value; use an `NSScanner` object for localized scanning of numeric values from a string.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `doubleValue` (page 1552)
- `floatValue` (page 1553)
- `scanInt:` (page 1353) (`NSScanner`)

Declared In
`NSString.h`

lossyCString

Returns a representation of the receiver as a C string in the default C-string encoding, possibly losing information in converting to that encoding. (Deprecated in Mac OS X v10.4. Use cStringUsingEncoding: (page 1549) or `dataUsingEncoding:allowLossyConversion:` (page 1550) instead.)

```
- (const char *)lossyCString
```

Discussion
This method does not raise an exception if the conversion is lossy. The returned C string will be automatically freed just as a returned object would be released; your code should copy the C string or use `getCString:` (page 1556) if it needs to store the C string outside of the autorelease context in which the C string is created.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
- `cStringUsingEncoding:` (page 1549)
- `dataUsingEncoding:allowLossyConversion:` (page 1550)

Declared In
`NSString.h`

lowercaseString

Returns lowercased representation of the receiver.

```
- (NSString *)lowercaseString
```

Return Value
A string with each character from the receiver changed to its corresponding lowercase value.

Discussion
Case transformations aren't guaranteed to be symmetrical or to produce strings of the same lengths as the originals. The result of this statement:

```
lcString = [myString lowercaseString];
```

might not be equal to this statement:

```
lcString = [[myString uppercaseString] lowercaseString];
```

For example, the uppercase form of "ß" in German is "SS", so converting "Straße" to uppercase, then lowercase, produces this sequence of strings:

"Straße"
"STRASSE"
"strasse"

Availability
Available in Mac OS X v10.0 and later.

See Also
- `capitalizedString` (page 1539)
- `uppercaseString` (page 1611)

Related Sample Code
NewsReader
People
Quartz Composer WWDC 2005 TextEdit
StickiesExample
TextEditPlus

Declared In
`NSString.h`

maximumLengthOfBytesUsingEncoding:

Returns the maximum number of bytes needed to store the receiver in a given encoding.

```
- (NSUInteger)maximumLengthOfBytesUsingEncoding:(NSStringEncoding)enc
```

Parameters
enc
The encoding for which to determine the receiver's length.

Return Value
The maximum number of bytes needed to store the receiver in *encoding* in a non-external representation. The length does not include space for a terminating NULL character.

Discussion
The result is an estimate and is returned in O(1) time; the estimate may be considerably greater than the actual length needed.

Availability
Available in Mac OS X v10.4 and later.

See Also
- lengthOfBytesUsingEncoding: (page 1580)
- length (page 1580)

Declared In
NSString.h

paragraphRangeForRange:

Returns the range of characters representing the paragraph or paragraphs containing a given range.

```
- (NSRange)paragraphRangeForRange:(NSRange)aRange
```

Parameters
aRange
A range within the receiver. The range must not exceed the bounds of the receiver.

Return Value
The range of characters representing the paragraph or paragraphs containing *aRange*, including the paragraph termination characters.

Availability
Available in Mac OS X v10.3 and later.

See Also
- getParagraphStart:end:contentsEnd:forRange: (page 1561)
- lineRangeForRange: (page 1581)

Declared In
NSString.h

pathComponents

Returns an array of NSString objects containing, in order, each path component of the receiver.

```
- (NSArray *)pathComponents
```

Return Value
An array of NSString objects containing, in order, each path component of the receiver.

Discussion
The strings in the array appear in the order they did in the receiver. If the string begins or ends with the path separator, then the first or last component, respectively, will contain the separator. Empty components (caused by consecutive path separators) are deleted. For example, this code excerpt:

```
NSString *path = @"tmp/scratch";
NSArray *pathComponents = [path pathComponents];
```

produces an array with these contents:

Index	Path Component
0	"tmp"
1	"scratch"

If the receiver begins with a slash—for example, "/tmp/scratch"—the array has these contents:

Index	Path Component
0	"/"
1	"tmp"
2	"scratch"

If the receiver has no separators—for example, "scratch"—the array contains the string itself, in this case "scratch".

Note that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in Mac OS X v10.0 and later.

See Also
+ pathWithComponents: (page 1529)
- stringByStandardizingPath (page 1607)
- componentsSeparatedByString: (page 1547)

Related Sample Code
CoreRecipes
CustomSave
ObjectPath

Declared In
NSPathUtilities.h

pathExtension

Interprets the receiver as a path and returns the receiver's extension, if any.

```
- (NSString *)pathExtension
```

Return Value
The receiver's extension, if any (not including the extension divider).

Discussion
The following table illustrates the effect of `pathExtension` on a variety of different paths:

Receiver's String Value	String Returned
"/tmp/scratch.tiff"	"tiff"
"/tmp/scratch"	"" (an empty string)
"/tmp/"	"" (an empty string)
"/tmp/scratch..tiff"	"tiff"

Note that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
GLChildWindowDemo
Quartz Composer WWDC 2005 TextEdit
Sketch-112
StickiesExample
TextEditPlus

Declared In
`NSPathUtilities.h`

precomposedStringWithCanonicalMapping

Returns a string made by normalizing the receiver's contents using Form C.

```
- (NSString *)precomposedStringWithCanonicalMapping
```

Return Value
A string made by normalizing the receiver's contents using the Unicode Normalization Form C.

Availability
Available in Mac OS X v10.2 and later.

See Also
- `precomposedStringWithCompatibilityMapping` (page 1588)
- `decomposedStringWithCanonicalMapping` (page 1551)

Declared In
`NSString.h`

precomposedStringWithCompatibilityMapping

Returns a string made by normalizing the receiver's contents using Form KC.

```
- (NSString *)precomposedStringWithCompatibilityMapping
```

Return Value
A string made by normalizing the receiver's contents using the Unicode Normalization Form KC.

Availability
Available in Mac OS X v10.2 and later.

See Also
- `precomposedStringWithCanonicalMapping` (page 1587)
- `decomposedStringWithCompatibilityMapping` (page 1551)

Declared In
`NSString.h`

propertyList

Parses the receiver as a text representation of a property list, returning an `NSString`, `NSData`, `NSArray`, or `NSDictionary` object, according to the topmost element.

```
- (id)propertyList
```

Return Value
A property list representation of returning an `NSString`, `NSData`, `NSArray`, or `NSDictionary` object, according to the topmost element.

Discussion
The receiver must contain a string in a property list format. For a discussion of property list formats, see *Property List Programming Guide*.

Important: Raises an `NSParseErrorException` if the receiver cannot be parsed as a property list.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `propertyListFromStringsFileFormat` (page 1588)
+ `stringWithContentsOfFile:` (page 1530)

Declared In
`NSString.h`

propertyListFromStringsFileFormat

Returns a dictionary object initialized with the keys and values found in the receiver.

```
- (NSDictionary *)propertyListFromStringsFileFormat
```

Return Value
A dictionary object initialized with the keys and values found in the receiver

Discussion
The receiver must contain text in the format used for `.strings` files. In this format, keys and values are separated by an equal sign, and each key-value pair is terminated with a semicolon. The value is optional—if not present, the equal sign is also omitted. The keys and values themselves are always strings enclosed in straight quotation marks. Comments may be included, delimited by `/*` and `*/` as for ANSI C comments. Here's a short example of a strings file:

```
/* Question in confirmation panel for quitting. */
"Confirm Quit" = "Are you sure you want to quit?";

/* Message when user tries to close unsaved document */
"Close or Save" = "Save changes before closing?";

/* Word for Cancel */
"Cancel";
```

Availability
Available in Mac OS X v10.0 and later.

See Also
`- propertyList` (page 1588)
`+ stringWithContentsOfFile:` (page 1530)

Declared In
`NSString.h`

rangeOfCharacterFromSet:

Finds and returns the range in the receiver of the first character from a given character set.

```
- (NSRange)rangeOfCharacterFromSet:(NSCharacterSet *)aSet
```

Parameters

aSet
A character set. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aSet* is `nil`.

Return Value
The range in the receiver of the first character found from *aSet*. Returns a range of `{NSNotFound, 0}` if none of the characters in *aSet* are found.

Discussion
Invokes `rangeOfCharacterFromSet:options:` (page 1590) with no options.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSString.h`

rangeOfCharacterFromSet:options:

Finds and returns the range in the receiver of the first character, using given options, from a given character set.

```
- (NSRange)rangeOfCharacterFromSet:(NSCharacterSet *)aSet
    options:(NSStringCompareOptions)mask
```

Parameters

aSet

A character set. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aSet* is `nil`.

mask

A mask specifying search options. The following options may be specified by combining them with the C bitwise `OR` operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSBackwardsSearch`. See *String Programming Guide for Cocoa* for details on these options.

Return Value

The range in the receiver of the first character found from *aSet*. Returns a range of `{NSNotFound, 0}` if none of the characters in *aSet* are found.

Discussion

Invokes `rangeOfCharacterFromSet:options:range:` (page 1590) with *mask* for the options and the entire extent of the receiver for the range.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSString.h`

rangeOfCharacterFromSet:options:range:

Finds and returns the range in the receiver of the first character from a given character set found in a given range with given options.

```
- (NSRange)rangeOfCharacterFromSet:(NSCharacterSet *)aSet
    options:(NSStringCompareOptions)mask range:(NSRange)aRange
```

Parameters

aSet

A character set. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aSet* is `nil`.

mask

A mask specifying search options. The following options may be specified by combining them with the C bitwise `OR` operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSBackwardsSearch`. See *String Programming Guide for Cocoa* for details on these options.

aRange
The range in which to search. *aRange* must not exceed the bounds of the receiver.

Important: Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the string.

Return Value
The range in the receiver of the first character found from *aSet* within *aRange*. Returns a range of `{NSNotFound, 0}` if none of the characters in *aSet* are found.

Discussion
Because pre-composed characters in *aSet* can match composed character sequences in the receiver, the length of the returned range can be greater than `1`. For example, if you search for "ü" in the string "stru¨del", the returned range is `{3,2}`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
VertexPerformanceTest

Declared In
`NSString.h`

rangeOfComposedCharacterSequenceAtIndex:

Returns the range in the receiver of the composed character sequence located at a given index.

```
- (NSRange)rangeOfComposedCharacterSequenceAtIndex:(NSUInteger)anIndex
```

Parameters
anIndex
The index of a character in the receiver. The value must not exceed the bounds of the receiver.

Return Value
The range in the receiver of the composed character sequence located at *anIndex*.

Discussion
The composed character sequence includes the first base character found at or before *anIndex*, and its length includes the base character and all non-base characters following the base character.

If you want to write a method to adjust an arbitrary range so it includes the composed character sequences on its boundaries, you can create a method such as the following:

```
- (NSRange)adjustRange:(NSRange)aRange
{
    NSUInteger index, endIndex;
    NSRange newRange, endRange;

    // Check for validity of range
    if ( aRange.location >= [self length] ||
         aRange.location + aRange.length > [self length] )
    {
        [NSException raise:NSRangeException format:@"Invalid  range %@.",
        NSStringFromRange(aRange)];
```

```
    }

    index = aRange.location;
    newRange = [self rangeOfComposedCharacterSequenceAtIndex:index];

    index = aRange.location + aRange.length - 1;
    endRange = [self rangeOfComposedCharacterSequenceAtIndex:index];
    endIndex = endRange.location + endRange.length;

    newRange.length = endIndex - newRange.location;

    return newRange;
}
```

First, `adjustRange:` corrects the location for the beginning of *aRange*, storing it in *newRange*. It then works at the end of *aRange*, correcting the location and storing it in *endIndex*. Finally, it sets the length of *newRange* to the difference between *endIndex* and the new range's location.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `rangeOfComposedCharacterSequencesForRange:` (page 1592)

Declared In
`NSString.h`

rangeOfComposedCharacterSequencesForRange:

Returns the range in the receiver of the composed character sequence in a given range.

```
- (NSRange)rangeOfComposedCharacterSequencesForRange:(NSRange)range
```

Parameters

range

A range in the receiver. The range must not exceed the bounds of the receiver.

Return Value
The range in the receiver of the composed character sequence in *range*.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `rangeOfComposedCharacterSequenceAtIndex:` (page 1591)

Declared In
`NSString.h`

rangeOfString:

Finds and returns the range of the first occurrence of a given string within the receiver.

```
- (NSRange)rangeOfString:(NSString *)aString
```

Parameters

aString

The string to search for. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aString* is `nil`.

Return Value

An `NSRange` structure giving the location and length in the receiver of the first occurrence of *aString*. Returns `{NSNotFound, 0}` if *aString* is not found or is empty (@"").

Discussion

Invokes `rangeOfString:options:` (page 1593) with no options.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

People

QTSSConnectionMonitor

QTSSInspector

Declared In

`NSString.h`

rangeOfString:options:

Finds and returns the range of the first occurrence of a given string within the receiver, subject to given options.

```
- (NSRange)rangeOfString:(NSString *)aString options:(NSStringCompareOptions)mask
```

Parameters

aString

The string to search for. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aString* is `nil`.

mask

A mask specifying search options. The following options may be specified by combining them with the C bitwise `OR` operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSBackwardsSearch`, `NSAnchoredSearch`. See *String Programming Guide for Cocoa* for details on these options.

Return Value

An `NSRange` structure giving the location and length in the receiver of the first occurrence of *aString*, modulo the options in *mask*. Returns `{NSNotFound, 0}` if *aString* is not found or is empty (@"").

Discussion

Invokes `rangeOfString:options:range:` (page 1594) with the options specified by *mask* and the entire extent of the receiver as the range.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
Sketch-112

Declared In
`NSString.h`

rangeOfString:options:range:

Finds and returns the range of the first occurrence of a given string, within the given range of the receiver, subject to given options.

```
- (NSRange)rangeOfString:(NSString *)aString options:(NSStringCompareOptions)mask
    range:(NSRange)aRange
```

Parameters

aString

The string for which to search. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aString* is `nil`.

mask

A mask specifying search options. The following options may be specified by combining them with the C bitwise `OR` operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSBackwardsSearch`, and `NSAnchoredSearch`. See *String Programming Guide for Cocoa* for details on these options.

aRange

The range within the receiver for which to search for *aString*.

Important: Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the string.

Return Value
An `NSRange` structure giving the location and length in the receiver of *aString* within *aRange* in the receiver, modulo the options in *mask*. The range returned is relative to the start of the string, not to the passed-in range. Returns `{NSNotFound, 0}` if *aString* is not found or is empty (@"").

Discussion
The length of the returned range and that of *aString* may differ if equivalent composed character sequences are matched.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
VertexPerformanceTest

Declared In
`NSString.h`

rangeOfString:options:range:locale:

Finds and returns the range of the first occurrence of a given string within a given range of the receiver, subject to given options, using the specified locale, if any.

```
- (NSRange)rangeOfString:(NSString *)aString options:(NSStringCompareOptions)mask
    range:(NSRange)searchRange locale:(NSLocale *)locale
```

Parameters

aString

The string for which to search. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aString* is `nil`.

mask

A mask specifying search options. The following options may be specified by combining them with the C bitwise `OR` operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSBackwardsSearch`, and `NSAnchoredSearch`. See *String Programming Guide for Cocoa* for details on these options.

aRange

The range within the receiver for which to search for *aString*.

Important: Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the string.

locale

The locale to use when comparing the receiver with *aString*. If this value is `nil`, uses the current locale.

The locale argument affects the equality checking algorithm. For example, for the Turkish locale, case-insensitive compare matches "I" to "ı" (Unicode code point U+0131, Latin Small Dotless I), not the normal "i" character.

Return Value

An `NSRange` structure giving the location and length in the receiver of *aString* within *aRange* in the receiver, modulo the options in *mask*. The range returned is relative to the start of the string, not to the passed-in range. Returns `{NSNotFound, 0}` if *aString* is not found or is empty (@"").

Discussion

The length of the returned range and that of *aString* may differ if equivalent composed character sequences are matched.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSString.h`

smallestEncoding

Returns the smallest encoding to which the receiver can be converted without loss of information.

```
- (NSStringEncoding)smallestEncoding
```

Return Value
The smallest encoding to which the receiver can be converted without loss of information.

Discussion
The returned encoding may not be the fastest for accessing characters, but is space-efficient. This method may take some time to execute.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `fastestEncoding` (page 1552)
- `getCharacters:range:` (page 1555)

Declared In
`NSString.h`

stringByAbbreviatingWithTildeInPath

Returns a new string representing the receiver as a path with a tilde (~) substituted for the full path to the current user's home directory.

```
- (NSString *)stringByAbbreviatingWithTildeInPath
```

Return Value
A new string representing the receiver as a path with a tilde (~) substituted for the full path to the current user's home directory. Returns a new string matching the receiver if the receiver doesn't begin with a user's home directory.

Discussion
Note that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in Mac OS X v10.0 and later.

See Also
- `stringByExpandingTildeInPath` (page 1602)

Related Sample Code
SimpleDownload

Declared In
`NSPathUtilities.h`

stringByAddingPercentEscapesUsingEncoding:

Returns a representation of the receiver using a given encoding to determine the percent escapes necessary to convert the receiver into a legal URL string.

```
- (NSString *)stringByAddingPercentEscapesUsingEncoding:(NSStringEncoding)encoding
```

Parameters

encoding

The encoding to use for the returned string.

Return Value

A representation of the receiver using *encoding* to determine the percent escapes necessary to convert the receiver into a legal URL string. Returns `nil` if *encoding* cannot encode a particular character

Discussion

See `CFURLCreateStringByAddingPercentEscapes` for more complex transformations.

Availability

Available in Mac OS X v10.3 and later.

See Also

`- stringByReplacingPercentEscapesUsingEncoding:` (page 1606)

Declared In

`NSURL.h`

stringByAppendingFormat:

Returns a string made by appending to the receiver a string constructed from a given format string and the following arguments.

```
- (NSString *)stringByAppendingFormat:(NSString *)format ...
```

Parameters

format

A format string. See Formatting String Objects for more information. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *format* is `nil`.

...

A comma-separated list of arguments to substitute into *format*.

Return Value

A string made by appending to the receiver a string constructed from *format* and the following arguments, in the manner of `stringWithFormat:` (page 1536).

Availability

Available in Mac OS X v10.0 and later.

See Also

`- stringByAppendingString:` (page 1599)

Related Sample Code

Departments and Employees

QTMetadataEditor

Declared In

`NSString.h`

stringByAppendingPathComponent:

Returns a new string made by appending to the receiver a given string.

```
- (NSString *)stringByAppendingPathComponent:(NSString *)aString
```

Parameters

aString

The path component to append to the receiver.

Return Value

A new string made by appending *aString* to the receiver, preceded if necessary by a path separator.

Discussion

The following table illustrates the effect of this method on a variety of different paths, assuming that *aString* is supplied as "scratch.tiff":

Receiver's String Value	Resulting String
"/tmp"	"/tmp/scratch.tiff"
"/tmp/"	"/tmp/scratch.tiff"
"/"	"/scratch.tiff"
"" (an empty string)	"scratch.tiff"

Note that this method only works with file paths (not, for example, string representations of URLs).

Availability

Available in Mac OS X v10.0 and later.

See Also

- stringsByAppendingPaths: (page 1609)
- stringByAppendingPathExtension: (page 1598)
- stringByDeletingLastPathComponent (page 1600)

Related Sample Code

Core Data HTML Store
CoreRecipes
Quartz Composer WWDC 2005 TextEdit
StickiesExample
TextEditPlus

Declared In

NSPathUtilities.h

stringByAppendingPathExtension:

Returns a new string made by appending to the receiver an extension separator followed by a given extension.

```
- (NSString *)stringByAppendingPathExtension:(NSString *)ext
```

Parameters

ext

The extension to append to the receiver.

Return Value

A new string made by appending to the receiver an extension separator followed by *ext*.

Discussion

The following table illustrates the effect of this method on a variety of different paths, assuming that *ext* is supplied as `@"tiff"`:

Receiver's String Value	Resulting String
"/tmp/scratch.old"	"/tmp/scratch.old.tiff"
"/tmp/scratch."	"/tmp/scratch..tiff"
"/tmp/"	"/tmp.tiff"
"scratch"	"scratch.tiff"

Note that adding an extension to `@"/tmp/"` causes the result to be `@"/tmp.tiff"` instead of `@"/tmp/.tiff"`. This difference is because a file named `@".tiff"` is not considered to have an extension, so the string is appended to the last nonempty path component.

This method does not allow you to append file extensions to filenames starting with the tilde character (~).

Note that this method only works with file paths (not, for example, string representations of URLs).

Availability

Available in Mac OS X v10.0 and later.

See Also

`- stringByAppendingPathComponent:` (page 1598)

`- stringByDeletingPathExtension` (page 1601)

Related Sample Code

QTRecorder

Quartz Composer WWDC 2005 TextEdit

SpotlightFortunes

TextEditPlus

WhackedTV

Declared In

`NSPathUtilities.h`

stringByAppendingString:

Returns a new string made by appending a given string to the receiver.

```
- (NSString *)stringByAppendingString:(NSString *)aString
```

Parameters

aString

The string to append to the receiver. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aString* is `nil`.

Return Value

A new string made by appending *aString* to the receiver.

Discussion

This code excerpt, for example:

```
NSString *errorTag = @"Error: ";
NSString *errorString = @"premature end of file.";
NSString *errorMessage = [errorTag stringByAppendingString:errorString];
```

produces the string "`Error: premature end of file.`".

Availability

Available in Mac OS X v10.0 and later.

See Also

`- stringByAppendingFormat:` (page 1597)

Related Sample Code

CocoaDVDPlayer

NumberInput_IMKit_Sample

QTSSConnectionMonitor

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In

`NSString.h`

stringByDeletingLastPathComponent

Returns a new string made by deleting the last path component from the receiver, along with any final path separator.

```
- (NSString *)stringByDeletingLastPathComponent
```

Return Value

A new string made by deleting the last path component from the receiver, along with any final path separator. If the receiver represents the root path it is returned unaltered.

Discussion

The following table illustrates the effect of this method on a variety of different paths:

Receiver's String Value	Resulting String
"/tmp/scratch.tiff"	"/tmp"

Receiver's String Value	Resulting String
"/tmp/lock/"	"/tmp"
"/tmp/"	"/"
"/tmp"	"/"
"/"	"/"
"scratch.tiff"	"" (an empty string)

Note that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in Mac OS X v10.0 and later.

See Also
- `stringByDeletingPathExtension` (page 1601)
- `stringByAppendingPathComponent:` (page 1598)

Related Sample Code
ExtractMovieAudioToAIFF
LSMSmartCategorizer
Quartz Composer WWDC 2005 TextEdit
TextEditPlus
WhackedTV

Declared In
`NSPathUtilities.h`

stringByDeletingPathExtension

Returns a new string made by deleting the extension (if any, and only the last) from the receiver.

```
- (NSString *)stringByDeletingPathExtension
```

Return Value
a new string made by deleting the extension (if any, and only the last) from the receiver. Strips any trailing path separator before checking for an extension. If the receiver represents the root path, it is returned unaltered.

Discussion
The following table illustrates the effect of this method on a variety of different paths:

Receiver's String Value	Resulting String
"/tmp/scratch.tiff"	"/tmp/scratch"
"/tmp/"	"/tmp"
"scratch.bundle/"	"scratch"

Receiver's String Value	Resulting String
"scratch..tiff"	"scratch."
".tiff"	".tiff"
"/"	"/"

Note that attempting to delete an extension from @".tiff" causes the result to be @".tiff" instead of an empty string. This difference is because a file named @".tiff" is not considered to have an extension, so nothing is deleted. Note also that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in Mac OS X v10.0 and later.

See Also
- pathExtension (page 1586)
- stringByDeletingLastPathComponent (page 1600)

Related Sample Code
AutoUpdater
EnhancedAudioBurn
QTAudioExtractionPanel
Reducer
StickiesExample

Declared In
NSPathUtilities.h

stringByExpandingTildeInPath

Returns a new string made by expanding the initial component of the receiver to its full path value.

```
- (NSString *)stringByExpandingTildeInPath
```

Return Value
A new string made by expanding the initial component of the receiver, if it begins with "~" or "~user", to its full path value. Returns a new string matching the receiver if the receiver's initial component can't be expanded.

Discussion
Note that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in Mac OS X v10.0 and later.

See Also
- stringByAbbreviatingWithTildeInPath (page 1596)

Related Sample Code
MyPhoto

Quartz Composer Offline Rendering
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus

Declared In
`NSPathUtilities.h`

stringByFoldingWithOptions:locale:

Returns a string with the given character folding options applied.

```
- (NSString *)stringByFoldingWithOptions:(NSStringCompareOptions)options
    locale:(NSLocale *)locale
```

Parameters

options

A mask of compare flags with a suffix `InsensitiveSearch`.

locale

The locale to use for the folding. The locale affects the folding logic. For example, for the Turkish locale, case-insensitive compare matches “I” to “ı” (Unicode code point U+0131, Latin Small Dotless I), not the normal “i” character.

Return Value
A string with the character folding options applied.

Discussion
Character folding operations remove distinctions between characters. For example, case folding may replace uppercase letters with their lowercase equivalents.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSString.h`

stringByPaddingToLength:withString:startingAtIndex:

Returns a new string formed from the receiver by either removing characters from the end, or by appending as many occurrences as necessary of a given pad string.

```
- (NSString *)stringByPaddingToLength:(NSUInteger)newLength withString:(NSString
    *)padString startingAtIndex:(NSUInteger)padIndex
```

Parameters

newLength

The new length for the receiver.

padString

The string with which to extend the receiver.

padIndex

The index in *padString* from which to start padding.

Return Value
A new string formed from the receiver by either removing characters from the end, or by appending as many occurrences of *padString* as necessary.

Discussion
Here are some examples of usage:

```
[@"abc" stringByPaddingToLength: 9 withString: @"." startingAtIndex:0];
    // Results in "abc......"

[@"abc" stringByPaddingToLength: 2 withString: @"." startingAtIndex:0];
    // Results in "ab"

[@"abc" stringByPaddingToLength: 9 withString: @". " startingAtIndex:1];
    // Results in "abc . . ."
    // Notice that the first character in the padding is " "
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
NSString.h

stringByReplacingCharactersInRange:withString:

Returns a new string in which the characters in a specified range of the receiver are replaced by a given string.

```
- (NSString *)stringByReplacingCharactersInRange:(NSRange)range withString:(NSString
    *)replacement
```

Parameters

range
A range of characters in the receiver.

replacement
The string with which to replace the characters in *range*.

Return Value
A new string in which the characters in *range* of the receiver are replaced by *replacement*.

Availability
Available in Mac OS X v10.5 and later.

See Also
- stringByReplacingOccurrencesOfString:withString: (page 1605)
- stringByReplacingOccurrencesOfString:withString:options:range: (page 1605)
- stringByReplacingPercentEscapesUsingEncoding: (page 1606)

Declared In
NSString.h

stringByReplacingOccurrencesOfString:withString:

Returns a new string in which all occurrences of a target string in the receiver are replaced by another given string.

```
- (NSString *)stringByReplacingOccurrencesOfString:(NSString *)target
    withString:(NSString *)replacement
```

Parameters

target
The string to replace.

replacement
The string with which to replace *target*.

Return Value
A new string in which all occurrences of *target* in the receiver are replaced by *replacement*.

Discussion
Invokes `stringByReplacingOccurrencesOfString:withString:options:range:` (page 1605)with `0` options and range of the whole string.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `stringByReplacingOccurrencesOfString:withString:options:range:` (page 1605)
- `stringByReplacingCharactersInRange:withString:` (page 1604)
- `stringByReplacingPercentEscapesUsingEncoding:` (page 1606)

Declared In
`NSString.h`

stringByReplacingOccurrencesOfString:withString:options:range:

Returns a new string in which all occurrences of a target string in a specified range of the receiver are replaced by another given string.

```
- (NSString *)stringByReplacingOccurrencesOfString:(NSString *)target
    withString:(NSString *)replacement options:(NSStringCompareOptions)options
    range:(NSRange)searchRange
```

Parameters

target
The string to replace.

replacement
The string with which to replace *target*.

options
A mask of options to use when comparing *target* with the receiver. Pass `0` to specify no options.

searchRange
The range in the receiver in which to search for *target*.

Return Value
A new string in which all occurrences of *target*, matched using *options*, in *searchRange* of the receiver are replaced by *replacement*.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `stringByReplacingOccurrencesOfString:withString:` (page 1605)
- `stringByReplacingCharactersInRange:withString:` (page 1604)
- `stringByReplacingPercentEscapesUsingEncoding:` (page 1606)

Declared In
`NSString.h`

stringByReplacingPercentEscapesUsingEncoding:

Returns a new string made by replacing in the receiver all percent escapes with the matching characters as determined by a given encoding.

```
- (NSString *)stringByReplacingPercentEscapesUsingEncoding:(NSStringEncoding)encoding
```

Parameters
encoding
The encoding to use for the returned string.

Return Value
A new string made by replacing in the receiver all percent escapes with the matching characters as determined by the given encoding *encoding*. Returns `nil` if the transformation is not possible, for example, the percent escapes give a byte sequence not legal in *encoding*.

Discussion
See `CFURLCreateStringByReplacingPercentEscapes` for more complex transformations.

Availability
Available in Mac OS X v10.3 and later.

See Also
- `stringByAddingPercentEscapesUsingEncoding:` (page 1596)

Declared In
`NSURL.h`

stringByResolvingSymlinksInPath

Returns a new string made from the receiver by resolving all symbolic links and standardizing path.

```
- (NSString *)stringByResolvingSymlinksInPath
```

Return Value
A new string made by expanding an initial tilde expression in the receiver, then resolving all symbolic links and references to current or parent directories if possible, to generate a standardized path. If the original path is absolute, all symbolic links are guaranteed to be removed; if it's a relative path, symbolic links that can't be resolved are left unresolved in the returned string. Returns `self` if an error occurs.

Discussion
If the name of the receiving path begins with `/private`, the `stringByResolvingSymlinksInPath` method strips off the `/private` designator, provided the result is the name of an existing file.

Note that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in Mac OS X v10.0 and later.

See Also
- `stringByStandardizingPath` (page 1607)
- `stringByExpandingTildeInPath` (page 1602)

Related Sample Code
CoreRecipes
DeskPictAppDockMenu
PredicateEditorSample
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSPathUtilities.h`

stringByStandardizingPath

Returns a new string made by removing extraneous path components from the receiver.

```
- (NSString *)stringByStandardizingPath
```

Return Value
A new string made by removing extraneous path components from the receiver.

Discussion
If `stringByStandardizingPath` detects symbolic links in a pathname, the `stringByResolvingSymlinksInPath` (page 1606) method is called to resolve them. If an invalid pathname is provided, `stringByStandardizingPath` may attempt to resolve it by calling `stringByResolvingSymlinksInPath`, and the results are undefined. If any other kind of error is encountered (such as a path component not existing), `self` is returned.

This method can make the following changes in the provided string:

- Expand an initial tilde expression using `stringByExpandingTildeInPath` (page 1602).
- Reduce empty components and references to the current directory (that is, the sequences "//" and "/./") to single path separators.

- In absolute paths only, resolve references to the parent directory (that is, the component "..") to the real parent directory if possible using `stringByResolvingSymlinksInPath` (page 1606), which consults the file system to resolve each potential symbolic link.

  In relative paths, because symbolic links can't be resolved, references to the parent directory are left in place.
- Remove an initial component of "`/private`" from the path if the result still indicates an existing file or directory (checked by consulting the file system).

Note that the path returned by this method may still have symbolic link components in it. Note also that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in Mac OS X v10.0 and later.

See Also
- `stringByExpandingTildeInPath` (page 1602)
- `stringByResolvingSymlinksInPath` (page 1606)

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus

Declared In
`NSPathUtilities.h`

stringByTrimmingCharactersInSet:

Returns a new string made by removing from both ends of the receiver characters contained in a given character set.

```
- (NSString *)stringByTrimmingCharactersInSet:(NSCharacterSet *)set
```

Parameters

set

A character set containing the characters to remove from the receiver. *set* must not be `nil`.

Return Value
A new string made by removing from both ends of the receiver characters contained in *set*. If the receiver is composed entirely of characters from *set*, the empty string is returned.

Discussion
Use `whitespaceCharacterSet` (page 252) or `whitespaceAndNewlineCharacterSet` (page 252) to remove whitespace around strings.

Availability
Available in Mac OS X v10.2 and later.

See Also
- `componentsSeparatedByCharactersInSet:` (page 1546)

Related Sample Code
CoreRecipes
iSpend
TextLinks

Declared In
`NSString.h`

stringsByAppendingPaths:

Returns an array of strings made by separately appending to the receiver each string in in a given array.

```
- (NSArray *)stringsByAppendingPaths:(NSArray *)paths
```

Parameters

paths
An array of `NSString` objects specifying paths to add to the receiver.

Return Value
An array of `NSString` objects made by separately appending each string in *paths* to the receiver, preceded if necessary by a path separator.

Discussion
Note that this method only works with file paths (not, for example, string representations of URLs). See `stringByAppendingPathComponent:` (page 1598) for an individual example.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSPathUtilities.h`

substringFromIndex:

Returns a new string containing the characters of the receiver from the one at a given index to the end.

```
- (NSString *)substringFromIndex:(NSUInteger)anIndex
```

Parameters

anIndex
An index. The value must lie within the bounds of the receiver, or be equal to the length of the receiver.

Important: Raises an `NSRangeException` if *anIndex* lies beyond the end of the receiver.

Return Value
A new string containing the characters of the receiver from the one at *anIndex* to the end. If *anIndex* is equal to the length of the string, returns an empty string.

Availability
Available in Mac OS X v10.0 and later.

See Also
- substringWithRange: (page 1611)
- substringToIndex: (page 1610)

Related Sample Code
Birthdays
Core Data HTML Store
NewsReader
Reminders
Sketch-112

Declared In
NSString.h

substringToIndex:

Returns a new string containing the characters of the receiver up to, but not including, the one at a given index.

```
- (NSString *)substringToIndex:(NSUInteger)anIndex
```

Parameters
anIndex
An index. The value must lie within the bounds of the receiver, or be equal to the length of the receiver.

Important: Raises an NSRangeException if (*anIndex* - 1) lies beyond the end of the receiver.

Return Value
A new string containing the characters of the receiver up to, but not including, the one at *anIndex*. If *anIndex* is equal to the length of the string, returns a copy of the receiver.

Availability
Available in Mac OS X v10.0 and later.

See Also
- substringFromIndex: (page 1609)
- substringWithRange: (page 1611)

Related Sample Code
DerivedProperty
People
Quartz Composer WWDC 2005 TextEdit
StickiesExample
TextEditPlus

Declared In
NSString.h

substringWithRange:

Returns a string object containing the characters of the receiver that lie within a given range.

```
- (NSString *)substringWithRange:(NSRange)aRange
```

Parameters

aRange

A range. The range must not exceed the bounds of the receiver.

Important: Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the receiver.

Return Value
A string object containing the characters of the receiver that lie within *aRange*.

Discussion
This method treats the length of the string as a valid range value that returns an empty string.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `substringFromIndex:` (page 1609)
- `substringToIndex:` (page 1610)

Related Sample Code
EnhancedDataBurn
iSpend
Quartz Composer WWDC 2005 TextEdit
TextEditPlus
VertexPerformanceTest

Declared In
`NSString.h`

uppercaseString

Returns an uppercased representation of the receiver.

```
- (NSString *)uppercaseString
```

Return Value
A string with each character from the receiver changed to its corresponding uppercase value.

Discussion
Case transformations aren't guaranteed to be symmetrical or to produce strings of the same lengths as the originals. See `lowercaseString` (page 1584) for an example.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `capitalizedString` (page 1539)

- lowercaseString (page 1584)

Related Sample Code
QTKitMovieShuffler
Worm

Declared In
NSString.h

UTF8String

Returns a null-terminated UTF8 representation of the receiver.

```
- (const char *)UTF8String
```

Return Value
A null-terminated UTF8 representation of the receiver.

Discussion
The returned C string is automatically freed just as a returned object would be released; you should copy the C string if it needs to store it outside of the autorelease context in which the C string is created.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
DynamicProperties
NameAndPassword

Declared In
NSString.h

writeToFile:atomically:

Writes the contents of the receiver to the file specified by a given path. (Deprecated in Mac OS X v10.4. Use writeToFile:atomically:encoding:error: (page 1613) instead.)

```
- (BOOL)writeToFile:(NSString *)path atomically:(BOOL)flag
```

Return Value
YES if the file is written successfully, otherwise NO.

Discussion
Writes the contents of the receiver to the file specified by *path* (overwriting any existing file at *path*). *path* is written in the default C-string encoding if possible (that is, if no information would be lost), in the Unicode encoding otherwise.

If *flag* is YES, the receiver is written to an auxiliary file, and then the auxiliary file is renamed to *path*. If *flag* is NO, the receiver is written directly to *path*. The YES option guarantees that *path*, if it exists at all, won't be corrupted even if the system should crash during writing.

If *path* contains a tilde (~) character, you must expand it with stringByExpandingTildeInPath (page 1602) before invoking this method.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
`- writeToFile:atomically:encoding:error:` (page 1613)

Related Sample Code
bMoviePalette
bMoviePaletteCocoa
Cropped Image
Monochrome Image
RGB Image

Declared In
`NSString.h`

writeToFile:atomically:encoding:error:

Writes the contents of the receiver to a file at a given path using a given encoding.

```
- (BOOL)writeToFile:(NSString *)path atomically:(BOOL)useAuxiliaryFile
    encoding:(NSStringEncoding)enc error:(NSError **)error
```

Parameters

path
The file to which to write the receiver. If *path* contains a tilde (~) character, you must expand it with `stringByExpandingTildeInPath` (page 1602) before invoking this method.

useAuxiliaryFile
If `YES`, the receiver is written to an auxiliary file, and then the auxiliary file is renamed to *path*. If `NO`, the receiver is written directly to *path*. The `YES` option guarantees that *path*, if it exists at all, won't be corrupted even if the system should crash during writing.

enc
The encoding to use for the output.

error
If there is an error, upon return contains an `NSError` object that describes the problem. If you are not interested in details of errors, you may pass in `NULL`.

Return Value
`YES` if the file is written successfully, otherwise `NO` (if there was a problem writing to the file or with the encoding).

Discussion
This method overwrites any existing file at *path*.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSString.h`

writeToURL:atomically:

Writes the contents of the receiver to the location specified by a given URL. (Deprecated in Mac OS X v10.4. Use `writeToURL:atomically:encoding:error:` (page 1614) instead.)

```
- (BOOL)writeToURL:(NSURL *)aURL atomically:(BOOL)atomically
```

Return Value
`YES` if the location is written successfully, otherwise `NO`.

Discussion
If *atomically* is `YES`, the receiver is written to an auxiliary location, and then the auxiliary location is renamed to *aURL*. If *atomically* is `NO`, the receiver is written directly to *aURL*. The `YES` option guarantees that *aURL*, if it exists at all, won't be corrupted even if the system should crash during writing.

The *atomically* parameter is ignored if *aURL* is not of a type that can be accessed atomically.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
`- writeToURL:atomically:encoding:error:` (page 1614)

Declared In
`NSString.h`

writeToURL:atomically:encoding:error:

Writes the contents of the receiver to the URL specified by *url* using the specified encoding.

```
- (BOOL)writeToURL:(NSURL *)url atomically:(BOOL)useAuxiliaryFile
    encoding:(NSStringEncoding)enc error:(NSError **)error
```

Parameters

url
The URL to which to write the receiver.

useAuxiliaryFile
If `YES`, the receiver is written to an auxiliary file, and then the auxiliary file is renamed to *url*. If `NO`, the receiver is written directly to *url*. The `YES` option guarantees that *url*, if it exists at all, won't be corrupted even if the system should crash during writing.

The *useAuxiliaryFile* parameter is ignored if *url* is not of a type that can be accessed atomically.

enc
The encoding to use for the output.

error
If there is an error, upon return contains an `NSError` object that describes the problem. If you are not interested in details of errors, you may pass in `NULL`.

Return Value
`YES` if the URL is written successfully, otherwise `NO` (if there was a problem writing to the URL or with the encoding).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSString.h`

Constants

unichar

Type for Unicode characters.

```
typedef unsigned short unichar;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSString.h`

NSMaximumStringLength

A constant to define the maximum number of characters in an `NSString` object. (**Deprecated.** This constant is not available in Mac OS X v10.5 and later.)

```
#define NSMaximumStringLength (INT_MAX-1)
```

Constants
`NSMaximumStringLength`
Maximum number of characters in an `NSString` object.
Available in Mac OS X v10.0 through Mac OS X v10.4.
Declared in `NSString.h`.

Availability
Available in Mac OS X v10.0.
Removed in Mac OS X v10.5.

Declared In
`NSString.h`

NSStringCompareOptions

Type for string comparison options.

```
typedef NSUInteger NSStringCompareOptions;
```

Discussion
See "Search and Comparison Options" (page 1616) for possible values.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSString.h`

Search and Comparison Options

These values represent the options available to many of the string classes' search and comparison methods.

```
enum {
    NSCaseInsensitiveSearch = 1,
    NSLiteralSearch = 2,
    NSBackwardsSearch = 4,
    NSAnchoredSearch = 8,
    NSNumericSearch = 64,
    NSDiacriticInsensitiveSearch = 128,
    NSWidthInsensitiveSearch = 256,
    NSForcedOrderingSearch = 512
};
```

Constants

`NSCaseInsensitiveSearch`
A case-insensitive search.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSLiteralSearch`
Exact character-by-character equivalence.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSBackwardsSearch`
Search from end of source string.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSAnchoredSearch`
Search is limited to start (or end, if `NSBackwardsSearch`) of source string.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSNumericSearch`
Numbers within strings are compared using numeric value, that is, `Foo2.txt` < `Foo7.txt` < `Foo25.txt`.

This option only applies to compare methods, not find.

Available in Mac OS X v10.3 and later.

Declared in `NSString.h`.

`NSDiacriticInsensitiveSearch`
Search ignores diacritic marks.

For example, ‘ö’ is equal to ‘o’.

Available in Mac OS X v10.5 and later.

Declared in `NSString.h`.

`NSWidthInsensitiveSearch`
Search ignores width differences in characters that have full-width and half-width forms, as occurs in East Asian character sets.

For example, with this option, the full-width Latin small letter 'a' (Unicode code point U+FF41) is equal to the basic Latin small letter 'a' (Unicode code point U+0061).

Available in Mac OS X v10.5 and later.

Declared in `NSString.h`.

`NSForcedOrderingSearch`
Comparisons are forced to return either `NSOrderedAscending` or `NSOrderedDescending` if the strings are equivalent but not strictly equal.

This option gives stability when sorting. For example, “aaa” is greater than "AAA” if `NSCaseInsensitiveSearch` is specified.

Available in Mac OS X v10.5 and later.

Declared in `NSString.h`.

Discussion
See Searching, Comparing, and Sorting Strings for details on the effects of these options.

Declared In
`NSString.h`

NSStringEncodingConversionOptions

Type for encoding conversion options.

```
typedef NSUInteger NSStringEncodingConversionOptions;
```

Discussion
See `NSStringEncodingConversionOptions` (page 1617) for possible values.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSString.h`

Encoding Conversion Options

Options for converting string encodings.

```
enum {
    NSStringEncodingConversionAllowLossy = 1,
    NSStringEncodingConversionExternalRepresentation = 2
};
```

Constants

`NSStringEncodingConversionAllowLossy`

Allows lossy conversion.

Available in Mac OS X v10.5 and later.

Declared in `NSString.h`.

`NSStringEncodingConversionExternalRepresentation`

Available in Mac OS X v10.5 and later.

Declared in `NSString.h`.

Special Considerations

These constants are available in Mac OS X v10.4; they are, however, differently named:

```
typedef enum {
    NSAllowLossyEncodingConversion = 1,
    NSExternalRepresentationEncodingConversion = 2
} NSStringEncodingConversionOptions;
```

You can use them on Mac OS X v10.4 if you define the symbols as `extern` constants.

Declared In

`NSString.h`

NSString Handling Exception Names

These constants define the names of exceptions raised if `NSString` cannot represent a string in a given encoding, or parse a string as a property list.

```
extern NSString *NSParseErrorException;
extern NSString *NSCharacterConversionException;
```

Constants

`NSCharacterConversionException`

`NSString` raises an `NSCharacterConversionException` if a string cannot be represented in a file-system or string encoding.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSParseErrorException`

`NSString` raises an `NSParseErrorException` if a string cannot be parsed as a property list.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

Declared In

`NSString.h`

NSStringEncoding

Type for string encoding.

```
typedef NSUInteger NSStringEncoding;
```

Discussion
See "String Encodings" (page 1619) for possible values.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSString.h`

String Encodings

The following constants are provided by `NSString` as possible string encodings.

```
enum {
    NSASCIIStringEncoding = 1,
    NSNEXTSTEPStringEncoding = 2,
    NSJapaneseEUCStringEncoding = 3,
    NSUTF8StringEncoding = 4,
    NSISOLatin1StringEncoding = 5,
    NSSymbolStringEncoding = 6,
    NSNonLossyASCIIStringEncoding = 7,
    NSShiftJISStringEncoding = 8,
    NSISOLatin2StringEncoding = 9,
    NSUnicodeStringEncoding = 10,
    NSWindowsCP1251StringEncoding = 11,
    NSWindowsCP1252StringEncoding = 12,
    NSWindowsCP1253StringEncoding = 13,
    NSWindowsCP1254StringEncoding = 14,
    NSWindowsCP1250StringEncoding = 15,
    NSISO2022JPStringEncoding = 21,
    NSMacOSRomanStringEncoding = 30,
    NSUTF16BigEndianStringEncoding = 0x90000100,
    NSUTF16LittleEndianStringEncoding = 0x94000100,
    NSUTF32StringEncoding = 0x8c000100,
    NSUTF32BigEndianStringEncoding = 0x98000100,
    NSUTF32LittleEndianStringEncoding = 0x9c000100,
    NSProprietaryStringEncoding = 65536
};
```

Constants

`NSASCIIStringEncoding`
Strict 7-bit ASCII encoding within 8-bit chars; ASCII values 0…127 only.
Available in Mac OS X v10.0 and later.
Declared in `NSString.h`.

`NSISO2022JPStringEncoding`
ISO 2022 Japanese encoding for email.
Available in Mac OS X v10.0 and later.
Declared in `NSString.h`.

`NSISOLatin1StringEncoding`
8-bit ISO Latin 1 encoding.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSISOLatin2StringEncoding`
8-bit ISO Latin 2 encoding.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSJapaneseEUCStringEncoding`
8-bit EUC encoding for Japanese text.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSMacOSRomanStringEncoding`
Classic Macintosh Roman encoding.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSNEXTSTEPStringEncoding`
8-bit ASCII encoding with NEXTSTEP extensions.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSNonLossyASCIIStringEncoding`
7-bit verbose ASCII to represent all Unicode characters.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSShiftJISStringEncoding`
8-bit Shift-JIS encoding for Japanese text.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSSymbolStringEncoding`
8-bit Adobe Symbol encoding vector.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSUTF8StringEncoding`
An 8-bit representation of Unicode characters, suitable for transmission or storage by ASCII-based systems.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSUnicodeStringEncoding`
The canonical Unicode encoding for string objects.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSWindowsCP1250StringEncoding`
Microsoft Windows codepage 1250; equivalent to WinLatin2.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSWindowsCP1251StringEncoding`
Microsoft Windows codepage 1251, encoding Cyrillic characters; equivalent to AdobeStandardCyrillic font encoding.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSWindowsCP1252StringEncoding`
Microsoft Windows codepage 1252; equivalent to WinLatin1.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSWindowsCP1253StringEncoding`
Microsoft Windows codepage 1253, encoding Greek characters.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSWindowsCP1254StringEncoding`
Microsoft Windows codepage 1254, encoding Turkish characters.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSUTF16BigEndianStringEncoding`
`NSUTF16StringEncoding` encoding with explicit endianness specified.

Available in Mac OS X v10.5 and later.

Declared in `NSString.h`.

`NSUTF16LittleEndianStringEncoding`
`NSUTF16StringEncoding` encoding with explicit endianness specified.

Available in Mac OS X v10.5 and later.

Declared in `NSString.h`.

`NSUTF32StringEncoding`
32-bit UTF encoding.

Available in Mac OS X v10.5 and later.

Declared in `NSString.h`.

`NSUTF32BigEndianStringEncoding`
`NSUTF32StringEncoding` encoding with explicit endianness specified.

Available in Mac OS X v10.5 and later.

Declared in `NSString.h`.

`NSUTF32LittleEndianStringEncoding`
`NSUTF32StringEncoding` encoding with explicit endianness specified.

Available in Mac OS X v10.5 and later.

Declared in `NSString.h`.

`NSProprietaryStringEncoding`

Installation-specific encoding.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

Discussion

These values represent the various character encodings supported by the `NSString` classes. This is an incomplete list. Additional encodings are defined in *Strings Programming Guide for Core Foundation* (see `CFStringEncodingExt.h`); these encodings can be used with `NSString` by first passing the Core Foundation encoding to the `CFStringConvertEncodingToNSStringEncoding` function.

Declared In

`NSString.h`

NSTask Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSTask.h
Companion guide	Interacting with the Operating System
Related sample code	Moriarity MP3 Player

Overview

Using the `NSTask` class, your program can run another program as a subprocess and can monitor that program's execution. An `NSTask` object creates a separate executable entity; it differs from `NSThread` in that it does not share memory space with the process that creates it.

A task operates within an environment defined by the current values for several items: the current directory, standard input, standard output, standard error, and the values of any environment variables. By default, an `NSTask` object inherits its environment from the process that launches it. If there are any values that should be different for the task, for example, if the current directory should change, you must change the value before you launch the task. A task's environment cannot be changed while it is running.

An `NSTask` object can only be run once. Subsequent attempts to run the task raise an error.

Tasks

Creating and Initializing an NSTask Object

`+ launchedTaskWithLaunchPath:arguments:` (page 1625)
Creates and launches a task with a specified executable and arguments.

`- init` (page 1627)
Returns an initialized `NSTask` object with the environment of the current process.

Returning Task Information

- `arguments` (page 1626)
  Returns the arguments used when the receiver was launched.
- `currentDirectoryPath` (page 1626)
  Returns the task's current directory.
- `environment` (page 1626)
  Returns a dictionary of variables for the environment from which the receiver was launched.
- `launchPath` (page 1628)
  Returns the path of the receiver's executable.
- `processIdentifier` (page 1629)
  Returns the receiver's process identifier.
- `standardError` (page 1633)
  Returns the standard error file used by the receiver.
- `standardInput` (page 1633)
  Returns the standard input file used by the receiver.
- `standardOutput` (page 1633)
  Returns the standard output file used by the receiver.

Running and Stopping a Task

- `interrupt` (page 1627)
  Sends an interrupt signal to the receiver and all of its subtasks.
- `launch` (page 1628)
  Launches the task represented by the receiver.
- `resume` (page 1629)
  Resumes execution of the receiver task that had previously been suspended with a `suspend` (page 1634) message.
- `suspend` (page 1634)
  Suspends execution of the receiver task.
- `terminate` (page 1634)
  Sends a terminate signal to the receiver and all of its subtasks.
- `waitUntilExit` (page 1635)
  Block until the receiver is finished.

Querying the Task State

- `isRunning` (page 1628)
  Returns whether the receiver is still running.
- `terminationStatus` (page 1635)
  Returns the exit status returned by the receiver's executable.

Configuring an NSTask Object

- `setArguments:` (page 1629)
  Sets the command arguments that should be used to launch the executable.
- `setCurrentDirectoryPath:` (page 1630)
  Sets the current directory for the receiver.
- `setEnvironment:` (page 1630)
  Sets the environment for the receiver.
- `setLaunchPath:` (page 1631)
  Sets the receiver's executable.
- `setStandardError:` (page 1631)
  Sets the standard error for the receiver.
- `setStandardInput:` (page 1632)
  Sets the standard input for the receiver.
- `setStandardOutput:` (page 1632)
  Sets the standard output for the receiver.

Class Methods

launchedTaskWithLaunchPath:arguments:

Creates and launches a task with a specified executable and arguments.

```
+ (NSTask *)launchedTaskWithLaunchPath:(NSString *)path arguments:(NSArray
    *)arguments
```

Parameters

path
  The path to the executable.

arguments
  An array of `NSString` objects that supplies the arguments to the task. If *arguments* is `nil`, an `NSInvalidArgumentException` is raised.

Discussion
The task inherits its environment from the process that invokes this method.

The `NSTask` object converts both *path* and the strings in *arguments* to appropriate C-style strings (using `fileSystemRepresentation` (page 1553)) before passing them to the task via `argv[]`. The strings in *arguments* do not undergo shell expansion, so you do not need to do special quoting, and shell variables, such as `$PWD`, are not resolved.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `init` (page 1627)

Declared In
NSTask.h

Instance Methods

arguments

Returns the arguments used when the receiver was launched.

```
- (NSArray *)arguments
```

Return Value
An array of `NSString` objects containing the arguments used when the receiver was launched.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setArguments:` (page 1629)

Declared In
NSTask.h

currentDirectoryPath

Returns the task's current directory.

```
- (NSString *)currentDirectoryPath
```

Return Value
The task's current working directory.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setCurrentDirectoryPath:` (page 1630)

Declared In
NSTask.h

environment

Returns a dictionary of variables for the environment from which the receiver was launched.

```
- (NSDictionary *)environment
```

Return Value
A dictionary of variables for the environment from which the receiver was launched. The dictionary keys are the environment variable names.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `setEnvironment:` (page 1630)
- `environment` (page 1288) (NSProcessInfo)

Declared In
`NSTask.h`

init

Returns an initialized `NSTask` object with the environment of the current process.

```
- (id)init
```

Return Value
An initialized `NSTask` object with the environment of the current process.

Discussion
If you need to modify the environment of a task, use alloc and init, and then set up the environment before launching the new task. Otherwise, just use the class method `launchedTaskWithLaunchPath:arguments:` (page 1625) to create and run the task.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSTask.h`

interrupt

Sends an interrupt signal to the receiver and all of its subtasks.

```
- (void)interrupt
```

Discussion
If the task terminates as a result, which is the default behavior, an `NSTaskDidTerminateNotification` (page 1636) gets sent to the default notification center. This method has no effect if the receiver was already launched and has already finished executing. If the receiver has not been launched yet, this method raises an `NSInvalidArgumentException`.

It is not always possible to interrupt the receiver because it might be ignoring the interrupt signal. `interrupt` sends `SIGINT`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSTask.h`

isRunning

Returns whether the receiver is still running.

```
- (BOOL)isRunning
```

Return Value
`YES` if the receiver is still running, otherwise `NO`. `NO` means either the receiver could not run or it has terminated.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `launch` (page 1628)
- `terminate` (page 1634)
- `waitUntilExit` (page 1635)

Declared In
`NSTask.h`

launch

Launches the task represented by the receiver.

```
- (void)launch
```

Discussion
Raises an `NSInvalidArgumentException` if the launch path has not been set or is invalid or if it fails to create a process.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `launchPath` (page 1628)
- `setLaunchPath:` (page 1631)
- `terminate` (page 1634)
- `waitUntilExit` (page 1635)

Declared In
`NSTask.h`

launchPath

Returns the path of the receiver's executable.

```
- (NSString *)launchPath
```

Return Value
The path of the receiver's executable.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ launchedTaskWithLaunchPath:arguments:` (page 1625)
`- setLaunchPath:` (page 1631)

Declared In
`NSTask.h`

processIdentifier

Returns the receiver's process identifier.

```
- (int)processIdentifier
```

Return Value
The receiver's process identifier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSTask.h`

resume

Resumes execution of the receiver task that had previously been suspended with a `suspend` (page 1634) message.

```
- (BOOL)resume
```

Return Value
`YES` if the receiver was able to resume execution, `NO` otherwise.

Discussion
If multiple `suspend` messages were sent to the receiver, an equal number of `resume` messages must be sent before the task resumes execution.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSTask.h`

setArguments:

Sets the command arguments that should be used to launch the executable.

```
- (void)setArguments:(NSArray *)arguments
```

Parameters

arguments

An array of `NSString` objects that supplies the arguments to the task. If *arguments* is `nil`, an `NSInvalidArgumentException` is raised.

Discussion
The `NSTask` object converts both *`path`* and the strings in *`arguments`* to appropriate C-style strings (using `fileSystemRepresentation` (page 1553)) before passing them to the task via `argv[]`. The strings in *`arguments`* do not undergo shell expansion, so you do not need to do special quoting, and shell variables, such as `$PWD`, are not resolved.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `arguments` (page 1626)

Declared In
`NSTask.h`

setCurrentDirectoryPath:

Sets the current directory for the receiver.

```
- (void)setCurrentDirectoryPath:(NSString *)path
```

Parameters

`path`

The current directory for the task.

Discussion
If this method isn't used, the current directory is inherited from the process that created the receiver. This method raises an `NSInvalidArgumentException` if the receiver has already been launched.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `currentDirectoryPath` (page 1626)

Declared In
`NSTask.h`

setEnvironment:

Sets the environment for the receiver.

```
- (void)setEnvironment:(NSDictionary *)environmentDictionary
```

Parameters

`environmentDictionary`

A dictionary of environment variable values whose keys are the variable names.

Discussion
If this method isn't used, the environment is inherited from the process that created the receiver. This method raises an `NSInvalidArgumentException` if the receiver has already been launched.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `environment` (page 1626)

Declared In
`NSTask.h`

setLaunchPath:

Sets the receiver's executable.

```
- (void)setLaunchPath:(NSString *)path
```

Parameters

path

The path to the executable.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `launchPath` (page 1628)

Declared In
`NSTask.h`

setStandardError:

Sets the standard error for the receiver.

```
- (void)setStandardError:(id)file
```

Parameters

file

The standard error for the receiver, which can be either an `NSFileHandle` or an `NSPipe` object.

Discussion
If *file* is an `NSPipe` object, launching the receiver automatically closes the write end of the pipe in the current task. Don't create a handle for the pipe and pass that as the argument, or the write end of the pipe won't be closed automatically.

If this method isn't used, the standard error is inherited from the process that created the receiver. This method raises an `NSInvalidArgumentException` if the receiver has already been launched.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `standardError` (page 1633)

Declared In
`NSTask.h`

setStandardInput:

Sets the standard input for the receiver.

```
- (void)setStandardInput:(id)file
```

Parameters

file

The standard input for the receiver, which can be either an `NSFileHandle` or an `NSPipe` object.

Discussion

If *file* is an `NSPipe` object, launching the receiver automatically closes the read end of the pipe in the current task. Don't create a handle for the pipe and pass that as the argument, or the read end of the pipe won't be closed automatically.

If this method isn't used, the standard input is inherited from the process that created the receiver. This method raises an `NSInvalidArgumentException` if the receiver has already been launched.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- standardInput` (page 1633)

Declared In

`NSTask.h`

setStandardOutput:

Sets the standard output for the receiver.

```
- (void)setStandardOutput:(id)file
```

Parameters

file

The standard output for the receiver, which can be either an `NSFileHandle` or an `NSPipe` object.

Discussion

If *file* is an `NSPipe` object, launching the receiver automatically closes the write end of the pipe in the current task. Don't create a handle for the pipe and pass that as the argument, or the write end of the pipe won't be closed automatically.

If this method isn't used, the standard output is inherited from the process that created the receiver. This method raises an `NSInvalidArgumentException` if the receiver has already been launched.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- standardOutput` (page 1633)

Declared In

`NSTask.h`

standardError

Returns the standard error file used by the receiver.

```
- (id)standardError
```

Return Value
The standard error file used by the receiver.

Discussion
Standard error is where all diagnostic messages are sent. The object returned is either an `NSFileHandle` or an `NSPipe` instance, depending on what type of object was passed to `setStandardError:` (page 1631).

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setStandardError:` (page 1631)

Declared In
`NSTask.h`

standardInput

Returns the standard input file used by the receiver.

```
- (id)standardInput
```

Return Value
The standard input file used by the receiver.

Discussion
Standard input is where the receiver takes its input from unless otherwise specified. The object returned is either an `NSFileHandle` or an `NSPipe` instance, depending on what type of object was passed to the `setStandardInput:` (page 1632) method.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setStandardInput:` (page 1632)

Declared In
`NSTask.h`

standardOutput

Returns the standard output file used by the receiver.

```
- (id)standardOutput
```

Return Value
The standard output file used by the receiver.

Discussion
Standard output is where the receiver displays its output. The object returned is either an `NSFileHandle` or an `NSPipe` instance, depending on what type of object was passed to the `setStandardOutput:` (page 1632) method.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setStandardOutput:` (page 1632)

Declared In
`NSTask.h`

suspend

Suspends execution of the receiver task.

```
- (BOOL)suspend
```

Return Value
`YES` if the receiver was successfully suspended, `NO` otherwise.

Discussion
Multiple `suspend` messages can be sent, but they must be balanced with an equal number of `resume` (page 1629) messages before the task resumes execution.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSTask.h`

terminate

Sends a terminate signal to the receiver and all of its subtasks.

```
- (void)terminate
```

Discussion
If the task terminates as a result, which is the default behavior, an `NSTaskDidTerminateNotification` (page 1636) gets sent to the default notification center. This method has no effect if the receiver was already launched and has already finished executing. If the receiver has not been launched yet, this method raises an `NSInvalidArgumentException`.

It is not always possible to terminate the receiver because it might be ignoring the terminate signal. `terminate` sends `SIGTERM`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ launchedTaskWithLaunchPath:arguments:` (page 1625)

- launch (page 1628)
- terminationStatus (page 1635)
- waitUntilExit (page 1635)

Declared In
NSTask.h

terminationStatus

Returns the exit status returned by the receiver's executable.

```
- (int)terminationStatus
```

Return Value
The exit status returned by the receiver's executable.

Discussion
Each task defines and documents how its return value should be interpreted. For example, many commands return 0 if they complete successfully or an error code if they don't. You'll need to look at the documentation for that task to learn what values it returns under what circumstances.

This method raises an NSInvalidArgumentException if the receiver is still running. Verify that the receiver is not running before you use it.

```
if (![aTask isRunning]) {
    int status = [aTask terminationStatus];
    if (status == ATASK_SUCCESS_VALUE)
        NSLog(@"Task succeeded.");
    else
        NSLog(@"Task failed.");
}
```

Availability
Available in Mac OS X v10.0 and later.

See Also
- terminate (page 1634)
- waitUntilExit (page 1635)

Declared In
NSTask.h

waitUntilExit

Block until the receiver is finished.

```
- (void)waitUntilExit
```

Discussion
This method first checks to see if the receiver is still running using isRunning (page 1628). Then it polls the current run loop using NSDefaultRunLoopMode until the task completes.

```
[aTask launch];
```

```
[aTask waitUntilExit];
int status = [aTask terminationStatus];

if (status == ATASK_SUCCESS_VALUE)
    NSLog(@"Task succeeded.");
else
    NSLog(@"Task failed.");
```

Availability
Available in Mac OS X v10.0 and later.

See Also
- `launch` (page 1628)
- `terminate` (page 1634)

Declared In
`NSTask.h`

Notifications

NSTaskDidTerminateNotification

Posted when the task has stopped execution. This notification can be posted either when the task has exited normally or as a result of `terminate` (page 1634) being sent to the `NSTask` object. If the `NSTask` object gets released, however, this notification will not get sent, as the port the message would have been sent on was released as part of the task release. The observer method can use `terminationStatus` (page 1635) to determine why the task died. See "Ending an NSTask" for an example.

The notification object is the `NSTask` object that was terminated. This notification does not contain a *`userInfo`* dictionary.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSTask.h`

NSThread Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSThread.h
Companion guide	Threading Programming Guide
Related sample code	ExtractMovieAudioToAIFF QTAudioExtractionPanel SimpleThreads TrivialThreads Vertex Optimization

Overview

An `NSThread` object controls a thread of execution. Use this class when you want to have an Objective-C method run in its own thread of execution. Threads are especially useful when you need to perform a lengthy task, but don't want it to block the execution of the rest of the application. In particular, you can use threads to avoid blocking the main thread of the application, which handles user interface and event-related actions. Threads can also be used to divide a large job into several smaller jobs, which can lead to performance increases on multi-core computers.

Prior to Mac OS X v10.5, the only way to start a new thread is to use the `detachNewThreadSelector:toTarget:withObject:` (page 1640) method. In Mac OS X v10.5 and later, you can create instances of `NSThread` and start them at a later time using the `start` (page 1650) method.

In Mac OS Xv10.5, the `NSThread` class supports semantics similar to those of `NSOperation` for monitoring the runtime condition of a thread. You can use these semantics to cancel the execution of a thread or determine if the thread is still executing or has finished its task. Canceling a thread requires support from your thread code; see the description for `cancel` (page 1645) for more information.

Subclassing Notes

In Mac OS X v10.5 and later, you can subclass `NSThread` and override the `main` method to implement your thread's main entry point. If you override `main`, you do not need to invoke the inherited behavior by calling `super`.

Tasks

Initializing an NSThread Object

- `init` (page 1645)
  Returns an initialized `NSThread` object.
- `initWithTarget:selector:object:` (page 1646)
  Returns an `NSThread` object initialized with the given arguments.

Starting a Thread

+ `detachNewThreadSelector:toTarget:withObject:` (page 1640)
  Detaches a new thread and uses the specified selector as the thread entry point.
- `start` (page 1650)
  Starts the receiver.
- `main` (page 1648)
  The main entry point routine for the thread.

Stopping a Thread

+ `sleepUntilDate:` (page 1644)
  Blocks the current thread until the time specified.
+ `sleepForTimeInterval:` (page 1643)
  Sleeps the thread for a given time interval.
+ `exit` (page 1641)
  Terminates the current thread.
- `cancel` (page 1645)
  Changes the cancelled state of the receiver to indicate that it should exit.

Determining the Thread's Execution State

- `isExecuting` (page 1647)
  Returns a Boolean value that indicates whether the receiver is executing.
- `isFinished` (page 1647)
  Returns a Boolean value that indicates whether the receiver has finished execution.
- `isCancelled` (page 1647)
  Returns a Boolean value that indicates whether the receiver is cancelled.

Working with the Main Thread

`+ isMainThread` (page 1642)
Returns a Boolean value that indicates whether the current thread is the main thread.

`- isMainThread` (page 1648)
Returns a Boolean value that indicates whether the receiver is the main thread.

`+ mainThread` (page 1642)
Returns the `NSThread` object representing the main thread.

Querying the Environment

`+ isMultiThreaded` (page 1642)
Returns whether the application is multithreaded.

`+ currentThread` (page 1640)
Returns the thread object representing the current thread of execution.

`+ callStackReturnAddresses` (page 1640)
Returns an array containing the call stack return addresses.

Working with Thread Properties

`- threadDictionary` (page 1650)
Returns the thread object's dictionary.

`- name` (page 1648)
Returns the name of the receiver.

`- setName:` (page 1649)
Sets the name of the receiver.

`- stackSize` (page 1650)
Returns the stack size of the receiver.

`- setStackSize:` (page 1649)
Sets the stack size of the receiver.

Working with Thread Priorities

`+ threadPriority` (page 1644)
Returns the current thread's priority.

`+ setThreadPriority:` (page 1643)
Sets the current thread's priority.

Class Methods

callStackReturnAddresses

Returns an array containing the call stack return addresses.

```
+ (NSArray *)callStackReturnAddresses
```

Return Value
An array containing the call stack return addresses. This value is `nil` by default.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSThread.h`

currentThread

Returns the thread object representing the current thread of execution.

```
+ (NSThread *)currentThread
```

Return Value
A thread object representing the current thread of execution.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ detachNewThreadSelector:toTarget:withObject:` (page 1640)

Declared In
`NSThread.h`

detachNewThreadSelector:toTarget:withObject:

Detaches a new thread and uses the specified selector as the thread entry point.

```
+ (void)detachNewThreadSelector:(SEL)aSelector toTarget:(id)aTarget
    withObject:(id)anArgument
```

Parameters

aSelector
The selector for the message to send to the target. This selector must take only one argument and must not have a return value.

aTarget
The object that will receive the message *aSelector* on the new thread.

anArgument
The single argument passed to the target. May be `nil`.

Discussion
For non garbage-collected applications, the method *aSelector* is responsible for setting up an autorelease pool for the newly detached thread and freeing that pool before it exits. Garbage-collected applications do not need to create an autorelease pool.

The objects *aTarget* and *anArgument* are retained during the execution of the detached thread, then released. The detached thread is exited (using the `exit` (page 1641) class method) as soon as *aTarget* has completed executing the *aSelector* method.

If this thread is the first thread detached in the application, this method posts the `NSWillBecomeMultiThreadedNotification` (page 1651) with object `nil` to the default notification center.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ currentThread` (page 1640)
`+ isMultiThreaded` (page 1642)
`- start` (page 1650)

Related Sample Code
ExtractMovieAudioToAIFF
MassiveImage
OpenGLCaptureToMovie
QTAudioExtractionPanel
SharedMemory

Declared In
`NSThread.h`

exit

Terminates the current thread.

```
+ (void)exit
```

Discussion
This method uses the `currentThread` (page 1640) class method to access the current thread. Before exiting the thread, this method posts the `NSThreadWillExitNotification` (page 1651) with the thread being exited to the default notification center. Because notifications are delivered synchronously, all observers of `NSThreadWillExitNotification` (page 1651) are guaranteed to receive the notification before the thread exits.

Invoking this method should be avoided as it does not give your thread a chance to clean up any resources it allocated during its execution.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ currentThread` (page 1640)
`+ sleepUntilDate:` (page 1644)

Related Sample Code
SimpleThreads
Vertex Optimization

Declared In
NSThread.h

isMainThread

Returns a Boolean value that indicates whether the current thread is the main thread.

```
+ (BOOL)isMainThread
```

Return Value
YES if the current thread is the main thread, otherwise NO.

Availability
Available in Mac OS X v10.5 and later.

See Also
+ mainThread (page 1642)

Declared In
NSThread.h

isMultiThreaded

Returns whether the application is multithreaded.

```
+ (BOOL)isMultiThreaded
```

Return Value
YES if the application is multithreaded, NO otherwise.

Discussion
An application is considered multithreaded if a thread was ever detached from the main thread using either detachNewThreadSelector:toTarget:withObject: (page 1640) or start (page 1650). If you detached a thread in your application using a non-Cocoa API, such as the POSIX or Multiprocessing Services APIs, this method could still return NO. The detached thread does not have to be currently running for the application to be considered multithreaded—this method only indicates whether a single thread has been spawned.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSThread.h

mainThread

Returns the NSThread object representing the main thread.

```
+ (NSThread *)mainThread
```

Return Value
The `NSThread` object representing the main thread.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- isMainThread` (page 1648)

Declared In
`NSThread.h`

setThreadPriority:

Sets the current thread's priority.

```
+ (BOOL)setThreadPriority:(double)priority
```

Parameters
priority
The new priority, specified with a floating point number from 0.0 to 1.0, where 1.0 is highest priority.

Return Value
`YES` if the priority assignment succeeded, `NO` otherwise.

Discussion
The priorities in this range are mapped to the operating system's priority values.

Availability
Available in Mac OS X v10.2 and later.

See Also
`+ threadPriority` (page 1644)

Related Sample Code
ExtractMovieAudioToAIFF
QTAudioExtractionPanel
QTExtractAndConvertToAIFF
QTExtractAndConvertToMovieFile
Vertex Optimization

Declared In
`NSThread.h`

sleepForTimeInterval:

Sleeps the thread for a given time interval.

```
+ (void)sleepForTimeInterval:(NSTimeInterval)ti
```

Parameters

ti

The duration of the sleep.

Discussion
No run loop processing occurs while the thread is blocked.

Availability
Available in Mac OS X v10.5 and later.

Declared In
NSThread.h

sleepUntilDate:

Blocks the current thread until the time specified.

```
+ (void)sleepUntilDate:(NSDate *)aDate
```

Parameters

aDate

The time at which to resume processing.

Discussion
No run loop processing occurs while the thread is blocked.

Availability
Available in Mac OS X v10.0 and later.

See Also
+ currentThread (page 1640)
+ exit (page 1641)

Related Sample Code
Core Data HTML Store
SharedMemory
SimpleThreads
TrivialThreads

Declared In
NSThread.h

threadPriority

Returns the current thread's priority.

```
+ (double)threadPriority
```

Return Value
The current thread's priority, which is specified by a floating point number from 0.0 to 1.0, where 1.0 is highest priority.

Discussion
The priorities in this range are mapped to the operating system's priority values. A “typical” thread priority might be 0.5, but because the priority is determined by the kernel, there is no guarantee what this value actually will be.

Availability
Available in Mac OS X v10.2 and later.

See Also
`+ setThreadPriority:` (page 1643)

Related Sample Code
ExtractMovieAudioToAIFF
QTAudioExtractionPanel
QTExtractAndConvertToAIFF
QTExtractAndConvertToMovieFile

Declared In
`NSThread.h`

Instance Methods

cancel

Changes the cancelled state of the receiver to indicate that it should exit.

```
- (void)cancel
```

Discussion
The semantics of this method are the same as those used for the `NSOperation` object. This method sets state information in the receiver that is then reflected by the `isCancelled` method. Threads that support cancellation should periodically call the `isCancelled` method to determine if the thread has in fact been cancelled, and exit if it has been.

For more information about cancellation and operation objects, see *NSOperation Class Reference*.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- isCancelled` (page 1647)

Declared In
`NSThread.h`

init

Returns an initialized `NSThread` object.

```
- (id)init
```

Return Value
An initialized `NSThread` object.

Discussion
This is the designated initializer for `NSThread`.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `initWithTarget:selector:object:` (page 1646)
- `start` (page 1650)

Declared In
`NSThread.h`

initWithTarget:selector:object:

Returns an `NSThread` object initialized with the given arguments.

```
- (id)initWithTarget:(id)target
    selector:(SEL)selector
    object:(id)argument
```

Parameters

target
The object to which the message specified by *selector* is sent.

selector
The selector for the message to send to *target*. This selector must take only one argument and must not have a return value.

argument
The single argument passed to the target. May be `nil`.

Return Value
An `NSThread` object initialized with the given arguments.

Discussion
For non garbage-collected applications, the method *selector* is responsible for setting up an autorelease pool for the newly detached thread and freeing that pool before it exits. Garbage-collected applications do not need to create an autorelease pool.

The objects *target* and *argument* are retained during the execution of the detached thread. They are released when the thread finally exits.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `init` (page 1645)
- `start` (page 1650)

Declared In
`NSThread.h`

isCancelled

Returns a Boolean value that indicates whether the receiver is cancelled.

```
- (BOOL)isCancelled
```

Return Value
`YES` if the receiver has been cancelled, otherwise `NO`.

Discussion
If your thread supports cancellation, it should call this method periodically and exit if it ever returns `YES`.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `cancel` (page 1645)
- `isExecuting` (page 1647)
- `isFinished` (page 1647)

Declared In
`NSThread.h`

isExecuting

Returns a Boolean value that indicates whether the receiver is executing.

```
- (BOOL)isExecuting
```

Return Value
`YES` if the receiver is executing, otherwise `NO`.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `isCancelled` (page 1647)
- `isFinished` (page 1647)

Declared In
`NSThread.h`

isFinished

Returns a Boolean value that indicates whether the receiver has finished execution.

```
- (BOOL)isFinished
```

Return Value
`YES` if the receiver has finished execution, otherwise `NO`.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `isCancelled` (page 1647)
- `isExecuting` (page 1647)

Declared In
`NSThread.h`

isMainThread

Returns a Boolean value that indicates whether the receiver is the main thread.

```
- (BOOL)isMainThread
```

Return Value
`YES` if the receiver is the main thread, otherwise `NO`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSThread.h`

main

The main entry point routine for the thread.

```
- (void)main
```

Discussion
The default implementation of this method takes the target and selector used to initialize the receiver and invokes the selector on the specified target. If you subclass `NSThread`, you can override this method and use it to implement the main body of your thread instead. If you do so, you do not need to invoke `super`.

You should never invoke this method directly. You should always start your thread by invoking the `start` method.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `start` (page 1650)

Declared In
`NSThread.h`

name

Returns the name of the receiver.

```
- (NSString *)name
```

Return Value
The name of the receiver.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `setName:` (page 1649)

Declared In
`NSThread.h`

setName:

Sets the name of the receiver.

```
- (void)setName:(NSString *)n
```

Parameters

n

The name for the receiver.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `name` (page 1648)

Declared In
`NSThread.h`

setStackSize:

Sets the stack size of the receiver.

```
- (void)setStackSize:(NSUInteger)s
```

Parameters

s

The stack size for the receiver. This value must be a multiple of 4KB.

Discussion
You must call this method before starting your thread. Setting the stack size after the thread has started changes the attribute size (which is reflected by the `stackSize` (page 1650) method), but it does not affect the actual number of pages set aside for the thread.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `stackSize` (page 1650)

Declared In
`NSThread.h`

stackSize

Returns the stack size of the receiver.

```
- (NSUInteger)stackSize
```

Return Value
The stack size of the receiver.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `setStackSize:` (page 1649)

Declared In
`NSThread.h`

start

Starts the receiver.

```
- (void)start
```

Discussion
This method spawns the new thread and invokes the receiver's `main` method on the new thread. If you initialized the receiver with a target and selector, the default `main` method invokes that selector automatically.

If this thread is the first thread detached in the application, this method posts the `NSWillBecomeMultiThreadedNotification` (page 1651) with object `nil` to the default notification center.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `init` (page 1645)
- `initWithTarget:selector:object:` (page 1646)
- `main` (page 1648)

Declared In
`NSThread.h`

threadDictionary

Returns the thread object's dictionary.

```
- (NSMutableDictionary *)threadDictionary
```

Return Value
The thread object's dictionary.

Discussion
You can use the returned dictionary to store thread-specific data. The thread dictionary is not used during any manipulations of the `NSThread` object—it is simply a place where you can store any interesting data. For example, Foundation uses it to store the thread's default `NSConnection` and `NSAssertionHandler` instances. You may define your own keys for the dictionary.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSThread.h`

Notifications

NSDidBecomeSingleThreadedNotification

Not implemented.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSThread.h`

NSThreadWillExitNotification

An `NSThread` object posts this notification when it receives the `exit` (page 1641) message, before the thread exits. Observer methods invoked to receive this notification execute in the exiting thread, before it exits.

The notification object is the exiting `NSThread` object. This notification does not contain a *`userInfo`* dictionary.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSThread.h`

NSWillBecomeMultiThreadedNotification

Posted when the first thread is detached from the current thread. The `NSThread` class posts this notification at most once—the first time a thread is detached using `detachNewThreadSelector:toTarget:withObject:` (page 1640) or the `start` (page 1650) method. Subsequent invocations of those methods do not post this notification. Observers of this notification have their notification method invoked in the main thread, not the new thread. The observer notification methods always execute before the new thread begins executing.

This notification does not contain a notification object or a *`userInfo`* dictionary.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSThread.h`

NSTimer Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSTimer.h
Companion guides	Timer Programming Topics for Cocoa Run Loops
Related sample code	FunkyOverlayWindow ImageClient NSOperationSample ThreadsImporter ThreadsImportMovie

Overview

`NSTimer` creates timer objects or, more simply, timers. A timer waits until a certain time interval has elapsed and then fires, sending a specified message to a specified object. For example, you could create an `NSTimer` object that sends a message to a window, telling it to update itself after a certain time interval.

Timers work in conjunction with run loops. To use a timer effectively, you should be aware of how run loops operate—see `NSRunLoop` and *Run Loops*. Note in particular that run loops retain their timers, so you can release a timer after you have added it to a run loop.

A timer is not a real-time mechanism; it fires only when one of the run loop modes to which the timer has been added is running and able to check if the timer's firing time has passed. Because of the various input sources a typical run loop manages, the effective resolution of the time interval for a timer is limited to on the order of 50-100 milliseconds. If a timer's firing time occurs while the run loop is in a mode that is not monitoring the timer or during a long callout, the timer does not fire until the next time the run loop checks the timer. Therefore, the actual time at which the timer fires potentially can be a significant period of time after the scheduled firing time.

A repeating timer reschedules itself based on the scheduled firing time, not the actual firing time. For example, if a timer is scheduled to fire at a particular time and every 5 seconds after that, the scheduled firing time will always fall on the original 5 second time intervals, even if the actual firing time gets delayed. If the firing time is delayed so far that it passes one or more of the scheduled firing times, the timer is fired only once for that time period; the timer is then rescheduled, after firing, for the next scheduled firing time in the future.

Each run loop timer can be registered in only one run loop at a time, although it can be added to multiple run loop modes within that run loop.

There are three ways to create a timer. The `scheduledTimerWithTimeInterval:invocation:repeats:` (page 1655) and `scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:` (page 1656) class methods automatically add the new timer to the current `NSRunLoop` object in the default mode (`NSDefaultRunLoopMode`). The `timerWithTimeInterval:invocation:repeats:` (page 1657) and `timerWithTimeInterval:target:selector:userInfo:repeats:` (page 1657) class methods create timers that you may add to a run loop at a later time by sending the message `addTimer:forMode:` (page 1333) to the `NSRunLoop` object. Finally, you can allocate the timer directly and initialize it with `initWithFireDate:interval:target:selector:userInfo:repeats:` (page 1659), which allows you to specify both an initial fire date and a repeating interval. If you specify that the timer should repeat, it automatically reschedules itself after it fires. If you specify that the timer should not repeat, it is automatically invalidated after it fires.

To request the removal of a timer from an `NSRunLoop` object, send the timer the `invalidate` (page 1660) message from the same thread on which the timer was installed. This message immediately disables the timer, so it no longer affects the `NSRunLoop` object. The run loop removes and releases the timer, either just before the `invalidate` (page 1660) method returns or at some later point.

`NSTimer` is "toll-free bridged" with its Core Foundation counterpart, *CFRunLoopTimer Reference*. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSTimer *` parameter, you can pass a `CFRunLoopTimerRef`, and in a function where you see a `CFRunLoopTimerRef` parameter, you can pass an `NSTimer` instance (you cast one type to the other to suppress compiler warnings). See Interchangeable Data Types for more information on toll-free bridging.

Tasks

Creating a Timer

`+ scheduledTimerWithTimeInterval:invocation:repeats:` (page 1655)
Returns a new `NSTimer` object, scheduled with the current `NSRunLoop` object in the default mode.

`+ scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:` (page 1656)
Returns a new `NSTimer` object, scheduled the current `NSRunLoop` object in the default mode.

`+ timerWithTimeInterval:invocation:repeats:` (page 1657)
Returns a new `NSTimer` that, when added to a run loop, will fire after a given number of seconds.

`+ timerWithTimeInterval:target:selector:userInfo:repeats:` (page 1657)
Returns a new `NSTimer` that, when added to a run loop, will fire after a specified number of seconds.

`- initWithFireDate:interval:target:selector:userInfo:repeats:` (page 1659)
Initializes a new `NSTimer` that, when added to a run loop, will fire at a given date.

Firing a Timer

`- fire` (page 1658)
Causes the receiver's message to be sent to its target.

Stopping a Timer

- `invalidate` (page 1660)
  Stops the receiver from ever firing again and requests its removal from its `NSRunLoop` object.

Information About a Timer

- `isValid` (page 1661)
  Returns a Boolean value that indicates whether the receiver is currently valid.
- `fireDate` (page 1659)
  Returns the date at which the receiver will fire.
- `setFireDate:` (page 1661)
  Resets the receiver to fire next at a given date.
- `timeInterval` (page 1661)
  Returns the receiver's time interval.
- `userInfo` (page 1661)
  Returns the receiver's *`userInfo`* object.

Class Methods

scheduledTimerWithTimeInterval:invocation:repeats:

Returns a new `NSTimer` object, scheduled with the current `NSRunLoop` object in the default mode.

```
+ (NSTimer *)scheduledTimerWithTimeInterval:(NSTimeInterval)seconds
    invocation:(NSInvocation *)invocation  repeats:(BOOL)repeats
```

Parameters

`seconds`
  The number of seconds between firings of the timer.
  If *`seconds`* is less than or equal to `0.0`, this method chooses a nonnegative interval.

`invocation`
  The invocation to use when the timer fires.
  The timer instructs the invocation object to retain its arguments.

`repeats`
  If `YES`, the timer will repeatedly reschedule itself until invalidated. If `NO`, the timer will be invalidated after it fires.

Return Value
A new `NSTimer` object, configured according to the specified parameters.

Discussion
After *`seconds`* seconds have elapsed, the timer fires, invoking *`invocation`*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSTimer.h

scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:

Returns a new NSTimer object, scheduled the current NSRunLoop object in the default mode.

```
+ (NSTimer *)scheduledTimerWithTimeInterval:(NSTimeInterval)seconds
    target:(id)target selector:(SEL)aSelector  userInfo:(id)userInfo
    repeats:(BOOL)repeats
```

Parameters

seconds

The number of seconds between firings of the timer.

If *seconds* is less than or equal to 0.0, this method chooses a nonnegative interval.

target

The object to which to send the message specified by *aSelector* when the timer fires.

aSelector

The message to send to *target* when the timer fires.

The selector must have the following signature:

```
- (void)timerFireMethod:(NSTimer*)theTimer
```

The timer passes itself as the argument to this method.

userInfo

The user info the new timer.

This parameter may be nil.

repeats

If YES, the timer will repeatedly reschedule itself until invalidated. If NO, the timer will be invalidated after it fires.

Return Value
A new NSTimer object, configured according to the specified parameters.

Discussion
After *seconds* seconds have elapsed, the timer fires, sending the message *aSelector* to *target*.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Draw Pixels
FunkyOverlayWindow
ImageClient
TextureRange
UIElementInspector

Declared In
NSTimer.h

timerWithTimeInterval:invocation:repeats:

Returns a new `NSTimer` that, when added to a run loop, will fire after a given number of seconds.

```
+ (NSTimer *)timerWithTimeInterval:(NSTimeInterval)seconds  invocation:(NSInvocation
     *)invocation  repeats:(BOOL)repeats
```

Parameters

seconds

The number of seconds between firings of the timer.

If *seconds* is less than or equal to `0.0`, this method chooses a nonnegative interval.

invocation

The invocation to use when the timer fires.

The timer instructs the invocation object to retain its arguments.

repeats

If `YES`, the timer will repeatedly reschedule itself until invalidated. If `NO`, the timer will be invalidated after it fires.

Return Value

A new `NSTimer` object, configured according to the specified parameters.

Discussion

You must add the new timer to a run loop, using `addTimer:forMode:` (page 1333). Then, after *seconds* have elapsed, the timer fires, invoking *invocation*. (If the timer is configured to repeat, there is no need to subsequently re-add the timer to the run loop.)

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSTimer.h`

timerWithTimeInterval:target:selector:userInfo:repeats:

Returns a new `NSTimer` that, when added to a run loop, will fire after a specified number of seconds.

```
+ (NSTimer *)timerWithTimeInterval:(NSTimeInterval)seconds  target:(id)target
    selector:(SEL)aSelector  userInfo:(id)userInfo  repeats:(BOOL)repeats
```

Parameters

seconds

The number of seconds between firings of the timer.

If *seconds* is less than or equal to `0.0`, this method chooses a nonnegative interval.

target

The object to which to send the message specified by *aSelector* when the timer fires.

aSelector

The message to send to *target* when the timer fires.

The selector must have the following signature:

```
- (void)timerFireMethod:(NSTimer*)theTimer
```

The timer passes itself as the argument to this method.

userInfo

The user info for the new timer.

This parameter may be `nil`.

repeats

If `YES`, the timer will repeatedly reschedule itself until invalidated. If `NO`, the timer will be invalidated after it fires.

Return Value
A new `NSTimer` object, configured according to the specified parameters.

Discussion
You must add the new timer to a run loop, using `addTimer:forMode:` (page 1333). Then, after *seconds* seconds have elapsed, the timer fires, sending the message *aSelector* to *target*. (If the timer is configured to repeat, there is no need to subsequently re-add the timer to the run loop.)

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
GLChildWindowDemo
NSGLImage
Quartz Composer QCTV
Quartz Composer Texture
SillyFrequencyLevels

Declared In
`NSTimer.h`

Instance Methods

fire

Causes the receiver's message to be sent to its target.

```
- (void)fire
```

Discussion
You can use this method to fire a repeating timer without interrupting its regular firing schedule.

If the timer is non-repeating, it is automatically invalidated after firing, even if its scheduled fire date has not arrived.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- invalidate` (page 1660)

Declared In
`NSTimer.h`

fireDate

Returns the date at which the receiver will fire.

```
- (NSDate *)fireDate
```

Return Value
The date at which the receiver will fire. If the timer is no longer valid, this method returns the last date at which the timer fired.

Discussion
Use `isValid` (page 1661) to verify that the timer is valid.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setFireDate:` (page 1661)

Declared In
`NSTimer.h`

initWithFireDate:interval:target:selector:userInfo:repeats:

Initializes a new `NSTimer` that, when added to a run loop, will fire at a given date.

```
- (id)initWithFireDate:(NSDate *)date interval:(NSTimeInterval)seconds
    target:(id)target  selector:(SEL)aSelector  userInfo:(id)userInfo
    repeats:(BOOL)repeats
```

Parameters

date
The time at which the timer should first fire.

seconds
The number of seconds between firings of the timer.

If *seconds* is less than or equal to `0.0`, this method chooses a nonnegative interval.

target
The object to which to send the message specified by *aSelector* when the timer fires.

aSelector
The message to send to *target* when the timer fires.

The selector must have the following signature:

```
- (void)timerFireMethod:(NSTimer*)theTimer
```

The timer passes itself as the argument to this method.

userInfo
The user info for the new timer.

This parameter may be `nil`.

repeats
If `YES`, the timer will repeatedly reschedule itself until invalidated. If `NO`, the timer will be invalidated after it fires.

Return Value
The receiver, initialized such that, when added to a run loop, it will fire at *date* and then, if *repeats* is YES, every *seconds* after that.

Discussion
You must add the new timer to a run loop, using addTimer:forMode: (page 1333). Upon firing, the timer sends the message *aSelector* to *target*. (If the timer is configured to repeat, there is no need to subsequently re-add the timer to the run loop.)

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
WhackedTV

Declared In
NSTimer.h

invalidate

Stops the receiver from ever firing again and requests its removal from its NSRunLoop object.

```
- (void)invalidate
```

Discussion
This is the only way to remove a timer from an NSRunLoop object. The NSRunLoop object removes and releases the timer, either just before the invalidate (page 1660) method returns or at some later point.

If it was configured with a target and user info, the receiver releases its references to the them at the point of invalidation.

Special Considerations

You must send this message from the thread on which the timer was installed. If you send this message from another thread, the input source associated with the timer may not be removed from its run loop, which could prevent the thread from exiting properly.

Availability
Available in Mac OS X v10.0 and later.

See Also
- fire (page 1658)

Related Sample Code
FunkyOverlayWindow
LiveVideoMixer2
Mountains
NSOperationSample
WhackedTV

Declared In
NSTimer.h

isValid

Returns a Boolean value that indicates whether the receiver is currently valid.

```
- (BOOL)isValid
```

Return Value
`YES` if the receiver is currently valid, otherwise `NO`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSTimer.h`

setFireDate:

Resets the receiver to fire next at a given date.

```
- (void)setFireDate:(NSDate *)date
```

Parameters
date
The date at which to fire the receiver.

Availability
Available in Mac OS X v10.2 and later.

See Also
`- fireDate` (page 1659)

Declared In
`NSTimer.h`

timeInterval

Returns the receiver's time interval.

```
- (NSTimeInterval)timeInterval
```

Return Value
The receiver's time interval. If the receiver is a non-repeating timer, returns `0` (even if a time interval was set).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSTimer.h`

userInfo

Returns the receiver's *userInfo* object.

```
- (id)userInfo
```

Return Value
The receiver's *userInfo* object.

Discussion
Do not invoke this method after the timer is invalidated. Use `isValid` (page 1661) to test whether the timer is valid.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:` (page 1656)
`+ timerWithTimeInterval:target:selector:userInfo:repeats:` (page 1657)
`- invalidate` (page 1660)

Declared In
`NSTimer.h`

NSTimeZone Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSTimeZone.h
Companion guide	Date and Time Programming Guide for Cocoa

Overview

`NSTimeZone` is an abstract class that defines the behavior of time zone objects. Time zone objects represent geopolitical regions. Consequently, these objects have names for these regions. Time zone objects also represent a temporal offset, either plus or minus, from Greenwich Mean Time (GMT) and an abbreviation (such as PST for Pacific Standard Time).

`NSTimeZone` provides several class methods to get time zone objects: `timeZoneWithName:` (page 1670), `timeZoneWithName:data:` (page 1670), `timeZoneWithAbbreviation:` (page 1669), and `timeZoneForSecondsFromGMT:` (page 1669). The class also permits you to set the default time zone *within your application* (`setDefaultTimeZone:` (page 1668)). You can access this default time zone at any time with the `defaultTimeZone` (page 1666) class method, and with the `localTimeZone` (page 1667) class method, you can get a relative time zone object that decodes itself to become the default time zone for any locale in which it finds itself.

Cocoa does not provide any API to change the time zone of the computer, or of other applications.

Some `NSCalendarDate` methods return date objects that are automatically bound to time zone objects. These date objects use the functionality of `NSTimeZone` to adjust dates for the proper locale. Unless you specify otherwise, objects returned from `NSCalendarDate` are bound to the default time zone for the current locale.

Note that, strictly, time zone database entries such as “America/Los_Angeles” are IDs not names. An example of a time zone name is “Pacific Daylight Time”. Although many `NSTimeZone` method names include the word “name”, they refer to IDs.

`NSTimeZone` is “toll-free bridged” with its Core Foundation counterpart, *CFTimeZone Reference*. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSTimeZone *` parameter, you can pass a `CFTimeZoneRef`,

and in a function where you see a `CFTimeZoneRef` parameter, you can pass an `NSTimeZone` instance (you cast one type to the other to suppress compiler warnings). See Interchangeable Data Types for more information on toll-free bridging.

Adopted Protocols

NSCoding
- `- encodeWithCoder:` (page 2034)
- `- initWithCoder:` (page 2034)

NSCopying
- `- copyWithZone:` (page 2042)

Tasks

Creating and Initializing Time Zone Objects

`+ timeZoneWithAbbreviation:` (page 1669)
Returns the time zone object identified by a given abbreviation.

`+ timeZoneWithName:` (page 1670)
Returns the time zone object identified by a given ID.

`+ timeZoneWithName:data:` (page 1670)
Returns the time zone with a given ID whose data has been initialized using given data,

`+ timeZoneForSecondsFromGMT:` (page 1669)
Returns a time zone object offset from Greenwich Mean Time by a given number of seconds.

`- initWithName:` (page 1673)
Returns a time zone initialized with a given ID.

`- initWithName:data:` (page 1674)
Initializes a time zone with a given ID and time zone data.

Working with System Time Zones

`+ localTimeZone` (page 1667)
Returns an object that forwards all messages to the default time zone for the current application.

`+ defaultTimeZone` (page 1666)
Returns the default time zone for the current application.

`+ setDefaultTimeZone:` (page 1668)
Sets the default time zone for the current application to a given time zone.

`+ resetSystemTimeZone` (page 1668)
Resets the system time zone object cached by the application, if any.

+ systemTimeZone (page 1668)
Returns the time zone currently used by the system.

Getting Time Zone Information

+ abbreviationDictionary (page 1666)
Returns a dictionary holding the mappings of time zone abbreviations to time zone names.

+ knownTimeZoneNames (page 1667)
Returns an array of strings listing the IDs of all the time zones known to the system.

Getting Information About a Specific Time Zone

- abbreviation (page 1671)
Returns the abbreviation for the receiver.

- abbreviationForDate: (page 1671)
Returns the abbreviation for the receiver at a given date.

- name (page 1676)
Returns the geopolitical region ID that identifies the receiver.

- secondsFromGMT (page 1677)
Returns the current difference in seconds between the receiver and Greenwich Mean Time.

- secondsFromGMTForDate: (page 1677)
Returns the difference in seconds between the receiver and Greenwich Mean Time at a given date.

- data (page 1672)
Returns the data that stores the information used by the receiver.

Comparing Time Zones

- isEqualToTimeZone: (page 1675)
Returns a Boolean value that indicates whether the receiver has the same name and data as another given time zone.

Describing a Time Zone

- description (page 1673)
Returns the description of the receiver.

- localizedName:locale: (page 1675)
Returns the name of the receiver localized for a given locale.

Getting Information About Daylight Saving

- isDaylightSavingTime (page 1674)
Returns a Boolean value that indicates whether the receiver is currently using daylight saving time.

- `daylightSavingTimeOffset` (page 1672)
  Returns the current daylight saving time offset of the receiver.
- `isDaylightSavingTimeForDate:` (page 1675)
  Returns a Boolean value that indicates whether the receiver uses daylight savings time at a given date.
- `daylightSavingTimeOffsetForDate:` (page 1673)
  Returns the daylight saving time offset for a given date.
- `nextDaylightSavingTimeTransition` (page 1676)
  Returns the date of the next daylight saving time transition for the receiver.
- `nextDaylightSavingTimeTransitionAfterDate:` (page 1677)
  Returns the next daylight saving time transition after a given date.

Class Methods

abbreviationDictionary

Returns a dictionary holding the mappings of time zone abbreviations to time zone names.

```
+ (NSDictionary *)abbreviationDictionary
```

Return Value
A dictionary holding the mappings of time zone abbreviations to time zone names.

Discussion
Note that more than one time zone may have the same abbreviation—for example, US/Pacific and Canada/Pacific both use the abbreviation "PST." In these cases, `abbreviationDictionary` chooses a single name to map the abbreviation to.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSTimeZone.h`

defaultTimeZone

Returns the default time zone for the current application.

```
+ (NSTimeZone *)defaultTimeZone
```

Return Value
The default time zone for the current application. If no default time zone has been set, this method invokes `systemTimeZone` (page 1668) and returns the system time zone.

Discussion
The default time zone is the one that the application is running with, which you can change (so you can make the application run as if it were in a different time zone).

If you get the default time zone and hold onto the returned object, it does not change if a subsequent invocation of `setDefaultTimeZone:` (page 1668) changes the default time zone—you still have the specific time zone you originally got. Contrast this behavior with the object returned by `localTimeZone` (page 1667).

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ localTimeZone` (page 1667)
`+ setDefaultTimeZone:` (page 1668)
`+ systemTimeZone` (page 1668)

Declared In
`NSTimeZone.h`

knownTimeZoneNames

Returns an array of strings listing the IDs of all the time zones known to the system.

```
+ (NSArray *)knownTimeZoneNames
```

Return Value
An array of strings listing the IDs of all the time zones known to the system.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSTimeZone.h`

localTimeZone

Returns an object that forwards all messages to the default time zone for the current application.

```
+ (NSTimeZone *)localTimeZone
```

Return Value
An object that forwards all messages to the default time zone for the current application.

Discussion
The local time zone represents the current state of the default time zone at all times. If you get the *default* time zone (using `defaultTimeZone` (page 1666)) and hold onto the returned object, it does not change if a subsequent invocation of `setDefaultTimeZone:` (page 1668) changes the default time zone—you still have the specific time zone you originally got. The *local* time zone adds a level of indirection, it acts as if it were the current default time zone whenever you invoke a method on it.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ defaultTimeZone` (page 1666)
`+ setDefaultTimeZone:` (page 1668)

Declared In
NSTimeZone.h

resetSystemTimeZone

Resets the system time zone object cached by the application, if any.

```
+ (void)resetSystemTimeZone
```

Discussion
If the application has cached the system time zone, this method clears that cached object. If you subsequently invoke systemTimeZone (page 1668), NSTimeZone will attempt to redetermine the system time zone and a new object will be created and cached (see systemTimeZone (page 1668)).

Availability
Available in Mac OS X v10.0 and later.

See Also
+ systemTimeZone (page 1668)

Declared In
NSTimeZone.h

setDefaultTimeZone:

Sets the default time zone for the current application to a given time zone.

```
+ (void)setDefaultTimeZone:(NSTimeZone *)aTimeZone
```

Parameters
aTimeZone
The new default time zone for the current application.

Discussion
There can be only one default time zone, so by setting a new default time zone, you lose the previous one.

Availability
Available in Mac OS X v10.0 and later.

See Also
+ defaultTimeZone (page 1666)
+ localTimeZone (page 1667)

Declared In
NSTimeZone.h

systemTimeZone

Returns the time zone currently used by the system.

```
+ (NSTimeZone *)systemTimeZone
```

Return Value
The time zone currently used by the system. If the current time zone cannot be determined, returns the GMT time zone.

Special Considerations

If you get the system time zone, it is cached by the application and does not change if the user subsequently changes the system time zone. The next time you invoke `systemTimeZone`, you get back the same time zone you originally got. You have to invoke `resetSystemTimeZone` (page 1668) to clear the cached object.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ resetSystemTimeZone` (page 1668)

Declared In
`NSTimeZone.h`

timeZoneForSecondsFromGMT:

Returns a time zone object offset from Greenwich Mean Time by a given number of seconds.

```
+ (id)timeZoneForSecondsFromGMT:(NSInteger)seconds
```

Parameters

seconds
The number of seconds by which the new time zone is offset from GMT.

Return Value
A time zone object offset from Greenwich Mean Time by *seconds*.

Discussion
The name of the new time zone is GMT +/– the offset, in hours and minutes. Time zones created with this method never have daylight savings, and the offset is constant no matter the date.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ timeZoneWithAbbreviation:` (page 1669)
`+ timeZoneWithName:` (page 1670)

Declared In
`NSTimeZone.h`

timeZoneWithAbbreviation:

Returns the time zone object identified by a given abbreviation.

```
+ (id)timeZoneWithAbbreviation:(NSString *)abbreviation
```

Parameters

abbreviation

An abbreviation for a time zone.

Return Value

The time zone object identified by *abbreviation* determined by resolving the abbreviation to a name using the abbreviation dictionary and then returning the time zone for that name. Returns `nil` if there is no match for *abbreviation*.

Discussion

In general, you are discouraged from using abbreviations except for unique instances such as "UTC" or "GMT". Time Zone abbreviations are not standardized and so a given abbreviation may have multiple meanings—for example, "EST" refers to Eastern Time in both the United States and Australia

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ abbreviationDictionary` (page 1666)

`+ timeZoneForSecondsFromGMT:` (page 1669)

`+ timeZoneWithName:` (page 1670)

Declared In

`NSTimeZone.h`

timeZoneWithName:

Returns the time zone object identified by a given ID.

```
+ (id)timeZoneWithName:(NSString *)aTimeZoneName
```

Parameters

aName

The ID for the time zone.

Return Value

The time zone in the information directory with a name matching *aName*. Returns `nil` if there is no match for the name.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ timeZoneForSecondsFromGMT:` (page 1669)

`+ timeZoneWithAbbreviation:` (page 1669)

`+ knownTimeZoneNames` (page 1667)

Declared In

`NSTimeZone.h`

timeZoneWithName:data:

Returns the time zone with a given ID whose data has been initialized using given data,

```
+ (id)timeZoneWithName:(NSString *)aTimeZoneName data:(NSData *)data
```

Parameters

aTimeZoneName

The ID for the time zone.

data

The data from the time-zone files located at `/usr/share/zoneinfo`.

Return Value

The time zone with the ID *aTimeZoneName* whose data has been initialized using the contents of *data*.

Discussion

You should not call this method directly—use `timeZoneWithName:` (page 1670) to get the time zone object for a given name.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ timeZoneWithName:` (page 1670)

Declared In

`NSTimeZone.h`

Instance Methods

abbreviation

Returns the abbreviation for the receiver.

```
- (NSString *)abbreviation
```

Return Value

The abbreviation for the receiver, such as “EDT” (Eastern Daylight Time).

Discussion

Invokes `abbreviationForDate:` (page 1671) with the current date as the argument.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSTimeZone.h`

abbreviationForDate:

Returns the abbreviation for the receiver at a given date.

```
- (NSString *)abbreviationForDate:(NSDate *)aDate
```

Parameters

aDate

The date for which to get the abbreviation for the receiver.

Return Value
The abbreviation for the receiver at *aDate*.

Discussion
Note that the abbreviation may be different at different dates. For example, during daylight savings time the US/Eastern time zone has an abbreviation of "EDT." At other times, its abbreviation is "EST."

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSTimeZone.h

data

Returns the data that stores the information used by the receiver.

```
- (NSData *)data
```

Return Value
The data that stores the information used by the receiver.

Discussion
This data should be treated as an opaque object.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSTimeZone.h

daylightSavingTimeOffset

Returns the current daylight saving time offset of the receiver.

```
- (NSTimeInterval)daylightSavingTimeOffset
```

Return Value
The daylight current saving time offset of the receiver.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `- isDaylightSavingTime` (page 1674)
- `- isDaylightSavingTimeForDate:` (page 1675)
- `- daylightSavingTimeOffsetForDate:` (page 1673)

Declared In
NSTimeZone.h

daylightSavingTimeOffsetForDate:

Returns the daylight saving time offset for a given date.

```
- (NSTimeInterval)daylightSavingTimeOffsetForDate:(NSDate *)aDate
```

Parameters

aDate

A date.

Return Value

The daylight saving time offset for *aDate*.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `isDaylightSavingTime` (page 1674)
- `daylightSavingTimeOffset` (page 1672)
- `isDaylightSavingTimeForDate:` (page 1675)
- `nextDaylightSavingTimeTransitionAfterDate:` (page 1677)

Declared In

`NSTimeZone.h`

description

Returns the description of the receiver.

```
- (NSString *)description
```

Return Value

The description of the receiver, including the name, abbreviation, offset from GMT, and whether or not daylight savings time is currently in effect.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSTimeZone.h`

initWithName:

Returns a time zone initialized with a given ID.

```
- (id)initWithName:(NSString *)aName
```

Parameters

aName

The ID for the time zone.

Return Value

A time zone object initialized with the ID *aName*.

Discussion
If *aName* is a known ID, this method calls `initWithName:data:` (page 1674) with the appropriate data object.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSTimeZone.h`

initWithName:data:

Initializes a time zone with a given ID and time zone data.

```
- (id)initWithName:(NSString *)aName data:(NSData *)data
```

Parameters

aName

The ID for the time zone.

data

The data from the time-zone files located at `/usr/share/zoneinfo`.

Discussion
You should not call this method directly—use `initWithName:` (page 1673) to get a time zone object.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSTimeZone.h`

isDaylightSavingTime

Returns a Boolean value that indicates whether the receiver is currently using daylight saving time.

```
- (BOOL)isDaylightSavingTime
```

Return Value
`YES` if the receiver is currently using daylight savings time, otherwise `NO`.

Discussion
This method invokes `isDaylightSavingTimeForDate:` (page 1675) with the current date as the argument.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `isDaylightSavingTimeForDate:` (page 1675)
- `daylightSavingTimeOffset` (page 1672)
- `daylightSavingTimeOffsetForDate:` (page 1673)
- `nextDaylightSavingTimeTransition` (page 1676)
- `nextDaylightSavingTimeTransitionAfterDate:` (page 1677)

Declared In
NSTimeZone.h

isDaylightSavingTimeForDate:

Returns a Boolean value that indicates whether the receiver uses daylight savings time at a given date.

```
- (BOOL)isDaylightSavingTimeForDate:(NSDate *)aDate
```

Parameters

aDate

The date against which to test the receiver.

Return Value
YES if the receiver uses daylight savings time at *aDate*, otherwise NO.

Availability
Available in Mac OS X v10.0 and later.

See Also
- isDaylightSavingTime (page 1674)
- daylightSavingTimeOffset (page 1672)
- daylightSavingTimeOffsetForDate: (page 1673)
- nextDaylightSavingTimeTransitionAfterDate: (page 1677)

Declared In
NSTimeZone.h

isEqualToTimeZone:

Returns a Boolean value that indicates whether the receiver has the same name and data as another given time zone.

```
- (BOOL)isEqualToTimeZone:(NSTimeZone *)aTimeZone
```

Parameters

aTimeZone

The time zone to compare with the receiver.

Return Value
YES if *aTimeZone* and the receiver have the same name and data, otherwise NO.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSTimeZone.h

localizedName:locale:

Returns the name of the receiver localized for a given locale.

```
- (NSString *)localizedName:(NSTimeZoneNameStyle)style locale:(NSLocale *)locale
```

Parameters

style

The format style for the returned string.

locale

The locale for which to format the name.

Return Value
The name of the receiver localized for *locale* using *style*.

Availability
Available in Mac OS X v10.5 and later.

Declared In
NSTimeZone.h

name

Returns the geopolitical region ID that identifies the receiver.

```
- (NSString *)name
```

Return Value
The geopolitical region ID that identifies the receiver.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSTimeZone.h

nextDaylightSavingTimeTransition

Returns the date of the next daylight saving time transition for the receiver.

```
- (NSDate *)nextDaylightSavingTimeTransition
```

Return Value
The date of the next (after the current instant) daylight saving time transition for the receiver.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `isDaylightSavingTime` (page 1674)
- `isDaylightSavingTimeForDate:` (page 1675)
- `nextDaylightSavingTimeTransitionAfterDate:` (page 1677)

Declared In
NSTimeZone.h

nextDaylightSavingTimeTransitionAfterDate:

Returns the next daylight saving time transition after a given date.

```
- (NSDate *)nextDaylightSavingTimeTransitionAfterDate:(NSDate *)aDate
```

Parameters

aDate

A date.

Return Value

The next daylight saving time transition after *aDate*.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `isDaylightSavingTime` (page 1674)
- `isDaylightSavingTimeForDate:` (page 1675)
- `nextDaylightSavingTimeTransition` (page 1676)

Declared In

`NSTimeZone.h`

secondsFromGMT

Returns the current difference in seconds between the receiver and Greenwich Mean Time.

```
- (NSInteger)secondsFromGMT
```

Return Value

The current difference in seconds between the receiver and Greenwich Mean Time.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSTimeZone.h`

secondsFromGMTForDate:

Returns the difference in seconds between the receiver and Greenwich Mean Time at a given date.

```
- (NSInteger)secondsFromGMTForDate:(NSDate *)aDate
```

Parameters

aDate

The date against which to test the receiver.

Return Value

The difference in seconds between the receiver and Greenwich Mean Time at *aDate*.

Discussion
The difference may be different from the current difference if the time zone changes its offset from GMT at different points in the year—for example, the U.S. time zones change with daylight savings time.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSTimeZone.h`

Constants

NSTimeZoneNameStyle

Defines a type for time zone name styles.

```
typedef NSInteger NSTimeZoneNameStyle;
```

Discussion
See "Time Zone Name Styles" (page 1678) for possible values.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSTimeZone.h`

Time Zone Name Styles

Specify styles for presenting time zone names.

```
enum {
    NSTimeZoneNameStyleStandard,
    NSTimeZoneNameStyleShortStandard,
    NSTimeZoneNameStyleDaylightSaving,
    NSTimeZoneNameStyleShortDaylightSaving
};
```

Constants

`NSTimeZoneNameStyleStandard`
Specifies a standard name style.
Available in Mac OS X v10.5 and later.
Declared in `NSTimeZone.h`.

`NSTimeZoneNameStyleShortStandard`
Specifies a short name style.
Available in Mac OS X v10.5 and later.
Declared in `NSTimeZone.h`.

`NSTimeZoneNameStyleDaylightSaving`
: Specifies a daylight saving name style.

  Available in Mac OS X v10.5 and later.

  Declared in `NSTimeZone.h`.

`NSTimeZoneNameStyleShortDaylightSaving`
: Specifies a short daylight saving name style.

  Available in Mac OS X v10.5 and later.

  Declared in `NSTimeZone.h`.

Declared In
`NSTimeZone.h`

Notifications

NSSystemTimeZoneDidChangeNotification

Sent when the time zone changed.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSTimeZone.h`

NSUnarchiver Class Reference

Inherits from	NSCoder : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSArchiver.h
Companion guide	Archives and Serializations Programming Guide for Cocoa
Related sample code	Departments and Employees MenuItemView QTMetadataEditor Sketch-112 StickiesExample

Overview

`NSUnarchiver`, a concrete subclass of `NSCoder`, defines methods for decoding a set of Objective-C objects from an archive. Such archives are produced by objects of the `NSArchiver` class.

In Mac OS X v10.2 and later, `NSArchiver` and `NSUnarchiver` have been replaced by `NSKeyedArchiver` and `NSKeyedUnarchiver` respectively—see *Archives and Serializations Programming Guide for Cocoa*.

Tasks

Initializing an NSUnarchiver

`- initForReadingWithData:` (page 1686)
Returns an `NSUnarchiver` object initialized to read an archive from a given data object.

Decoding Objects

`+ unarchiveObjectWithData:` (page 1684)
Decodes and returns the object archived in a given `NSData` object.

+ unarchiveObjectWithFile: (page 1684)
Decodes and returns the object archived in the file *path*.

Managing an NSUnarchiver

- isAtEnd (page 1686)
Returns a Boolean value that indicates whether the receiver has reached the end of the encoded data while decoding.
- objectZone (page 1687)
Returns the memory zone used to allocate decoded objects.
- setObjectZone: (page 1688)
Sets the memory zone used to allocate decoded objects.
- systemVersion (page 1688)
Returns the system version number in effect when the archive was created.

Substituting Classes or Objects

+ classNameDecodedForArchiveClassName: (page 1682)
Returns the name of the class used when instantiating objects whose ostensible class, according to the archived data, is a given name.
+ decodeClassName:asClassName: (page 1683)
Instructs instances of NSUnarchiver to use the class with a given name when instantiating objects whose ostensible class, according to the archived data, is another given name.
- classNameDecodedForArchiveClassName: (page 1685)
Returns the name of the class that will be used when instantiating objects whose ostensible class, according to the archived data, is a given name.
- decodeClassName:asClassName: (page 1685)
Instructs the receiver to use the class with a given name when instantiating objects whose ostensible class, according to the archived data, is another given name.
- replaceObject:withObject: (page 1687)
Causes the receiver to substitute one given object for another whenever the latter is extracted from the archive.

Class Methods

classNameDecodedForArchiveClassName:

Returns the name of the class used when instantiating objects whose ostensible class, according to the archived data, is a given name.

```
+ (NSString *)classNameDecodedForArchiveClassName:(NSString *)nameInArchive
```

Parameters

nameInArchive

The name of a class.

Return Value

The name of the class used when instantiating objects whose ostensible class, according to the archived data, is *nameInArchive*. Returns *nameInArchive* if no substitute name has been specified using the class method (not the instance method) `decodeClassName:asClassName:` (page 1683).

Discussion

Note that each individual instance of `NSUnarchiver` can be given its own class name mappings by invoking the instance method `decodeClassName:asClassName:` (page 1685). The `NSUnarchiver` class has no information about these instance-specific mappings, however, so they don't affect the return value of `classNameDecodedForArchiveClassName:`.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- classNameDecodedForArchiveClassName:` (page 1685)

Declared In

`NSArchiver.h`

decodeClassName:asClassName:

Instructs instances of `NSUnarchiver` to use the class with a given name when instantiating objects whose ostensible class, according to the archived data, is another given name.

```
+ (void)decodeClassName:(NSString *)nameInArchive asClassName:(NSString *)trueName
```

Parameters

nameInArchive

The ostensible name of a class in an archive.

trueName

The name of the class to use when instantiating objects whose ostensible class, according to the archived data, is *nameInArchive*.

Discussion

This method enables easy conversion of unarchived data when the name of a class has changed since the archive was created.

Note that there is also an instance method of the same name. An instance of `NSUnarchiver` can maintain its own mapping of class names. However, if both the class method and the instance method have been invoked using an identical value for *nameInArchive*, the class method takes precedence.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ classNameDecodedForArchiveClassName:` (page 1682)

`- decodeClassName:asClassName:` (page 1685)

Declared In
NSArchiver.h

unarchiveObjectWithData:

Decodes and returns the object archived in a given NSData object.

```
+ (id)unarchiveObjectWithData:(NSData *)data
```

Parameters

data

An NSData object that contains an archive created using NSArchiver.

Return Value
The object, or object graph, that was archived in *data*. Returns nil if *data* cannot be unarchived.

Discussion
This method invokes initForReadingWithData: (page 1686) and decodeObject (page 279) to create a temporary NSUnarchiver object that decodes the object. If the archived object is the root of a graph of objects, the entire graph is unarchived.

Availability
Available in Mac OS X v10.0 and later.

See Also
encodeRootObject: (page 101) (NSArchiver)

Related Sample Code
Departments and Employees
MenuItemView
QTMetadataEditor
Sketch-112
StickiesExample

Declared In
NSArchiver.h

unarchiveObjectWithFile:

Decodes and returns the object archived in the file *path*.

```
+ (id)unarchiveObjectWithFile:(NSString *)path
```

Parameters

path

The path to a file than contains an archive created using NSArchiver.

Return Value
The object, or object graph, that was archived in the file at *path*. Returns nil if the file at *path* cannot be unarchived.

Discussion
This convenience method reads the file by invoking the `NSData` method `dataWithContentsOfFile:` (page 372) and then invokes `unarchiveObjectWithData:` (page 1684).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSArchiver.h`

Instance Methods

classNameDecodedForArchiveClassName:

Returns the name of the class that will be used when instantiating objects whose ostensible class, according to the archived data, is a given name.

```
- (NSString *)classNameDecodedForArchiveClassName:(NSString *)nameInArchive
```

Parameters

nameInArchive
The ostensible name of a class in an archive.

Return Value
The name of the class that will be used when instantiating objects whose ostensible class, according to the archived data, is *nameInArchive*. Returns *nameInArchive* unless a substitute name has been specified using the instance method (not the class method) `decodeClassName:asClassName:` (page 1685).

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ classNameDecodedForArchiveClassName:` (page 1682)

Declared In
`NSArchiver.h`

decodeClassName:asClassName:

Instructs the receiver to use the class with a given name when instantiating objects whose ostensible class, according to the archived data, is another given name.

```
- (void)decodeClassName:(NSString *)nameInArchive asClassName:(NSString *)trueName
```

Parameters

nameInArchive
The ostensible name of a class in an archive.

trueName
The name of the class to use when instantiating objects whose ostensible class, according to the archived data, is *nameInArchive*.

Discussion
This method enables easy conversion of unarchived data when the name of a class has changed since the archive was created.

Note that there's also a class method of the same name. The class method has precedence in case of conflicts.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `classNameDecodedForArchiveClassName:` (page 1685)
+ `decodeClassName:asClassName:` (page 1683)

Declared In
`NSArchiver.h`

initForReadingWithData:

Returns an `NSUnarchiver` object initialized to read an archive from a given data object.

```
- (id)initForReadingWithData:(NSData *)data
```

Parameters
data
The archive data.

Return Value
An `NSUnarchiver` object initialized to read an archive from *data*. Returns `nil` if *data* is not a valid archive.

Discussion
The method decodes the system version number that was archived in *data* prepares the `NSUnarchiver` object for a subsequent invocation of `decodeObject` (page 279).

Raises an `NSInvalidArgumentException` if *data* is `nil`.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `systemVersion` (page 1688)

Declared In
`NSArchiver.h`

isAtEnd

Returns a Boolean value that indicates whether the receiver has reached the end of the encoded data while decoding.

```
- (BOOL)isAtEnd
```

Return Value
`YES` if the receiver has reached the end of the encoded data while decoding, otherwise `NO`.

Discussion
You can invoke this method after invoking `decodeObject` to discover whether the archive contains extra data following the encoded object graph. If it does, you can either ignore this anomaly or consider it an error.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSArchiver.h`

objectZone

Returns the memory zone used to allocate decoded objects.

```
- (NSZone *)objectZone
```

Return Value
The memory zone used to allocate decoded objects.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- setObjectZone:` (page 1688)

Declared In
`NSArchiver.h`

replaceObject:withObject:

Causes the receiver to substitute one given object for another whenever the latter is extracted from the archive.

```
- (void)replaceObject:(id)object withObject:(id)newObject
```

Parameters
object
The archived object to replace.

newObject
The object with which to replace *object*.

Discussion
newObject can be of a different class from object, and the class mappings set by `classNameDecodedForArchiveClassName:` (page 1682) and `decodeClassName:asClassName:` (page 1685) are ignored.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSArchiver.h`

setObjectZone:

Sets the memory zone used to allocate decoded objects.

```
- (void)setObjectZone:(NSZone *)zone
```

Parameters

zone

The memory zone used to allocate decoded objects.

Discussion

If *zone* is `nil`, or if this method is never invoked, the default zone is used, as given by `NSDefaultMallocZone()`.

Availability

Available in Mac OS X v10.0 and later.

See Also

- objectZone (page 1687)

Declared In

`NSArchiver.h`

systemVersion

Returns the system version number in effect when the archive was created.

```
- (unsigned)systemVersion
```

Return Value

The system version number in effect when the archive was created.

Discussion

This information is available as soon as the receiver has been initialized.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSArchiver.h`

NSUndoManager Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSUndoManager.h
Companion guide	Undo Architecture
Related sample code	CoreRecipes iSpend Quartz Composer WWDC 2005 TextEdit Sketch-112 TextEditPlus

Overview

`NSUndoManager` is a general-purpose recorder of operations for undo and redo.

You register an undo operation by specifying the object that's changing (or the owner of that object), along with a method to invoke to revert its state, and the arguments for that method. When performing undo an `NSUndoManager` saves the operations reverted so that you can redo the undos. If used in a Cocoa Application Kit-based application, `NSUndoManager` groups all operations within a single cycle of the run loop, so that performing an undo reverts all changes that occurred during the cycle.

`NSUndoManager` is implemented as a class of the Foundation framework because executables other than applications might want to revert changes to their states. For example, you might have an interactive command-line tool with undo and redo commands, or there could be distributed object implementations that can revert operations "over the wire." However, users typically see undo and redo as application features. The Application Kit implements undo and redo in its NSTextView object and makes it easy to implement it in objects along the responder chain.

Tasks

Registering Undo Operations

- `registerUndoWithTarget:selector:object:` (page 1700)
  Records a single undo operation for a given target, so that when an undo is performed it is sent a specified selector with a given object as the sole argument.
- `prepareWithInvocationTarget:` (page 1698)
  Prepares the receiver for invocation-based undo with the given target as the subject of the next undo operation and returns `self`.
- `forwardInvocation:` (page 1695)
  Overrides `NSObject`'s implementation to record the given invocation as an undo operation.

Checking Undo Ability

- `canUndo` (page 1693)
  Returns a Boolean value that indicates whether the receiver has any actions to undo.
- `canRedo` (page 1693)
  Returns a Boolean value that indicates whether the receiver has any actions to redo.

Performing Undo and Redo

- `undo` (page 1704)
  Closes the top-level undo group if necessary and invokes `undoNestedGroup` (page 1706).
- `undoNestedGroup` (page 1706)
  Performs the undo operations in the last undo group (whether top-level or nested), recording the operations on the redo stack as a single group.
- `redo` (page 1699)
  Performs the operations in the last group on the redo stack, if there are any, recording them on the undo stack as a single group.

Limiting the Undo Stack

- `setLevelsOfUndo:` (page 1703)
  Sets the maximum number of top-level undo groups the receiver holds.
- `levelsOfUndo` (page 1698)
  Returns the maximum number of top-level undo groups the receiver holds.

Creating Undo Groups

- `beginUndoGrouping` (page 1692)
  Marks the beginning of an undo group.

- `endUndoGrouping` (page 1694)
  Marks the end of an undo group.
- `enableUndoRegistration` (page 1694)
  Enables the recording of undo operations.
- `groupsByEvent` (page 1696)
  Returns a Boolean value that indicates whether the receiver automatically creates undo groups around each pass of the run loop.
- `setGroupsByEvent:` (page 1703)
  Sets a Boolean value that specifies whether the receiver automatically groups undo operations during the run loop.
- `groupingLevel` (page 1696)
  Returns the number of nested undo groups (or redo groups, if Redo was invoked last) in the current event loop.

Disabling Undo

- `disableUndoRegistration` (page 1694)
  Disables the recording of undo operations, whether by `registerUndoWithTarget:selector:object:` (page 1700) or by invocation-based undo.
- `isUndoRegistrationEnabled` (page 1697)
  Returns a Boolean value that indicates whether the recording of undo operations is enabled.

Checking Whether Undo or Redo Is Being Performed

- `isUndoing` (page 1697)
  Returns a Boolean value that indicates whether the receiver is in the process of performing its `undo` (page 1704) or `undoNestedGroup` (page 1706) method.
- `isRedoing` (page 1696)
  Returns a Boolean value that indicates whether the receiver is in the process of performing its `redo` (page 1699) method.

Clearing Undo Operations

- `removeAllActions` (page 1701)
  Clears the undo and redo stacks and re-enables the receiver.
- `removeAllActionsWithTarget:` (page 1701)
  Clears the undo and redo stacks of all operations involving the specified target as the recipient of the undo message.

Managing the Action Name

- `setActionName:` (page 1702)
  Sets the name of the action associated with the Undo or Redo command.

- `redoActionName` (page 1699)
  Returns the name identifying the redo action.
- `undoActionName` (page 1705)
  Returns the name identifying the undo action.

Getting and Localizing the Menu Item Title

- `redoMenuItemTitle` (page 1699)
  Returns the complete title of the Redo menu command, for example, “Redo Paste.”
- `undoMenuItemTitle` (page 1705)
  Returns the complete title of the Undo menu command, for example, “Undo Paste.”
- `redoMenuTitleForUndoActionName:` (page 1700)
  Returns the complete, localized title of the Redo menu command for the action identified by the given name.
- `undoMenuTitleForUndoActionName:` (page 1706)
  Returns the complete, localized title of the Undo menu command for the action identified by the given name.

Working with Run Loops

- `runLoopModes` (page 1702)
  Returns the modes governing the types of input handled during a cycle of the run loop.
- `setRunLoopModes:` (page 1704)
  Sets the modes that determine the types of input handled during a cycle of the run loop.

Instance Methods

beginUndoGrouping

Marks the beginning of an undo group.

```
- (void)beginUndoGrouping
```

Discussion
All individual undo operations before a subsequent `endUndoGrouping` (page 1694) message are grouped together and reversed by a later `undo` (page 1704) message. By default undo groups are begun automatically at the start of the event loop, but you can begin your own undo groups with this method, and nest them within other groups.

This method posts an `NSUndoManagerCheckpointNotification` (page 1707) unless a top-level undo is in progress. It posts an `NSUndoManagerDidOpenUndoGroupNotification` (page 1707) if a new group was successfully created.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSUndoManager.h

canRedo

Returns a Boolean value that indicates whether the receiver has any actions to redo.

```
- (BOOL)canRedo
```

Return Value
YES if the receiver has any actions to redo, otherwise NO.

Discussion
Because any undo operation registered clears the redo stack, this method posts an NSUndoManagerCheckpointNotification (page 1707) to allow clients to apply their pending operations before testing the redo stack.

Availability
Available in Mac OS X v10.0 and later.

See Also
- canUndo (page 1693)
- redo (page 1699)

Declared In
NSUndoManager.h

canUndo

Returns a Boolean value that indicates whether the receiver has any actions to undo.

```
- (BOOL)canUndo
```

Return Value
YES if the receiver has any actions to undo, otherwise NO.

Discussion
The return value does not mean you can safely invoke undo (page 1704) or undoNestedGroup (page 1706)—you may have to close open undo groups first.

Availability
Available in Mac OS X v10.0 and later.

See Also
- canRedo (page 1693)
- enableUndoRegistration (page 1694)
- registerUndoWithTarget:selector:object: (page 1700)

Declared In
NSUndoManager.h

disableUndoRegistration

Disables the recording of undo operations, whether by `registerUndoWithTarget:selector:object:` (page 1700) or by invocation-based undo.

```
- (void)disableUndoRegistration
```

Discussion
This method can be invoked multiple times by multiple clients. The `enableUndoRegistration` (page 1694) method must be invoked an equal number of times to re-enable undo registration.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Departments and Employees
File Wrappers with Core Data Documents

Declared In
`NSUndoManager.h`

enableUndoRegistration

Enables the recording of undo operations.

```
- (void)enableUndoRegistration
```

Discussion
Because undo registration is enabled by default, it is often used to balance a prior `disableUndoRegistration` (page 1694) message. Undo registration isn't actually re-enabled until an enable message balances the last disable message in effect. Raises an `NSInternalInconsistencyException` if invoked while no `disableUndoRegistration` (page 1694) message is in effect.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Departments and Employees
File Wrappers with Core Data Documents

Declared In
`NSUndoManager.h`

endUndoGrouping

Marks the end of an undo group.

```
- (void)endUndoGrouping
```

Discussion
All individual undo operations back to the matching `beginUndoGrouping` (page 1692) message are grouped together and reversed by a later `undo` (page 1704) or `undoNestedGroup` (page 1706) message. Undo groups can be nested, thus providing functionality similar to nested transactions. Raises an `NSInternalInconsistencyException` if there's no `beginUndoGrouping` (page 1692) message in effect.

This method posts an `NSUndoManagerCheckpointNotification` (page 1707) and an `NSUndoManagerWillCloseUndoGroupNotification` (page 1708) just before the group is closed.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `levelsOfUndo` (page 1698)

Declared In
`NSUndoManager.h`

forwardInvocation:

Overrides `NSObject`'s implementation to record the given invocation as an undo operation.

```
- (void)forwardInvocation:(NSInvocation *)anInvocation
```

Parameters
anInvocation
The invocation to record.

Discussion
Also clears the redo stack. *anInvocation* and its arguments that are objects are retained. You can override this method if you want different or supplementary invocation-based behavior. See "Registering Undo Operations" for more information.

Raises an `NSInternalInconsistencyException` if `prepareWithInvocationTarget:` (page 1698) was not invoked before this method. This method then clears the prepared invocation target. Also raises an `NSInternalInconsistencyException` if invoked when no undo group has been established using `beginUndoGrouping` (page 1692). Undo groups are normally set by default, so you should rarely need to begin a top-level undo group explicitly.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

See Also
- `undoNestedGroup` (page 1706)
- `registerUndoWithTarget:selector:object:` (page 1700)
- `groupingLevel` (page 1696)

Declared In
`NSUndoManager.h`

groupingLevel

Returns the number of nested undo groups (or redo groups, if Redo was invoked last) in the current event loop.

```
- (NSInteger)groupingLevel
```

Return Value
An integer indicating the number of nested groups. If 0 is returned, there is no open undo or redo group.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `levelsOfUndo` (page 1698)
- `setLevelsOfUndo:` (page 1703)

Declared In
`NSUndoManager.h`

groupsByEvent

Returns a Boolean value that indicates whether the receiver automatically creates undo groups around each pass of the run loop.

```
- (BOOL)groupsByEvent
```

Return Value
`YES` if the receiver automatically creates undo groups around each pass of the run loop, otherwise `NO`.

Discussion
The default is `YES`.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `beginUndoGrouping` (page 1692)
- `setGroupsByEvent:` (page 1703)

Declared In
`NSUndoManager.h`

isRedoing

Returns a Boolean value that indicates whether the receiver is in the process of performing its `redo` (page 1699) method.

```
- (BOOL)isRedoing
```

Return Value
`YES` if the method is being performed, otherwise `NO`.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `isUndoing` (page 1697)

Declared In
`NSUndoManager.h`

isUndoing

Returns a Boolean value that indicates whether the receiver is in the process of performing its `undo` (page 1704) or `undoNestedGroup` (page 1706) method.

```
- (BOOL)isUndoing
```

Return Value
`YES` if the method is being performed, otherwise `NO`.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `isRedoing` (page 1696)

Declared In
`NSUndoManager.h`

isUndoRegistrationEnabled

Returns a Boolean value that indicates whether the recording of undo operations is enabled.

```
- (BOOL)isUndoRegistrationEnabled
```

Return Value
`YES` if registration is enabled; otherwise, `NO`.

Discussion
Undo registration is enabled by default.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `disableUndoRegistration` (page 1694)
- `enableUndoRegistration` (page 1694)

Declared In
`NSUndoManager.h`

levelsOfUndo

Returns the maximum number of top-level undo groups the receiver holds.

```
- (NSUInteger)levelsOfUndo
```

Return Value
An integer specifying the number of undo groups. A limit of `0` indicates no limit, so old undo groups are never dropped.

Discussion
When ending an undo group results in the number of groups exceeding this limit, the oldest groups are dropped from the stack. The default is `0`.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `enableUndoRegistration` (page 1694)
- `setLevelsOfUndo:` (page 1703)

Declared In
`NSUndoManager.h`

prepareWithInvocationTarget:

Prepares the receiver for invocation-based undo with the given target as the subject of the next undo operation and returns `self`.

```
- (id)prepareWithInvocationTarget:(id)target
```

Parameters
target
The target of the undo operation.

Return Value
`self`.

Discussion
See "Registering Undo Operations" for more information.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `forwardInvocation:` (page 1695)

Related Sample Code
Squiggles

Declared In
`NSUndoManager.h`

redo

Performs the operations in the last group on the redo stack, if there are any, recording them on the undo stack as a single group.

```
- (void)redo
```

Discussion
Raises an `NSInternalInconsistencyException` if the method is invoked during an undo operation.

This method posts an `NSUndoManagerCheckpointNotification` (page 1707) and `NSUndoManagerWillRedoChangeNotification` (page 1708) before it performs the redo operation, and it posts the `NSUndoManagerDidRedoChangeNotification` (page 1708) after it performs the redo operation.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `registerUndoWithTarget:selector:object:` (page 1700)

Declared In
`NSUndoManager.h`

redoActionName

Returns the name identifying the redo action.

```
- (NSString *)redoActionName
```

Return Value
The redo action name. Returns an empty string (@"") if no action name has been assigned or if there is nothing to redo.

Discussion
For example, if the menu title is “Redo Delete,” the string returned is “Delete.”

Availability
Available in Mac OS X v10.0 and later.

See Also
- `setActionName:` (page 1702)
- `undoActionName` (page 1705)

Declared In
`NSUndoManager.h`

redoMenuItemTitle

Returns the complete title of the Redo menu command, for example, “Redo Paste.”

```
- (NSString *)redoMenuItemTitle
```

Return Value
The menu item title.

Discussion
Returns “Redo” if no action name has been assigned or `nil` if there is nothing to redo.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- undoMenuItemTitle` (page 1705)

Declared In
`NSUndoManager.h`

redoMenuTitleForUndoActionName:

Returns the complete, localized title of the Redo menu command for the action identified by the given name.

```
- (NSString *)redoMenuTitleForUndoActionName:(NSString *)actionName
```

Parameters
actionName
The name of the undo action.

Return Value
The localized title of the redo menu item.

Discussion
Override this method if you want to customize the localization behavior. This method is invoked by `redoMenuItemTitle` (page 1699).

Availability
Available in Mac OS X v10.0 and later.

See Also
`- undoMenuTitleForUndoActionName:` (page 1706)

Declared In
`NSUndoManager.h`

registerUndoWithTarget:selector:object:

Records a single undo operation for a given target, so that when an undo is performed it is sent a specified selector with a given object as the sole argument.

```
- (void)registerUndoWithTarget:(id)target selector:(SEL)aSelector object:(id)anObject
```

Parameters
target
The target of the undo operation.
aSelector
The selector for the undo operation.
anObject
The argument sent with the selector.

Discussion
Also clears the redo stack. Does not retain *target*, but does retain *anObject*. See "Registering Undo Operations" for more information.

Raises an `NSInternalInconsistencyException` if invoked when no undo group has been established using `beginUndoGrouping` (page 1692). Undo groups are normally set by default, so you should rarely need to begin a top-level undo group explicitly.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `undoNestedGroup` (page 1706)
- `forwardInvocation:` (page 1695)
- `groupingLevel` (page 1696)

Related Sample Code
File Wrappers with Core Data Documents

Declared In
`NSUndoManager.h`

removeAllActions

Clears the undo and redo stacks and re-enables the receiver.

```
- (void)removeAllActions
```

Availability
Available in Mac OS X v10.0 and later.

See Also
- `enableUndoRegistration` (page 1694)
- `removeAllActionsWithTarget:` (page 1701)

Related Sample Code
Departments and Employees

Declared In
`NSUndoManager.h`

removeAllActionsWithTarget:

Clears the undo and redo stacks of all operations involving the specified target as the recipient of the undo message.

```
- (void)removeAllActionsWithTarget:(id)target
```

Parameters
target
The recipient of the undo messages to be removed.

Discussion
Doesn't re-enable the receiver if it's disabled. An object that shares an `NSUndoManager` with other clients should invoke this message in its implementation of `dealloc`.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `enableUndoRegistration` (page 1694)
- `removeAllActions` (page 1701)

Declared In
`NSUndoManager.h`

runLoopModes

Returns the modes governing the types of input handled during a cycle of the run loop.

```
- (NSArray *)runLoopModes
```

Return Value
An array of string constants specifying the current run-loop modes.

Discussion
By default, the sole run-loop mode is `NSDefaultRunLoopMode` (which excludes data from `NSConnection` objects).

Availability
Available in Mac OS X v10.0 and later.

See Also
- `setRunLoopModes:` (page 1704)
- `performSelector:target:argument:order:modes:` (page 1336) (`NSRunLoop`)

Declared In
`NSUndoManager.h`

setActionName:

Sets the name of the action associated with the Undo or Redo command.

```
- (void)setActionName:(NSString *)actionName
```

Parameters
actionName
The name of the action.

Discussion
If *actionName* is an empty string, the action name currently associated with the menu command is removed. There is no effect if *actionName* is `nil`.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `redoActionName` (page 1699)
- `undoActionName` (page 1705)

Related Sample Code
Sketch-112

Declared In
`NSUndoManager.h`

setGroupsByEvent:

Sets a Boolean value that specifies whether the receiver automatically groups undo operations during the run loop.

```
- (void)setGroupsByEvent:(BOOL)flag
```

Parameters

flag

If `YES`, the receiver creates undo groups around each pass through the run loop; if `NO` it doesn't.

Discussion
The default is `YES`. If you turn automatic grouping off, you must close groups explicitly before invoking either `undo` (page 1704) or `undoNestedGroup` (page 1706).

Availability
Available in Mac OS X v10.0 and later.

See Also
- `groupingLevel` (page 1696)
- `groupsByEvent` (page 1696)

Declared In
`NSUndoManager.h`

setLevelsOfUndo:

Sets the maximum number of top-level undo groups the receiver holds.

```
- (void)setLevelsOfUndo:(NSUInteger)anInt
```

Parameters

anInt

The maximum number of undo groups. A limit of `0` indicates no limit, so that old undo groups are never dropped.

Discussion
When ending an undo group results in the number of groups exceeding this limit, the oldest groups are dropped from the stack. The default is `0`.

If invoked with a limit below the prior limit, old undo groups are immediately dropped.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `enableUndoRegistration` (page 1694)
- `levelsOfUndo` (page 1698)

Declared In
`NSUndoManager.h`

setRunLoopModes:

Sets the modes that determine the types of input handled during a cycle of the run loop.

```
- (void)setRunLoopModes:(NSArray *)modes
```

Parameters

modes

An array of string constants specifying the run-loop modes to set.

Discussion
By default, the sole run-loop mode is `NSDefaultRunLoopMode` (which excludes data from `NSConnection` objects). With this method, you could limit the input to data received during a mouse-tracking session by setting the mode to `NSEventTrackingRunLoopMode`, or you could limit it to data received from a modal panel with `NSModalPanelRunLoopMode`.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `runLoopModes` (page 1702)
- `performSelector:target:argument:order:modes:` (page 1336) (`NSRunLoop`)

Declared In
`NSUndoManager.h`

undo

Closes the top-level undo group if necessary and invokes `undoNestedGroup` (page 1706).

```
- (void)undo
```

Discussion
This method also invokes `endUndoGrouping` (page 1694) if the nesting level is 1. Raises an `NSInternalInconsistencyException` if more than one undo group is open (that is, if the last group isn't at the top level).

This method posts an `NSUndoManagerCheckpointNotification` (page 1707).

Availability
Available in Mac OS X v10.0 and later.

See Also
- `enableUndoRegistration` (page 1694)
- `groupingLevel` (page 1696)

Declared In
`NSUndoManager.h`

undoActionName

Returns the name identifying the undo action.

```
- (NSString *)undoActionName
```

Return Value
The undo action name. Returns an empty string (`@""`) if no action name has been assigned or if there is nothing to undo.

Discussion
For example, if the menu title is “Undo Delete,” the string returned is “Delete.”

Availability
Available in Mac OS X v10.0 and later.

See Also
- `redoActionName` (page 1699)
- `setActionName:` (page 1702)

Declared In
`NSUndoManager.h`

undoMenuItemTitle

Returns the complete title of the Undo menu command, for example, “Undo Paste.”

```
- (NSString *)undoMenuItemTitle
```

Return Value
The menu item title.

Discussion
Returns “Undo” if no action name has been assigned or `nil` if there is nothing to undo.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `redoMenuItemTitle` (page 1699)

Declared In
`NSUndoManager.h`

undoMenuTitleForUndoActionName:

Returns the complete, localized title of the Undo menu command for the action identified by the given name.

```
- (NSString *)undoMenuTitleForUndoActionName:(NSString *)actionName
```

Parameters

actionName

The name of the undo action.

Return Value

The localized title of the undo menu item.

Discussion

Override this method if you want to customize the localization behavior. This method is invoked by `undoMenuItemTitle` (page 1705).

Availability

Available in Mac OS X v10.0 and later.

See Also

`- redoMenuTitleForUndoActionName:` (page 1700)

Declared In

`NSUndoManager.h`

undoNestedGroup

Performs the undo operations in the last undo group (whether top-level or nested), recording the operations on the redo stack as a single group.

```
- (void)undoNestedGroup
```

Discussion

Raises an `NSInternalInconsistencyException` if any undo operations have been registered since the last `enableUndoRegistration` (page 1694) message.

This method posts an `NSUndoManagerCheckpointNotification` (page 1707) and `NSUndoManagerWillUndoChangeNotification` (page 1708) before it performs the undo operation, and it posts an `NSUndoManagerDidUndoChangeNotification` (page 1708) after it performs the undo operation.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- undo` (page 1704)

Declared In

`NSUndoManager.h`

Constants

NSUndoCloseGroupingRunLoopOrdering

`NSUndoManager` provides this constant as a convenience; you can use it to compare to values returned by some `NSUndoManager` methods.

```
enum {
   NSUndoCloseGroupingRunLoopOrdering = 350000
};
```

Constants

`NSUndoCloseGroupingRunLoopOrdering`

Used with `NSRunLoop`'s `performSelector:target:argument:order:modes:` (page 1336).

Available in Mac OS X v10.0 and later.

Declared in `NSUndoManager.h`.

Declared In

`NSUndoManager.h`

Notifications

NSUndoManagerCheckpointNotification

Posted whenever an `NSUndoManager` object opens or closes an undo group (except when it opens a top-level group) and when checking the redo stack in `canRedo` (page 1693). The notification object is the `NSUndoManager` object. This notification does not contain a *`userInfo`* dictionary.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSUndoManager.h`

NSUndoManagerDidOpenUndoGroupNotification

Posted whenever an `NSUndoManager` object opens an undo group, which occurs in the implementation of the `beginUndoGrouping` (page 1692) method. The notification object is the `NSUndoManager` object. This notification does not contain a *`userInfo`* dictionary.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSUndoManager.h`

NSUndoManagerDidRedoChangeNotification

Posted just after an `NSUndoManager` object performs a redo operation (`redo` (page 1699)). The notification object is the `NSUndoManager` object. This notification does not contain a *`userInfo`* dictionary.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSUndoManager.h`

NSUndoManagerDidUndoChangeNotification

Posted just after an `NSUndoManager` object performs an undo operation. If you invoke `undo` (page 1704) or `undoNestedGroup` (page 1706), this notification is posted. The notification object is the `NSUndoManager` object. This notification does not contain a *`userInfo`* dictionary.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSUndoManager.h`

NSUndoManagerWillCloseUndoGroupNotification

Posted before an `NSUndoManager` object closes an undo group, which occurs in the implementation of the `endUndoGrouping` (page 1694) method. The notification object is the `NSUndoManager` object. This notification does not contain a *`userInfo`* dictionary.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSUndoManager.h`

NSUndoManagerWillRedoChangeNotification

Posted just before an `NSUndoManager` object performs a redo operation (`redo` (page 1699)). The notification object is the `NSUndoManager` object. This notification does not contain a *`userInfo`* dictionary.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSUndoManager.h`

NSUndoManagerWillUndoChangeNotification

Posted just before an `NSUndoManager` object performs an undo operation. If you invoke `undo` (page 1704) or `undoNestedGroup` (page 1706), this notification is posted. The notification object is the `NSUndoManager` object. This notification does not contain a *`userInfo`* dictionary.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSUndoManager.h`

NSUniqueIDSpecifier Class Reference

Inherits from	NSScriptObjectSpecifier : NSObject
Conforms to	NSCoding (NSScriptObjectSpecifier) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSScriptObjectSpecifiers.h
Availability	Available in Mac OS X v10.2 and later.
Companion guide	Cocoa Scripting Guide
Related sample code	SimpleScriptingObjects

Overview

Specifies an object in a collection (or container) by unique ID. This specifier works only for objects that have an ID property. The unique ID object passed to an instance of `NSUniqueIDSpecifier` must be either an `NSNumber` object or an `NSString` object. The exact type should match the scripting dictionary declaration of the ID attribute for the relevant scripting class.

You can expect that the ID property will be *read only* for any object that supports it. Therefore a scripter can obtain the unique ID for an object and refer to the object by the ID, but cannot set the unique ID.

You don't normally subclass `NSUniqueIDSpecifier`.

The evaluation of `NSUniqueIDSpecifier` objects follows these steps until the specified object is found:

1. If the container implements a method whose selector matches the relevant `valueIn<Key>WithUniqueID:` pattern established by scripting key-value coding, the method is invoked. This method can potentially be very fast, and it may be relatively easy to implement.

2. As is the case when evaluating any script object specifier, the container of the specified object is given a chance to evaluate the object specifier. If the container class implements the `indicesOfObjectsByEvaluatingObjectSpecifier:` (page 2123) method, the method is invoked. This method can potentially be very fast, but it is relatively difficult to implement.

3. An `NSWhoseSpecifier` object that specifies the first object whose relevant `'ID '` attribute matches the ID is synthesized and evaluated. The `NSWhoseSpecifier` object must search through all of the keyed elements in the container, looking for a match. The search is potentially very slow.

Tasks

Initializing a Unique ID Specifier

- `initWithContainerClassDescription:containerSpecifier:key:uniqueID:` (page 1712)
  Returns an `NSUniqueIDSpecifier` object, initialized with the given arguments.

Accessing Unique ID Information

- `setUniqueID:` (page 1713)
  Sets the ID encapsulated by the receiver.
- `uniqueID` (page 1713)
  Returns the ID encapsulated by the receiver.

Instance Methods

initWithContainerClassDescription:containerSpecifier:key:uniqueID:

Returns an `NSUniqueIDSpecifier` object, initialized with the given arguments.

```
- (id)initWithContainerClassDescription:(NSScriptClassDescription *)classDesc
    containerSpecifier:(NSScriptObjectSpecifier *)container key:(NSString *)property
     uniqueID:(id)uniqueID
```

Parameters

classDesc
The class description for the new object.

container
The container for the new object.

property
The property for the new object.

uniqueID
The unique ID for the new object.

uniqueID must be an instance of `NSNumber` or `NSString`. The type should match the declared type of the attribute of the specified scriptable class whose four-character code is `'ID '`.

Return Value
An `NSUniqueIDSpecifier` object, initialized with the given arguments.

Discussion
Invokes the super class's `initWithContainerClassDescription:containerSpecifier:key:` (page 1418) method and sets the ID to *uniqueID*.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SimpleScriptingObjects

Declared In
`NSScriptObjectSpecifiers.h`

setUniqueID:

Sets the ID encapsulated by the receiver.

```
- (void)setUniqueID:(id)uniqueID
```

Parameters

uniqueID

The ID for the receiver.

uniqueID must be an instance of `NSNumber` or `NSString`. The type should match the declared type of the attribute of the specified scriptable class whose four-character code is `'ID '`.

Discussion
Although `NSUniqueIDSpecifier` supports setting the unique ID, the ID for a specified object is likely to remain static over the life of the object.

Availability
Available in Mac OS X v10.2 and later.

See Also
`- uniqueID` (page 1713)

Declared In
`NSScriptObjectSpecifiers.h`

uniqueID

Returns the ID encapsulated by the receiver.

```
- (id)uniqueID
```

Return Value
The ID encapsulated by the receiver.

Availability
Available in Mac OS X v10.2 and later.

See Also
`- setUniqueID:` (page 1713)

Declared In
`NSScriptObjectSpecifiers.h`

NSURL Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSURLHandleClient NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSURL.h
Companion guide	URL Loading System
Related sample code	CoreRecipes ImageClient iSpend LSMSmartCategorizer StickiesExample

Overview

The NSURL class provides a way to manipulate URLs and the resources they reference. NSURL objects understand URLs as specified in RFCs 1808, 1738, and 2732. The litmus test for conformance to RFC 1808 is as recommended in RFC 1808—whether the first two characters of `resourceSpecifier` (page 1730) are `@"//"`.

NSURL objects can be used to refer to files, and are the preferred way to do so. ApplicationKit objects that can read data from or write data to a file generally have methods that accept an NSURL object instead of a pathname as the file reference. NSWorkspace provides `openURL:` to open a location specified by a URL. To get the contents of a URL, NSString provides `stringWithContentsOfURL:` (page 1532) and NSData provides `dataWithContentsOfURL:` (page 374).

An NSURL object is composed of two parts—a potentially `nil` base URL and a string that is resolved relative to the base URL. An NSURL object whose string is fully resolved without a base is considered absolute; all others are considered relative.

The NSURL class will fail to create a new NSURL object if the path being passed is not well-formed—the path must comply with RFC 2396. Examples of cases that will not succeed are strings containing space characters and high-bit characters. Should creating an NSURL object fail, the creation methods will return `nil`, which you must be prepared to handle. If you are creating NSURL objects using file system paths, you should use

`fileURLWithPath:` (page 1718) or `initFileURLWithPath:` (page 1722), which handle the subtle differences between URL paths and file system paths. If you wish to be tolerant of malformed path strings, you'll need to use functions provided by the Core Foundation framework to clean up the strings.

The informal protocol `NSURLClient` defines a set of methods useful for managing the loading of a URL resource in the background.

See also NSURL Additions in the Application Kit framework, which add methods supporting pasteboards.

NSURL is "toll-free bridged" with its Core Foundation counterpart, CFURL. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object, providing you cast one type to the other. In an API where you see an `NSURL *` parameter, you can pass in a `CFURLRef`, and in an API where you see a `CFURLRef` parameter, you can pass in a pointer to an `NSURL` instance. This approach also applies to your concrete subclasses of NSURL. See Interchangeable Data Types for more information on toll-free bridging.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 2034)
- `initWithCoder:` (page 2034)

NSCopying
- `copyWithZone:` (page 2042)

NSURLHandleClient
- `URLHandleResourceDidBeginLoading:` (page 2134)
- `URLHandleResourceDidCancelLoading:` (page 2135)
- `URLHandleResourceDidFinishLoading:` (page 2135)
- `URLHandle:resourceDataDidBecomeAvailable:` (page 2134)
- `URLHandle:resourceDidFailLoadingWithReason:` (page 2134)

Tasks

Creating an NSURL

`- initWithScheme:host:path:` (page 1724)
Initializes a newly created NSURL with a specified scheme, host, and path.

`+ URLWithString:` (page 1720)
Creates and returns an NSURL object initialized with a provided string.

`- initWithString:` (page 1724)
Initializes an NSURL object with a provided string.

`+ URLWithString:relativeToURL:` (page 1720)
Creates and returns an NSURL object initialized with a base URL and a relative string.

- `initWithString:relativeToURL:` (page 1725)
  Initializes an NSURL object with a base URL and a relative string.
+ `fileURLWithPath:isDirectory:` (page 1719)
  Initializes and returns a newly created NSURL object as a file URL with a specified path.
+ `fileURLWithPath:` (page 1718)
  Initializes and returns a newly created NSURL object as a file URL with a specified path.
- `initFileURLWithPath:isDirectory:` (page 1723)
  Initializes a newly created NSURL referencing the local file or directory at *path*.
- `initFileURLWithPath:` (page 1722)
  Initializes a newly created NSURL referencing the local file or directory at *path*.

Identifying and Comparing Objects

- `isEqual:` (page 1725)
  Returns a Boolean value that indicates whether the receiver and a given object are equal.

Querying an NSURL

- `isFileURL` (page 1726)
  Returns whether the receiver uses the file scheme.

Loading the Resource of an NSURL Object

- `loadResourceDataNotifyingClient:usingCache:` (page 1726) Deprecated in Mac OS X v10.4
  Loads the receiver's resource data in the background.
- `propertyForKey:` (page 1728) Deprecated in Mac OS X v10.4
  Returns the specified property of the receiver's resource.
- `resourceDataUsingCache:` (page 1729) Deprecated in Mac OS X v10.4
  Returns the receiver's resource data, loading it if necessary.
- `setProperty:forKey:` (page 1731) Deprecated in Mac OS X v10.4
  Changes the specified property of the receiver's resource.
- `setResourceData:` (page 1731) Deprecated in Mac OS X v10.4
  Attempts to set the resource data for the receiver.
- `URLHandleUsingCache:` (page 1732) Deprecated in Mac OS X v10.4
  Returns a URL handle to service the receiver.

Accessing the Parts of the URL

- `absoluteString` (page 1721)
  Returns the string for the receiver as if it were an absolute URL.
- `absoluteURL` (page 1721)
  Returns an absolute URL that refers to the same resource as the receiver.

- `baseURL` (page 1722)
  Returns the base URL of the receiver.
- `fragment` (page 1722)
  Returns the fragment of a URL conforming to RFC 1808.
- `host` (page 1722)
  Returns the host of a URL conforming to RFC 1808.
- `parameterString` (page 1727)
  Returns the parameter string of a URL conforming to RFC 1808.
- `password` (page 1727)
  Returns the password of a URL conforming to RFC 1808.
- `path` (page 1727)
  Returns the path of a URL conforming to RFC 1808.
- `port` (page 1728)
  Returns the port number of a URL conforming to RFC 1808.
- `query` (page 1728)
  Returns the query of a URL conforming to RFC 1808.
- `relativePath` (page 1729)
  Returns the path of a URL conforming to RFC 1808, without resolving against the receiver's base URL.
- `relativeString` (page 1729)
  Returns a string representation of the relative portion of the URL.
- `resourceSpecifier` (page 1730)
  Returns the resource specifier of the URL.
- `scheme` (page 1730)
  Returns the scheme of the URL.
- `standardizedURL` (page 1731)
  Returns a new NSURL object with any instances of " . . " or " . " removed from its path.
- `user` (page 1732)
  Returns the user portion of a URL conforming to RFC 1808.

Class Methods

fileURLWithPath:

Initializes and returns a newly created NSURL object as a file URL with a specified path.

```
+ (id)fileURLWithPath:(NSString *)path
```

Parameters

path

The path that the NSURL object will represent. *path* should be a valid system path. If *path* begins with a tilde, it must first be expanded with `stringByExpandingTildeInPath` (page 1602).

Return Value

An NSURL object initialized with *path*.

Discussion
This method examines *path* in the file system to determine if it is a directory. If *path* is a directory, then a trailing slash is appended. If the file does not exist, it is assumed that *path* represents a directory and a trailing slash is appended. As an alternative, consider using `fileURLWithPath:isDirectory:` (page 1719) which allows you to explicitly specify whether the returned `NSURL` object represents a file or directory.

Availability
Available in Mac OS X v10.0 and later.

See Also
`initFileURLWithPath:` (page 1722)

Related Sample Code
CoreRecipes
iSpend
Quartz Composer WWDC 2005 TextEdit
StickiesExample
TextEditPlus

Declared In
`NSURL.h`

fileURLWithPath:isDirectory:

Initializes and returns a newly created NSURL object as a file URL with a specified path.

```
+ (id)fileURLWithPath:(NSString *)path
    isDirectory:(BOOL)isDir
```

Parameters

path

The path that the NSURL object will represent. *path* should be a valid system path. If *path* begins with a tilde, it must first be expanded with `stringByExpandingTildeInPath` (page 1602).

isDir

A Boolean value that specifies whether *path* is treated as a directory path when resolving against relative path components. Pass `YES` if the *path* indicates a directory, `NO` otherwise.

Return Value
An NSURL object initialized with *path*.

Availability
Available in Mac OS X v10.5 and later.

See Also
`initFileURLWithPath:` (page 1722)

Related Sample Code
AutoSample
IKSlideshowDemo

Declared In
`NSURL.h`

URLWithString:

Creates and returns an NSURL object initialized with a provided string.

```
+ (id)URLWithString:(NSString *)URLString
```

Parameters

URLString

The string with which to initialize the NSURL object. Must conform to RFC 2396. This method parses *URLString* according to RFCs 1738 and 1808.

Return Value

An NSURL object initialized with *URLString*. If the string was malformed, returns `nil`.

Discussion

This method expects *URLString* to contain any necessary percent escape codes, which are ':', '/', '%', '#', ';', and '@'. Note that '%' escapes are translated via UTF-8.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AutoUpdater

Core Data HTML Store

NewsReader

ObjectPath

VertexPerformanceTest

Declared In

`NSURL.h`

URLWithString:relativeToURL:

Creates and returns an NSURL object initialized with a base URL and a relative string.

```
+ (id)URLWithString:(NSString *)URLString
    relativeToURL:(NSURL *)baseURL
```

Parameters

URLString

The string with which to initialize the NSURL object. May not be `nil`. Must conform to RFC 2396. *URLString* is interpreted relative to *baseURL*.

baseURL

The base URL for the NSURL object.

Return Value

An NSURL object initialized with *URLString* and *baseURL*. If *URLString* was malformed, returns `nil`.

Discussion

This method expects *URLString* to contain any necessary percent escape codes.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
CocoaHTTPServer
CocoaSOAP
Quartz Composer WWDC 2005 TextEdit
Reducer
TextEditPlus

Declared In
`NSURL.h`

Instance Methods

absoluteString

Returns the string for the receiver as if it were an absolute URL.

```
- (NSString *)absoluteString
```

Return Value
An absolute string for the URL. Creating by resolving the receiver's string against its base according to the algorithm given in RFC 1808.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CocoaDragAndDrop
CoreRecipes
NewsReader
Reducer

Declared In
`NSURL.h`

absoluteURL

Returns an absolute URL that refers to the same resource as the receiver.

```
- (NSURL *)absoluteURL
```

Return Value
An absolute URL that refers to the same resource as the receiver. If the receiver is already absolute, returns `self`. Resolution is performed per RFC 1808.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSURL.h`

baseURL

Returns the base URL of the receiver.

```
- (NSURL *)baseURL
```

Return Value
The base URL of the receiver. If the receiver is an absolute URL, returns `nil`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSURL.h`

fragment

Returns the fragment of a URL conforming to RFC 1808.

```
- (NSString *)fragment
```

Return Value
The fragment of the URL. If the receiver does not conform to RFC 1808, returns `nil`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSURL.h`

host

Returns the host of a URL conforming to RFC 1808.

```
- (NSString *)host
```

Return Value
The host of the URL. If the receiver does not conform to RFC 1808, returns `nil`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CoreRecipes

Declared In
`NSURL.h`

initFileURLWithPath:

Initializes a newly created NSURL referencing the local file or directory at *path*.

```
- (id)initFileURLWithPath:(NSString *)path
```

Parameters

path

The path that the NSURL object will represent. *path* should be a valid system path. If *path* begins with a tilde, it must first be expanded with `stringByExpandingTildeInPath` (page 1602).

Return Value
An NSURL object initialized with *path*.

Discussion
Invoking this method is equivalent to invoking `initWithScheme:host:path:` (page 1724) with scheme `NSFileScheme`, a `nil` host, and *path*.

This method examines *path* in the file system to determine if it is a directory. If *path* is a directory, then a trailing slash is appended. If the file does not exist, it is assumed that *path* represents a directory and a trailing slash is appended. As an alternative, consider using `initFileURLWithPath:isDirectory:` (page 1723) which allows you to explicitly specify whether the returned NSURL represents a file or directory.

Availability
Available in Mac OS X v10.0 and later.

See Also
`fileURLWithPath:` (page 1718)

Related Sample Code
AttachAScript
bMoviePaletteCocoa
CoreRecipes
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSURL.h`

initFileURLWithPath:isDirectory:

Initializes a newly created NSURL referencing the local file or directory at *path*.

```
- (id)initFileURLWithPath:(NSString *)path
    isDirectory:(BOOL)isDir
```

Parameters

path

The path that the NSURL object will represent. *path* should be a valid system path. If *path* begins with a tilde, it must first be expanded with `stringByExpandingTildeInPath` (page 1602).

isDir

A Boolean value that specifies whether *path* is treated as a directory path when resolving against relative path components. Pass `YES` if the *path* indicates a directory, `NO` otherwise

Return Value
An NSURL object initialized with *path*.

Discussion
Invoking this method is equivalent to invoking `initWithScheme:host:path:` (page 1724) with scheme `NSFileScheme`, a `nil` host, and *path*.

Availability
Available in Mac OS X v10.5 and later.

See Also
`fileURLWithPath:` (page 1718)

Declared In
`NSURL.h`

initWithScheme:host:path:

Initializes a newly created NSURL with a specified scheme, host, and path.

```
- (id)initWithScheme:(NSString *)scheme
    host:(NSString *)host
    path:(NSString *)path
```

Parameters

scheme
The scheme for the NSURL object.

host
The host for the NSURL object. May be the empty string.

path
The path for the NSURL object. If *path* begins with a tilde, it must first be expanded with `stringByExpandingTildeInPath` (page 1602).

Return Value
The newly initialized NSURL object.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CoreRecipes

Declared In
`NSURL.h`

initWithString:

Initializes an NSURL object with a provided string.

```
- (id)initWithString:(NSString *)URLString
```

Parameters

URLString
The string with which to initialize the NSURL object. Must conform to RFC 2396. This method parses *URLString* according to RFCs 1738 and 1808.

Return Value
An NSURL object initialized with *URLString*. If the string was malformed, returns `nil`.

Discussion
This method expects *URLString* to contain any necessary percent escape codes, which are ':', '/', '%', '#', ';', and '@'. Note that '%' escapes are translated via UTF-8.

Availability
Available in Mac OS X v10.0 and later.

See Also
`URLWithString:` (page 1720)

Declared In
`NSURL.h`

initWithString:relativeToURL:

Initializes an NSURL object with a base URL and a relative string.

```
- (id)initWithString:(NSString *)URLString
    relativeToURL:(NSURL *)baseURL
```

Parameters

URLString
The string with which to initialize the NSURL object. Must conform to RFC 2396. *URLString* is interpreted relative to *baseURL*.

baseURL
The base URL for the NSURL object.

Return Value
An NSURL object initialized with *URLString* and *baseURL*. If *URLString* was malformed, returns `nil`.

Discussion
This method expects *URLString* to contain any necessary percent escape codes.

`initWithString:relativeToURL:` is the designated initializer for NSURL.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- baseURL` (page 1722)
`- relativeString` (page 1729)
`URLWithString:relativeToURL:` (page 1720)

Declared In
`NSURL.h`

isEqual:

Returns a Boolean value that indicates whether the receiver and a given object are equal.

```
- (BOOL)isEqual:(id)anObject
```

Parameters

anObject

The object to be compared to the receiver.

Return Value

YES if the receiver and *anObject* are equal, otherwise NO.

Discussion

This method defines what it means for instances to be equal. For example, two NSURLs are considered equal if they both have the same base baseURL (page 1722) and relativeString (page 1729).

isFileURL

Returns whether the receiver uses the file scheme.

```
- (BOOL)isFileURL
```

Return Value

Returns YES if the receiver uses the file scheme, NO otherwise.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSURL.h

loadResourceDataNotifyingClient:usingCache:

Loads the receiver's resource data in the background. (Deprecated in Mac OS X v10.4.)

```
- (void)loadResourceDataNotifyingClient:(id)client
    usingCache:(BOOL)shouldUseCache
```

Parameters

client

The client of the loading operation. *client* is notified of the receiver's progress loading the resource data using the NSURLClient informal protocol. The NSURLClient messages are delivered on the current thread and require the run loop to be running.

shouldUseCache

Whether the URL should use cached resource data from an already loaded URL that refers to the same resource. If *YES*, the cache is consulted when loading data. If *NO*, the data is always loaded directly, without consulting the cache.

Discussion

A given NSURL object can perform only one background load at a time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
NSURL.h

parameterString

Returns the parameter string of a URL conforming to RFC 1808.

```
- (NSString *)parameterString
```

Return Value
The parameter string of the URL. If the receiver does not conform to RFC 1808, returns `nil`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSURL.h

password

Returns the password of a URL conforming to RFC 1808.

```
- (NSString *)password
```

Return Value
The password of the URL. If the receiver does not conform to RFC 1808, returns `nil`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSURL.h

path

Returns the path of a URL conforming to RFC 1808.

```
- (NSString *)path
```

Return Value
The path of the URL. If the receiver does not conform to RFC 1808, returns `nil`. If `isFileURL` (page 1726) returns`YES`, the return value is suitable for input into NSFileManager or NSPathUtilities. If the path has a trailing slash it is stripped.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CoreRecipes
File Wrappers with Core Data Documents
iSpend
QTKitCreateMovie

Quartz Composer WWDC 2005 TextEdit

Declared In
NSURL.h

port

Returns the port number of a URL conforming to RFC 1808.

```
- (NSNumber *)port
```

Return Value
The port number of the URL. If the receiver does not conform to RFC 1808, returns `nil`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSURL.h

propertyForKey:

Returns the specified property of the receiver's resource. (Deprecated in Mac OS X v10.4.)

```
- (id)propertyForKey:(NSString *)propertyKey
```

Parameters
propertyKey
The key of the desired property.

Return Value
The value of the property of the receiver's resource for the provided key. Returns `nil` if there is no such key.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
`setProperty:forKey:` (page 1731)

Declared In
NSURL.h

query

Returns the query of a URL conforming to RFC 1808.

```
- (NSString *)query
```

Return Value
The query of the URL. If the receiver does not conform to RFC 1808, returns `nil`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSURL.h

relativePath

Returns the path of a URL conforming to RFC 1808, without resolving against the receiver's base URL.

```
- (NSString *)relativePath
```

Return Value
The relative path of the URL without resolving against the base URL. If the receiver is an absolute URL, this method returns the same value as `path` (page 1727). If the receiver does not conform to RFC 1808, returns `nil`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
IdentitySample

Declared In
NSURL.h

relativeString

Returns a string representation of the relative portion of the URL.

```
- (NSString *)relativeString
```

Return Value
A string representation of the relative portion of the URL. If the receiver is an absolute URL this method returns the same value as `absoluteString` (page 1721).

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSURL.h

resourceDataUsingCache:

Returns the receiver's resource data, loading it if necessary. (Deprecated in Mac OS X v10.4.)

```
- (NSData *)resourceDataUsingCache:(BOOL)shouldUseCache
```

Parameters

shouldUseCache

Whether the URL should use cached resource data from an already loaded URL that refers to the same resource. If *YES*, the cache is consulted when loading data. If *NO*, the data is always loaded directly, without consulting the cache.

Return Value
The receiver's resource data.

Discussion
If the receiver has not already loaded its resource data, it will attempt to load it as a blocking operation.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Related Sample Code
ImageClient

Declared In
NSURL.h

resourceSpecifier

Returns the resource specifier of the URL.

```
- (NSString *)resourceSpecifier
```

Return Value
The resource specifier of the URL.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSURL.h

scheme

Returns the scheme of the URL.

```
- (NSString *)scheme
```

Return Value
The scheme of the URL.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
NewsReader

Declared In
NSURL.h

setProperty:forKey:

Changes the specified property of the receiver's resource. (Deprecated in Mac OS X v10.4.)

```
- (BOOL)setProperty:(id)propertyValue
    forKey:(NSString *)propertyKey
```

Parameters

propertyValue

The new value of the property of the receiver's resource.

propertyKey

The key of the desired property.

Return Value

Returns `YES` if the modification was successful, `NO` otherwise.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`NSURL.h`

setResourceData:

Attempts to set the resource data for the receiver. (Deprecated in Mac OS X v10.4.)

```
- (BOOL)setResourceData:(NSData *)data
```

Parameters

data

The data to set for the URL.

Return Value

Returns `YES` if successful, `NO` otherwise.

Discussion

In the case of a file URL, setting the data involves writing *data* to the specified file.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`NSURL.h`

standardizedURL

Returns a new NSURL object with any instances of " . . " or " . " removed from its path.

```
- (NSURL *)standardizedURL
```

Return Value
A new `NSURL` object initialized with a version of the receiver's URL that has had any instances of " . . " or " . " removed from its path.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSURL.h`

URLHandleUsingCache:

Returns a URL handle to service the receiver. (Deprecated in Mac OS X v10.4.)

```
- (NSURLHandle *)URLHandleUsingCache:(BOOL)shouldUseCache
```

Parameters

shouldUseCache

Whether to use a cached URL handle. If *shouldUseCache* is `YES`, the cache is searched for a URL handle that has serviced the receiver or another identical URL. If *shouldUseCache* is `NO`, a newly instantiated handle is returned, even if an equivalent URL has been loaded.

Return Value
A URL handle to service the receiver.

Discussion
Sophisticated clients use the URL handle directly for additional control.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
`cachedHandleForURL:` (page 1795) (NSURLHandle)

Declared In
`NSURL.h`

user

Returns the user portion of a URL conforming to RFC 1808.

```
- (NSString *)user
```

Return Value
The user portion of the URL. If the receiver does not conform to RFC 1808, returns `nil`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSURL.h`

Constants

NSURL Schemes

These schemes are the ones that `NSURL` can parse.

```
extern NSString *NSURLFileScheme;
```

Constants

`NSURLFileScheme`

Identifies a URL that points to a file on a mounted volume.

Available in Mac OS X v10.0 and later.

Declared in `NSURL.h`.

Discussion

For more information, see initWithScheme:host:path: (page 1724).

Declared In

`NSURL.h`

NSURLHandle FTP Property Keys

FTP-specific property keys.

```
extern NSString *NSFTPPropertyUserLoginKey;
extern NSString *NSFTPPropertyUserPasswordKey;
extern NSString *NSFTPPropertyActiveTransferModeKey;
extern NSString *NSFTPPropertyFileOffsetKey;
extern NSString *NSFTPPropertyFTPProxy;
```

Constants

`NSFTPPropertyUserLoginKey`

Key for the user login, returned as an `NSString` object.

The default value for this key is “anonymous”.

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.4.

Declared in `NSURLHandle.h`.

`NSFTPPropertyUserPasswordKey`

Key for the user password, returned as an `NSString` object.

The default value for this key is "`NSURLHandle@apple.com`".

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.4.

Declared in `NSURLHandle.h`.

`NSFTPPropertyActiveTransferModeKey`
Key for retrieving whether in active transfer mode, returned as a boolean wrapped in an `NSNumber` object.

The default value for this key is `NO` (passive mode).

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.4.

Declared in `NSURLHandle.h`.

`NSFTPPropertyFileOffsetKey`
Key for retrieving the file offset, returned as an `NSNumber` object. The default value for this key is zero.

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.4.

Declared in `NSURLHandle.h`.

`NSFTPPropertyFTPProxy`
`NSDictionary` containing proxy information to use in place of proxy identified in `SystemConfiguration.framework`.

To avoid any proxy use, pass an empty dictionary.

Available in Mac OS X v10.3 and later.

Deprecated in Mac OS X v10.4.

Declared in `NSURLHandle.h`.

Discussion
Pass these keys to `NSURLHandle`'s `propertyForKeyIfAvailable:` (page 1803) to request specific data. All keys are optional. The default configuration allows an anonymous, passive-mode, one-off transfer of the specified URL.

Declared In
`NSURL.h`

NSURLHandle HTTP Property Keys

HTTP-specific property keys.

```
extern NSString *NSHTTPPropertyStatusCodeKey;
extern NSString *NSHTTPPropertyStatusReasonKey;
extern NSString *NSHTTPPropertyServerHTTPVersionKey;
extern NSString *NSHTTPPropertyRedirectionHeadersKey;
extern NSString *NSHTTPPropertyErrorPageDataKey;
extern NSString *NSHTTPPropertyHTTPProxy;
```

Constants
`NSHTTPPropertyStatusCodeKey`
Key for the status code, returned as an integer wrapped in an `NSNumber` object.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared in `NSURLHandle.h`.

`NSHTTPPropertyStatusReasonKey`

Key for the remainder of the HTTP status line following the status code, returned as an `NSString` object.

This string usually contains an explanation of the error in English. Because this string is taken straight from the server response, it's not localized.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared in `NSURLHandle.h`.

`NSHTTPPropertyServerHTTPVersionKey`

Key for retrieving the HTTP version as an `NSString` object containing the initial server status line up to the first space.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared in `NSURLHandle.h`.

`NSHTTPPropertyRedirectionHeadersKey`

Key for retrieving the redirection headers as an `NSDictionary` object with each header value keyed to the header name.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared in `NSURLHandle.h`.

`NSHTTPPropertyErrorPageDataKey`

Key for retrieving an error page as an `NSData` object.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared in `NSURLHandle.h`.

`NSHTTPPropertyHTTPProxy`

Key for retrieving the `NSDictionary` object containing proxy information to use in place of proxy identified in `SystemConfiguration.framework`.

To avoid any proxy use, pass an empty dictionary.

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.4.

Declared in `NSURLHandle.h`.

Discussion

Pass these keys to `NSURLHandle`'s `propertyForKeyIfAvailable:` (page 1803) to request specific data.

Declared In

`NSURL.h`

NSURLAuthenticationChallenge Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSURLAuthenticationChallenge.h
Availability	Available in Mac OS X v10.2 with Safari 1.0 installed. Available in Mac OS X v10.2.7 and later.
Companion guide	URL Loading System

Overview

NSURLAuthenticationChallenge encapsulates a challenge from a server requiring authentication from the client.

Tasks

Creating an Authentication Challenge Instance

`- initWithAuthenticationChallenge:sender:` (page 1739)
Returns an initialized NSURLAuthenticationChallenge object copying the properties from *challenge*, and setting the authentication sender to *sender*.

`- initWithProtectionSpace:proposedCredential:previousFailureCount:failureResponse:error:sender:` (page 1739)
Returns an initialized NSURLAuthenticationChallenge object for the specified *space* using the *credential*, or `nil` if there is no proposed credential.

Getting Authentication Challenge Properties

`- error` (page 1738)
Returns the NSError object representing the last authentication failure.

`- failureResponse` (page 1738)
Returns the NSURLResponse object representing the last authentication failure.

- `previousFailureCount` (page 1739)
  Returns the receiver's count of failed authentication attempts.
- `proposedCredential` (page 1740)
  Returns the proposed credential for this challenge.
- `protectionSpace` (page 1740)
  Returns the receiver's protection space.
- `sender` (page 1740)
  Returns the receiver's sender.

Instance Methods

error

Returns the NSError object representing the last authentication failure.

```
- (NSError *)error
```

Discussion
This method returns `nil` if the protocol doesn't use errors to indicate an authentication failure.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- `failureResponse` (page 1738)

Declared In
`NSURLAuthenticationChallenge.h`

failureResponse

Returns the NSURLResponse object representing the last authentication failure.

```
- (NSURLResponse *)failureResponse
```

Discussion
This method will return `nil` if the protocol doesn't use responses to indicate an authentication failure.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- `error` (page 1738)

Declared In
`NSURLAuthenticationChallenge.h`

initWithAuthenticationChallenge:sender:

Returns an initialized NSURLAuthenticationChallenge object copying the properties from *challenge*, and setting the authentication sender to *sender*.

```
- (id)initWithAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
    sender:(id < NSURLAuthenticationChallengeSender >)sender
```

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- initWithProtectionSpace:proposedCredential:previousFailureCount:failureResponse:error:sender: (page 1739)

Declared In
NSURLAuthenticationChallenge.h

initWithProtectionSpace:proposedCredential:previousFailureCount:failureResponse:error:sender:

Returns an initialized NSURLAuthenticationChallenge object for the specified *space* using the *credential*, or `nil` if there is no proposed credential.

```
- (id)initWithProtectionSpace:(NSURLProtectionSpace *)space
    proposedCredential:(NSURLCredential *)credential
    previousFailureCount:(NSInteger)count failureResponse:(NSURLResponse *)response
     error:(NSError *)error sender:(id < NSURLAuthenticationChallengeSender >)sender
```

Discussion
The previous failure count is set to *count*. The *response* should contain the NSURLResponse for the authentication failure, or `nil` if it is not applicable to the challenge. The *error* should contain the NSError for the authentication failure, or `nil` if it is not applicable to the challenge. The object that initiated the authentication challenge is set to *sender*.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- initWithAuthenticationChallenge:sender: (page 1739)

Declared In
NSURLAuthenticationChallenge.h

previousFailureCount

Returns the receiver's count of failed authentication attempts.

```
- (NSInteger)previousFailureCount
```

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLAuthenticationChallenge.h`

proposedCredential

Returns the proposed credential for this challenge.

```
- (NSURLCredential *)proposedCredential
```

Discussion
This method will return `nil` if there is no default credential for this challenge.

If the proposed credential is not `nil` and returns `YES` when sent the message `hasPassword` (page 1769), then the credential is ready to use as-is. If the proposed credential returns `NO` for `hasPassword`, then the credential provides a default user name and the client must prompt the user for a corresponding password.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLAuthenticationChallenge.h`

protectionSpace

Returns the receiver's protection space.

```
- (NSURLProtectionSpace *)protectionSpace
```

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLAuthenticationChallenge.h`

sender

Returns the receiver's sender.

```
- (id < NSURLAuthenticationChallengeSender >)sender
```

Discussion
The sender should be sent a `useCredential:forAuthenticationChallenge:` (page 2126), `continueWithoutCredentialForAuthenticationChallenge:` (page 2126) or `cancelAuthenticationChallenge:` (page 2126) when the client is finished processing the authentication challenge.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLAuthenticationChallenge.h`

NSURLCache Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSURLCache.h
Availability	Available in Mac OS X v10.2 with Safari 1.0 installed. Available in Mac OS X v10.2.7 and later.
Companion guide	URL Loading System
Related sample code	URL CacheInfo

Overview

NSURLCache implements the caching of responses to URL load requests by mapping NSURLRequest objects to NSCachedURLResponse objects. It is a composite of an in-memory and an on-disk cache.

Methods are provided to manipulate the sizes of each of these caches as well as to control the path on disk to use for persistent storage of cache data.

Tasks

Getting and Setting Shared Cache

`+ sharedURLCache` (page 1745)
Returns the shared NSURLCache instance.

`+ setSharedURLCache:` (page 1744)
Sets the shared NSURLCache instance to a specified cache object.

Creating a New Cache Object

`- initWithMemoryCapacity:diskCapacity:diskPath:` (page 1747)
Initializes an NSURLCache object with the specified values.

Getting and Storing Cached Objects

- `cachedResponseForRequest:` (page 1746)
  Returns the cached URL response in the cache for the specified URL request.
- `storeCachedResponse:forRequest:` (page 1750)
  Stores a cached URL response for a specified request

Removing Cached Objects

- `removeAllCachedResponses` (page 1748)
  Clears the receiver's cache, removing all stored cached URL responses.
- `removeCachedResponseForRequest:` (page 1749)
  Removes the cached URL response for a specified URL request.

Getting and Setting On-disk Cache Properties

- `currentDiskUsage` (page 1746)
  Returns the current size of the receiver's on-disk cache, in bytes.
- `diskCapacity` (page 1747)
  Returns the capacity of the receiver's on-disk cache, in bytes.
- `setDiskCapacity:` (page 1749)
  Sets the receiver's on-disk cache capacity

Getting and Setting In-memory Cache Properties

- `currentMemoryUsage` (page 1747)
  Returns the current size of the receiver's in-memory cache, in bytes.
- `memoryCapacity` (page 1748)
  Returns the capacity of the receiver's in-memory cache, in bytes.
- `setMemoryCapacity:` (page 1750)
  Sets the receiver's in-memory cache capacity.

Class Methods

setSharedURLCache:

Sets the shared NSURLCache instance to a specified cache object.

```
+ (void)setSharedURLCache:(NSURLCache *)cache
```

Parameters

cache
The cache object to use as the shared cache object.

Discussion
Applications that have special caching requirements or constraints should use this method to specify an NSURLCache instance with customized cache settings.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
`+ sharedURLCache` (page 1745)

Related Sample Code
URL CacheInfo

Declared In
`NSURLCache.h`

sharedURLCache

Returns the shared NSURLCache instance.

```
+ (NSURLCache *)sharedURLCache
```

Return Value
The shared NSURLCache instance.

Discussion
The disk path is set to: `<user_home_directory>/Library/Caches/<current_process_name>`. The user's home directory is determined by calling `NSHomeDirectory` (page 2199) and the current process name is determined using `[[NSProcessInfo processInfo] processName]`.

Applications that do not have special caching requirements or constraints should find the default shared cache instance acceptable. Applications with more specific needs can create a custom `NSURLCache` object and set it as the shared cache instance using `setSharedURLCache:` (page 1744).

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
`+ setSharedURLCache:` (page 1744)

Related Sample Code
URL CacheInfo

Declared In
`NSURLCache.h`

Instance Methods

cachedResponseForRequest:

Returns the cached URL response in the cache for the specified URL request.

```
- (NSCachedURLResponse *)cachedResponseForRequest:(NSURLRequest *)request
```

Parameters

request

The URL request whose cached response is desired.

Return Value

The cached URL response for *request*, or `nil` if no response has been cached.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

See Also

- `storeCachedResponse:forRequest:` (page 1750)

Related Sample Code

URL CacheInfo

Declared In

`NSURLCache.h`

currentDiskUsage

Returns the current size of the receiver's on-disk cache, in bytes.

```
- (NSUInteger)currentDiskUsage
```

Return Value

The current size of the receiver's on-disk cache, in bytes.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

See Also

- `diskCapacity` (page 1747)
- `setDiskCapacity:` (page 1749)

Related Sample Code

URL CacheInfo

Declared In

`NSURLCache.h`

currentMemoryUsage

Returns the current size of the receiver's in-memory cache, in bytes.

```
- (NSUInteger)currentMemoryUsage
```

Return Value
The current size of the receiver's in-memory cache, in bytes.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- `memoryCapacity` (page 1748)
- `setMemoryCapacity:` (page 1750)

Related Sample Code
URL CacheInfo

Declared In
`NSURLCache.h`

diskCapacity

Returns the capacity of the receiver's on-disk cache, in bytes.

```
- (NSUInteger)diskCapacity
```

Return Value
The capacity of the receiver's on-disk cache, in bytes.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- `currentDiskUsage` (page 1746)
- `setDiskCapacity:` (page 1749)

Declared In
`NSURLCache.h`

initWithMemoryCapacity:diskCapacity:diskPath:

Initializes an NSURLCache object with the specified values.

```
- (id)initWithMemoryCapacity:(NSUInteger)memoryCapacity
    diskCapacity:(NSUInteger)diskCapacity diskPath:(NSString *)path
```

Parameters
memoryCapacity
The memory capacity of the cache, in bytes.

diskCapacity
The disk capacity of the cache, in bytes.

path
The location at which to store the on-disk cache.

Return Value
The initialized NSURLCache object.

Discussion
The returned NSURLCache is backed by disk, so developers can be more liberal with space when choosing the capacity for this kind of cache. A disk cache measured in the tens of megabytes should be acceptable in most cases.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
`+ sharedURLCache` (page 1745)

Related Sample Code
URL CacheInfo

Declared In
`NSURLCache.h`

memoryCapacity

Returns the capacity of the receiver's in-memory cache, in bytes.

```
- (NSUInteger)memoryCapacity
```

Return Value
The capacity of the receiver's in-memory cache, in bytes.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
`- currentMemoryUsage` (page 1747)
`- setMemoryCapacity:` (page 1750)

Declared In
`NSURLCache.h`

removeAllCachedResponses

Clears the receiver's cache, removing all stored cached URL responses.

```
- (void)removeAllCachedResponses
```

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- `removeCachedResponseForRequest:` (page 1749)

Related Sample Code
URL CacheInfo

Declared In
`NSURLCache.h`

removeCachedResponseForRequest:

Removes the cached URL response for a specified URL request.

```
- (void)removeCachedResponseForRequest:(NSURLRequest *)request
```

Parameters

request
The URL request whose cached URL response should be removed. If there is no corresponding cached URL response, no action is taken.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- `removeAllCachedResponses` (page 1748)

Declared In
`NSURLCache.h`

setDiskCapacity:

Sets the receiver's on-disk cache capacity

```
- (void)setDiskCapacity:(NSUInteger)diskCapacity
```

Parameters

diskCapacity
The new on-disk cache capacity, in bytes. The on-disk cache will truncate its contents to *diskCapacity*, if necessary.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- `currentDiskUsage` (page 1746)
- `diskCapacity` (page 1747)

Related Sample Code
URL CacheInfo

Declared In
NSURLCache.h

setMemoryCapacity:

Sets the receiver's in-memory cache capacity.

```
- (void)setMemoryCapacity:(NSUInteger)memoryCapacity
```

Parameters

memoryCapacity
The new in-memory cache capacity, in bytes. The in-memory cache will truncate its contents to *memoryCapacity*, if necessary.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- currentMemoryUsage (page 1747)
- memoryCapacity (page 1748)

Related Sample Code
URL CacheInfo

Declared In
NSURLCache.h

storeCachedResponse:forRequest:

Stores a cached URL response for a specified request

```
- (void)storeCachedResponse:(NSCachedURLResponse *)cachedResponse
    forRequest:(NSURLRequest *)request
```

Parameters

cachedResponse
The cached URL response to store.

request
The request for which the cached URL response is being stored.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- cachedResponseForRequest: (page 1746)

Declared In
`NSURLCache.h`

NSURLConnection Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSURLConnection.h
Availability	Available in Mac OS X v10.2 with Safari 1.0 installed. Available in Mac OS X v10.2.7 and later.
Companion guide	URL Loading System
Related sample code	URL CacheInfo

Overview

An NSURLConnection object provides support to perform the loading of a URL request. The interface for NSURLConnection is sparse, providing only the controls to start and cancel asynchronous loads of a URL request.

NSURLConnection's delegate methods allow an object to receive informational callbacks about the asynchronous load of a URL request. Other delegate methods provide facilities that allow the delegate to customize the process of performing an asynchronous URL load.

Note that these delegate methods will be called on the thread that started the asynchronous load operation for the associated NSURLConnection object.

The following contract governs the delegate methods defined in this interface:

- Zero or more `connection:willSendRequest:redirectResponse:` (page 1764) messages will be sent to the delegate before any further messages are sent if it is determined that the download must redirect to a new location. The delegate can allow the redirect, modify the destination or deny the redirect.
- Zero or more `connection:didReceiveAuthenticationChallenge:` (page 1761) messages will be sent to the delegate if it is necessary to authenticate in order to download the request and NSURLConnection does not already have authenticated credentials.
- Zero or more `connection:didCancelAuthenticationChallenge:` (page 1761) messages will be sent to the delegate if the connection cancels the authentication challenge due to the protocol implementation encountering an error.

- Zero or more `connection:didReceiveResponse:` (page 1763) messages will be sent to the delegate before receiving a `connection:didReceiveData:` (page 1762) message. The only case where `connection:didReceiveResponse:` is not sent to a delegate is when the protocol implementation encounters an error before a response could be created.
- Zero or more `connection:didReceiveData:` (page 1762) messages will be sent before any of the following messages are sent to the delegate: `connection:willCacheResponse:` (page 1764), `connectionDidFinishLoading:` (page 1765), `connection:didFailWithError:` (page 1761).
- Zero or one `connection:willCacheResponse:` (page 1764) messages will be sent to the delegate after `connection:didReceiveData:` (page 1762) is sent but before a `connectionDidFinishLoading:` (page 1765) message is sent.
- Unless a NSURLConnection receives a `cancel` (page 1758) message, the delegate will receive one and only one of `connectionDidFinishLoading:` (page 1765), or `connection:didFailWithError:` (page 1761) message, but never both. In addition, once either of messages are sent, the delegate will receive no further messages for the given NSURLConnection.

NSURLConnection also has a convenience class method, `sendSynchronousRequest:returningResponse:error:` (page 1757), to load a URL request synchronously.

`NSHTTPURLResponse` is a subclass of `NSURLResponse` that provides methods for accessing information specific to HTTP protocol responses. An `NSHTTPURLResponse` object represents a response to an HTTP URL load request.

Tasks

Preflighting a Request

`+ canHandleRequest:` (page 1756)
Returns whether a request can be handled based on a "preflight" evaluation.

Loading Data Synchronously

`+ sendSynchronousRequest:returningResponse:error:` (page 1757)
Performs a synchronous load of the specified URL request.

Loading Data Asynchronously

`+ connectionWithRequest:delegate:` (page 1756)
Creates and returns an initialized URL connection and begins to load the data for the URL request.

`- initWithRequest:delegate:` (page 1758)
Returns an initialized URL connection and begins to load the data for the URL request.

`- initWithRequest:delegate:startImmediately:` (page 1759)
Returns an initialized URL connection and begins to load the data for the URL request, if specified.

`- start` (page 1760)
Causes the receiver to begin loading data, if it has not already.

Stopping a Connection

- `cancel` (page 1758)
  Cancels an asynchronous load of a request.

Runloop Scheduling

- `scheduleInRunLoop:forMode:` (page 1759)
  Determines the runloop and mode that the receiver uses to send delegate messages to the receiver.
- `unscheduleFromRunLoop:forMode:` (page 1760)
  Causes the receiver to stop sending delegate messages using the specified runloop and mode.

Connection Authentication

- `connection:didCancelAuthenticationChallenge:` (page 1761) *delegate method*
  Sent when a connection cancels an authentication challenge.
- `connection:didReceiveAuthenticationChallenge:` (page 1761) *delegate method*
  Sent when a connection must authenticate a challenge in order to download its request.

Connection Data and Responses

- `connection:willCacheResponse:` (page 1764) *delegate method*
  Sent before the connection stores a cached response in the cache, to give the delegate an opportunity to alter it.
- `connection:didReceiveResponse:` (page 1763) *delegate method*
  Sent when the connection has received sufficient data to construct the URL response for its request.
- `connection:didReceiveData:` (page 1762) *delegate method*
  Sent as a connection loads data incrementally.
- `connection:willSendRequest:redirectResponse:` (page 1764) *delegate method*
  Sent when the connection determines that it must change URLs in order to continue loading a request.

Connection Completion

- `connection:didFailWithError:` (page 1761) *delegate method*
  Sent when a connection fails to load its request successfully.
- `connectionDidFinishLoading:` (page 1765) *delegate method*
  Sent when a connection has finished loading successfully.

Class Methods

canHandleRequest:

Returns whether a request can be handled based on a "preflight" evaluation.

```
+ (BOOL)canHandleRequest:(NSURLRequest *)request
```

Parameters

request

The request to evaluate.

Return Value

`YES` if a “preflight” operation determines that a connection with *request* can be created and the associated I/O can be started, `NO` otherwise.

Discussion

The result of this method is valid as long as no NSURLProtocol classes are registered or unregistered, and the specified *request* remains unchanged. Applications should be prepared to handle failures even if they have performed request preflighting by calling this method.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

See Also

`+ registerClass:` (page 1819)

`+ unregisterClass:` (page 1821)

Declared In

`NSURLConnection.h`

connectionWithRequest:delegate:

Creates and returns an initialized URL connection and begins to load the data for the URL request.

```
+ (NSURLConnection *)connectionWithRequest:(NSURLRequest *)request
    delegate:(id)delegate
```

Parameters

request

The URL request to load. The *request* object is deep-copied as part of the initialization process. Changes made to *request* after this method returns do not affect the request that is used for the loading process.

delegate

The delegate object for the connection. The delegate will receive delegate messages as the load progresses. Messages to the delegate will be sent on the thread that calls this method. For the connection to work correctly the calling thread’s run loop must be operating in the default run loop mode.]

Return Value

The URL connection for the URL request. Returns `nil` if a connection can't be created.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
`- initWithRequest:delegate:` (page 1758)

Declared In
`NSURLConnection.h`

sendSynchronousRequest:returningResponse:error:

Performs a synchronous load of the specified URL request.

```
+ (NSData *)sendSynchronousRequest:(NSURLRequest *)request
    returningResponse:(NSURLResponse **)response error:(NSError **)error
```

Parameters

request
The URL request to load. The *request* object is deep-copied as part of the initialization process. Changes made to *request* after this method returns do not affect the request that is used for the loading process.

response
Out parameter for the URL response returned by the server.

error
Out parameter used if an error occurs while processing the request. May be `NULL`.

Return Value
The downloaded data for the URL request. Returns `nil` if a connection could not be created or if the download fails.

Discussion
A synchronous load is built on top of the asynchronous loading code made available by the class. The calling thread is blocked while the asynchronous loading system performs the URL load on a thread spawned specifically for this load request. No special threading or run loop configuration is necessary in the calling thread in order to perform a synchronous load.

If authentication is required in order to download the request, the required credentials must be specified as part of the URL. If authentication fails, or credentials are missing, the connection will attempt to continue without credentials.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLConnection.h`

Instance Methods

cancel

Cancels an asynchronous load of a request.

```
- (void)cancel
```

Discussion
Once this method is called, the receiver's delegate will no longer receive any messages for this NSURLConnection.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
`+ connectionWithRequest:delegate:` (page 1756)
`- initWithRequest:delegate:` (page 1758)

Declared In
`NSURLConnection.h`

initWithRequest:delegate:

Returns an initialized URL connection and begins to load the data for the URL request.

```
- (id)initWithRequest:(NSURLRequest *)request delegate:(id)delegate
```

Parameters

request

The URL request to load. The *request* object is deep-copied as part of the initialization process. Changes made to *request* after this method returns do not affect the request that is used for the loading process.

delegate

The delegate object for the connection. The delegate will receive delegate messages as the load progresses. Messages to the delegate will be sent on the thread that calls this method. By default, for the connection to work correctly the calling thread's run loop must be operating in the default run loop mode. See `scheduleInRunLoop:forMode:` (page 1759) to change the runloop and mode.

Return Value
The URL connection for the URL request. Returns `nil` if a connection can't be initialized.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
`+ connectionWithRequest:delegate:` (page 1756)
`- initWithRequest:delegate:startImmediately:` (page 1759)

Declared In
NSURLConnection.h

initWithRequest:delegate:startImmediately:

Returns an initialized URL connection and begins to load the data for the URL request, if specified.

```
- (id)initWithRequest:(NSURLRequest *)request delegate:(id)delegate
    startImmediately:(BOOL)startImmediately
```

Parameters

request

The URL request to load. The *request* object is deep-copied as part of the initialization process. Changes made to *request* after this method returns do not affect the request that is used for the loading process.

delegate

The delegate object for the connection. The delegate will receive delegate messages as the load progresses. Messages to the delegate will be sent on the thread that calls this method. By default, for the connection to work correctly the calling thread's run loop must be operating in the default run loop mode. See `scheduleInRunLoop:forMode:` (page 1759) to change the runloop and mode.]

startImmediately

`YES` if the connection should being loading data immediately, otherwise `NO`.

Return Value
The URL connection for the URL request. Returns `nil` if a connection can't be initialized.

Availability
Available in Mac OS X v10.5 and later.

Related Sample Code
URL CacheInfo

Declared In
NSURLConnection.h

scheduleInRunLoop:forMode:

Determines the runloop and mode that the receiver uses to send delegate messages to the receiver.

```
- (void)scheduleInRunLoop:(NSRunLoop *)aRunLoop forMode:(NSString *)mode
```

Parameters

aRunLoop

The NSRunloop instance to use for delegate messages.

mode

The mode in which to supply delegate messages.

Discussion
At creation, a connection is scheduled on the current thread (the one where the creation takes place) in the default mode. That can be changed to add or remove runloop + mode pairs using the following methods. It is permissible to be scheduled on multiple run loops and modes, or on the same run loop in multiple modes, so scheduling in one place does not cause unscheduling in another.

You may call these methods after the connection has started. However, if the connection is scheduled on multiple threads or if you are not calling these methods from the thread where the connection is scheduled, there is a race between these methods and the delivery of delegate methods on the other threads. The caller must either be prepared for additional delegation messages on the other threads, or must halt the run loops on the other threads before calling these methods to guarantee that no further callbacks will occur.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSURLConnection.h`

start

Causes the receiver to begin loading data, if it has not already.

```
- (void)start
```

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSURLConnection.h`

unscheduleFromRunLoop:forMode:

Causes the receiver to stop sending delegate messages using the specified runloop and mode.

```
- (void)unscheduleFromRunLoop:(NSRunLoop *)aRunLoop forMode:(NSString *)mode
```

Parameters

aRunLoop
The runloop instance to unschedule.

mode
The mode to unschedule.

Discussion
At creation, a connection is scheduled on the current thread (the one where the creation takes place) in the default mode. That can be changed to add or remove runloop + mode pairs using the following methods. It is permissible to be scheduled on multiple run loops and modes, or on the same run loop in multiple modes, so scheduling in one place does not cause unscheduling in another.

You may call these methods after the connection has started. However, if the connection is scheduled on multiple threads or if you are not calling these methods from the thread where the connection is scheduled, there is a race between these methods and the delivery of delegate methods on the other threads. The caller must either be prepared for additional delegation messages on the other threads, or must halt the run loops on the other threads before calling these methods to guarantee that no further callbacks will occur.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSURLConnection.h`

Delegate Methods

connection:didCancelAuthenticationChallenge:

Sent when a connection cancels an authentication challenge.

```
- (void)connection:(NSURLConnection *)connection
    didCancelAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
```

Parameters

connection

The connection sending the message.

challenge

The challenge that was canceled.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

NSURLConnection.h

connection:didFailWithError:

Sent when a connection fails to load its request successfully.

```
- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError *)error
```

Parameters

connection

The connection sending the message.

error

An error object containing details of why the connection failed to load the request successfully.

Discussion

Once the delegate receives this message, it will receive no further messages for *connection*.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

NSURLConnection.h

connection:didReceiveAuthenticationChallenge:

Sent when a connection must authenticate a challenge in order to download its request.

```
- (void)connection:(NSURLConnection *)connection
    didReceiveAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
```

Parameters

connection

The connection sending the message.

challenge

The challenge that *connection* must authenticate in order to download its request.

Discussion

This method gives the delegate the opportunity to determine the course of action taken for the challenge: provide credentials, continue without providing credentials, or cancel the authentication challenge and the download.

The delegate can determine the number of previous authentication challenges by sending the message `previousFailureCount` (page 1739) to *challenge*.

If the previous failure count is 0 and the value returned by `proposedCredential` (page 1740) is `nil`, the delegate can create a new NSURLCredential object, providing a user name and password, and send a `useCredential:forAuthenticationChallenge:` (page 2126) message to `[challenge sender]`, passing the credential and *challenge* as parameters. If `proposedCredential` is not `nil`, the value is a credential from the URL or the shared credential storage that can be provided to the user as feedback.

The delegate may decide to abandon further attempts at authentication at any time by sending `[challenge sender]` a `continueWithoutCredentialForAuthenticationChallenge:` (page 2126) or a `cancelAuthenticationChallenge:` (page 2126) message. The specific action will be implementation dependent.

If the delegate implements this method, the download will suspend until `[challenge sender]` is sent one of the following messages: `useCredential:forAuthenticationChallenge:` (page 2126), `continueWithoutCredentialForAuthenticationChallenge:` (page 2126) or `cancelAuthenticationChallenge:` (page 2126).

If the delegate does not implement this method the default implementation is used. If a valid credential for the request is provided as part of the URL, or is available from the NSURLCredentialStorage the `[challenge sender]` is sent a `useCredential:forAuthenticationChallenge:` (page 2126) with the credential. If the challenge has no credential or the credentials fail to authorize access, then `continueWithoutCredentialForAuthenticationChallenge:` (page 2126) is sent to `[challenge sender]` instead.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

See Also

- `cancelAuthenticationChallenge:` (page 2126)
- `continueWithoutCredentialForAuthenticationChallenge:` (page 2126)
- `useCredential:forAuthenticationChallenge:` (page 2126)

Declared In

`NSURLConnection.h`

connection:didReceiveData:

Sent as a connection loads data incrementally.

```
- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data
```

Parameters

connection

The connection sending the message.

data

The newly available data. The delegate should concatenate the contents of each *data* object delivered to build up the complete data for a URL load.

Discussion

This method provides the only way for an asynchronous delegate to retrieve the loaded data. It is the responsibility of the delegate to retain or copy this data as it is delivered.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

`NSURLConnection.h`

connection:didReceiveResponse:

Sent when the connection has received sufficient data to construct the URL response for its request.

```
- (void)connection:(NSURLConnection *)connection didReceiveResponse:(NSURLResponse
    *)response
```

Parameters

connection

The connection sending the message.

response

The URL response for the connection's request. This object is immutable and will not be modified by the URL loading system once it is presented to the delegate.

Discussion

In rare cases, for example in the case of an HTTP load where the content type of the load data is `multipart/x-mixed-replace`, the delegate will receive more than one `connection:didReceiveResponse:` message. In the event this occurs, delegates should discard all data previously delivered by `connection:didReceiveData:`, and should be prepared to handle the, potentially different, MIME type reported by the newly reported URL response.

The only case where this message is not sent to the delegate is when the protocol implementation encounters an error before a response could be created.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

`NSURLConnection.h`

connection:willCacheResponse:

Sent before the connection stores a cached response in the cache, to give the delegate an opportunity to alter it.

```
- (NSCachedURLResponse *)connection:(NSURLConnection *)connection
    willCacheResponse:(NSCachedURLResponse *)cachedResponse
```

Parameters

connection

The connection sending the message.

cachedResponse

The proposed cached response to store in the cache.

Return Value

The actual cached response to store in the cache. The delegate may return *cachedResponse* unmodified, return a modified cached response, or return `nil` if no cached response should be stored for the connection.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

`NSURLConnection.h`

connection:willSendRequest:redirectResponse:

Sent when the connection determines that it must change URLs in order to continue loading a request.

```
- (NSURLRequest *)connection:(NSURLConnection *)connection
    willSendRequest:(NSURLRequest *)request redirectResponse:(NSURLResponse
    *)redirectResponse
```

Parameters

connection

The connection sending the message.

request

The proposed redirected request. The delegate should inspect the redirected request to verify that it meets its needs, and create a copy with new attributes to return to the connection if necessary.

redirectResponse

The URL response that caused the redirect. May be *nil* in cases where this method is not being sent as a result of involving the delegate in redirect processing.

Return Value

The actual URL request to use in light of the redirection response. The delegate may copy and modify *request* as necessary to change its attributes, return *request* unmodified, or return `nil`.

Discussion

If the delegate wishes to cancel the redirect, it should call the *connection* object's `cancel` method. Alternatively, the delegate method can return `nil` to cancel the redirect, and the connection will continue to process. This has special relevance in the case where *redirectResponse* is not `nil`. In this case, any data that is loaded for the connection will be sent to the delegate, and the delegate will receive a `connectionDidFinishLoading` or `connection:didFailLoadingWithError:` message, as appropriate.

Special Considerations

The delegate can receive this message as a result of transforming a request's URL to its canonical form, or for protocol-specific reasons, such as an HTTP redirect. The delegate implementation should be prepared to receive this message multiple times.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In

`NSURLConnection.h`

connectionDidFinishLoading:

Sent when a connection has finished loading successfully.

```
- (void)connectionDidFinishLoading:(NSURLConnection *)connection
```

Parameters

connection

The connection sending the message.

Discussion

The delegate will receive no further messages for *connection*.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In

`NSURLConnection.h`

NSURLCredential Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSURLCredential.h
Availability	Available in Mac OS X v10.2 with Safari 1.0 installed. Available in Mac OS X v10.2.7 and later.
Companion guide	URL Loading System

Overview

`NSURLCredential` is an immutable object representing an authentication credential consisting of the user name, a password and the type of persistent storage to use, if any.

Adopted Protocols

NSCopying
`copyWithZone:` (page 2042)

Tasks

Creating a Credential

`+ credentialWithUser:password:persistence:` (page 1768)
Creates and returns an `NSURLCredential` object with a given user name and password using a given persistence setting.

`- initWithUser:password:persistence:` (page 1769)
Returns an `NSURLCredential` object initialized with a given user name and password using a given persistence setting.

Getting Credential Properties

- `hasPassword` (page 1769)
  Returns a Boolean value that indicates whether the receiver has a password.
- `password` (page 1770)
  Returns the receiver's password.
- `persistence` (page 1770)
  Returns the receiver's persistence setting.
- `user` (page 1770)
  Returns the receiver's user name.

Class Methods

credentialWithUser:password:persistence:

Creates and returns an `NSURLCredential` object with a given user name and password using a given persistence setting.

```
+ (NSURLCredential *)credentialWithUser:(NSString *)user password:(NSString
    *)password persistence:(NSURLCredentialPersistence)persistence
```

Parameters

user
The user for the credential.

password
The password for *user*.

persistence
The persistence setting for the credential.

Return Value
An `NSURLCredential` object with user name *user*, password *password*, and using persistence setting *persistence*.

Discussion
If *persistence* is `NSURLCredentialPersistencePermanent` the credential is stored in the keychain.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
`- initWithUser:password:persistence:` (page 1769)

Declared In
`NSURLCredential.h`

Instance Methods

hasPassword

Returns a Boolean value that indicates whether the receiver has a password.

```
- (BOOL)hasPassword
```

Return Value
`YES` if the receiver has a password, `NO` otherwise.

Discussion
This method does not attempt to retrieve the password.

If this credential's password is stored in the user's keychain, `password` (page 1770) may return `NO` even if this method returns `YES`, since getting the password may fail, or the user may refuse access.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLCredential.h`

initWithUser:password:persistence:

Returns an `NSURLCredential` object initialized with a given user name and password using a given persistence setting.

```
- (id)initWithUser:(NSString *)user password:(NSString *)password
    persistence:(NSURLCredentialPersistence)persistence
```

Parameters

user
The user for the credential.

password
The password for *user*.

persistence
The persistence setting for the credential.

Return Value
An `NSURLCredential` object initialized with user name *user*, password *password*, and using persistence setting *persistence*.

Discussion
If *persistence* is `NSURLCredentialPersistencePermanent` the credential is stored in the keychain.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
`+ credentialWithUser:password:persistence:` (page 1768)

Declared In
`NSURLCredential.h`

password

Returns the receiver's password.

```
- (NSString *)password
```

Return Value
The receiver's password.

Discussion
If the password is stored in the user's keychain, this method may result in prompting the user for access.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
`- hasPassword` (page 1769)

Declared In
`NSURLCredential.h`

persistence

Returns the receiver's persistence setting.

```
- (NSURLCredentialPersistence)persistence
```

Return Value
The receiver's persistence setting.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLCredential.h`

user

Returns the receiver's user name.

```
- (NSString *)user
```

Return Value
The receiver's user name.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLCredential.h`

Constants

NSURLCredentialPersistence

These constants specify how long the credential will be kept.

```
typedef enum {
    NSURLCredentialPersistenceNone,
    NSURLCredentialPersistenceForSession,
    NSURLCredentialPersistencePermanent
} NSURLCredentialPersistence;
```

Constants

`NSURLCredentialPersistenceNone`
Credential won't be stored.
Available in Mac OS X v10.2 and later.
Declared in `NSURLCredential.h`.

`NSURLCredentialPersistenceForSession`
Credential will be stored only for this session.
Available in Mac OS X v10.2 and later.
Declared in `NSURLCredential.h`.

`NSURLCredentialPersistencePermanent`
Credential will be stored in the user's keychain and shared with other applications.
Available in Mac OS X v10.2 and later.
Declared in `NSURLCredential.h`.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLCredential.h`

NSURLCredentialStorage Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSURLCredentialStorage.h
Availability	Available in Mac OS X v10.2 with Safari 1.0 installed. Available in Mac OS X v10.2.7 and later.
Companion guide	URL Loading System

Overview

NSURLCredentialStorage implements a singleton (shared object) that manages the credential storage.

Tasks

Getting the Credential Storage

`+ sharedCredentialStorage` (page 1774)
Returns the shared URL credential storage object.

Getting and Setting Default Credentials

`- defaultCredentialForProtectionSpace:` (page 1775)
Returns the default credential for the specified *protectionSpace*.

`- setDefaultCredential:forProtectionSpace:` (page 1777)
Sets the default credential for a specified protection space.

Adding and Removing Credentials

`- removeCredential:forProtectionSpace:` (page 1776)
Removes a specified credential from the credential storage for the specified protection space.

- `setCredential:forProtectionSpace:` (page 1776)
  Adds *credential* to the credential storage for the specified *protectionSpace*.

Retrieving Credentials

- `allCredentials` (page 1774)
  Returns a dictionary containing the credentials for all available protection spaces.
- `credentialsForProtectionSpace:` (page 1775)
  Returns a dictionary containing the credentials for the specified protection space.

Class Methods

sharedCredentialStorage

Returns the shared URL credential storage object.

```
+ (NSURLCredentialStorage *)sharedCredentialStorage
```

Return Value
The shared NSURLCredentialStorage object.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLCredentialStorage.h`

Instance Methods

allCredentials

Returns a dictionary containing the credentials for all available protection spaces.

```
- (NSDictionary *)allCredentials
```

Return Value
A dictionary containing the credentials for all available protection spaces. The dictionary has keys corresponding to the `NSURLProtectionSpace` objects. The values for the `NSURLProtectionSpace` keys consist of dictionaries where the keys are user name strings, and the value is the corresponding `NSURLCredential` object.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- `credentialsForProtectionSpace:` (page 1775)

Declared In
`NSURLCredentialStorage.h`

credentialsForProtectionSpace:

Returns a dictionary containing the credentials for the specified protection space.

```
- (NSDictionary *)credentialsForProtectionSpace:(NSURLProtectionSpace
    *)protectionSpace
```

Parameters

protectionSpace

The protection space whose credentials you want to retrieve.

Return Value
A dictionary containing the credentials for *protectionSpace*. The dictionary's keys are user name strings, and the value is the corresponding `NSURLCredential`.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- `allCredentials` (page 1774)

Declared In
`NSURLCredentialStorage.h`

defaultCredentialForProtectionSpace:

Returns the default credential for the specified *protectionSpace*.

```
- (NSURLCredential *)defaultCredentialForProtectionSpace:(NSURLProtectionSpace
    *)protectionSpace
```

Parameters

protectionSpace

The URL protection space of interest.

Return Value
The default credential for *protectionSpace* or `nil` if no default has been set.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- `setDefaultCredential:forProtectionSpace:` (page 1777)

Declared In
NSURLCredentialStorage.h

removeCredential:forProtectionSpace:

Removes a specified credential from the credential storage for the specified protection space.

```
- (void)removeCredential:(NSURLCredential *)credential
    forProtectionSpace:(NSURLProtectionSpace *)protectionSpace
```

Parameters

credential
The credential to remove.

protectionSpace
The protection space from which to remove the credential.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- setCredential:forProtectionSpace: (page 1776)

Declared In
NSURLCredentialStorage.h

setCredential:forProtectionSpace:

Adds *credential* to the credential storage for the specified *protectionSpace*.

```
- (void)setCredential:(NSURLCredential *)credential
    forProtectionSpace:(NSURLProtectionSpace *)protectionSpace
```

Parameters

credential
The credential to add. If a credential with the same user name already exists in *protectionSpace*, then *credential* replaces the existing object.

protectionSpace
The protection space to which to add the credential.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- removeCredential:forProtectionSpace: (page 1776)

Declared In
NSURLCredentialStorage.h

setDefaultCredential:forProtectionSpace:

Sets the default credential for a specified protection space.

```
- (void)setDefaultCredential:(NSURLCredential *)credential
    forProtectionSpace:(NSURLProtectionSpace *)protectionSpace
```

Parameters

credential

The URL credential to set as the default for *protectionSpace*. If the receiver does not contain *credential* in the specified *protectionSpace* it will be added.

protectionSpace

The protection space whose default credential is being set.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- defaultCredentialForProtectionSpace: (page 1775)

Declared In
NSURLCredentialStorage.h

Notifications

NSURLCredentialStorageChangedNotification

This notification is posted when the set of stored credentials changes.

The notification object is the NSURLCredentialStorage instance. This notification does not contain a userInfo dictionary.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSURLCredentialStorage.h

NSURLDownload Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSURLDownload.h
Availability	Available in Mac OS X v10.2 with Safari 1.0 installed. Available in Mac OS X v10.2.7 and later.
Companion guide	URL Loading System

Overview

NSURLDownload downloads a request asynchronously and saves the data to a file. The interface for NSURLDownload is sparse, providing methods to initialize a download, set the destination path and cancel loading the request.

NSURLDownload's delegate methods allow an object to receive informational callbacks about the asynchronous load of the URL request. Other delegate methods provide facilities that allow the delegate to customize the process of performing an asynchronous URL load.

Note that these delegate methods will be called on the thread that started the asynchronous load operation for the associated NSURLDownload object.

- A `downloadDidBegin:` (page 1791) message will be sent to the delegate immediately upon starting the download.
- Zero or more `download:willSendRequest:redirectResponse:` (page 1791) messages will be sent to the delegate before any further messages are sent if it is determined that the download must redirect to a new location. The delegate can allow the redirect, modify the destination or deny the redirect.
- Zero or more `download:didReceiveAuthenticationChallenge:` (page 1788) messages will be sent to the delegate if it is necessary to authenticate in order to download the request and NSURLDownload does not already have authenticated credentials.
- Zero or more `download:didCancelAuthenticationChallenge:` (page 1786) messages will be sent to the delegate if NSURLDownload cancels the authentication challenge due to encountering a protocol implementation error.
- Zero or more `download:didReceiveResponse:` (page 1789) messages will be sent to the delegate before receiving a `download:didReceiveDataOfLength:` (page 1789) message. The only case where `download:didReceiveResponse:` is not sent to a delegate is when the protocol implementation encounters an error before a response could be created.

- Zero or more `download:didReceiveDataOfLength:` (page 1789) messages will be sent before `downloadDidFinish:` (page 1792) or `download:didFailWithError:` (page 1787) is sent to the delegate.
- Zero or one `download:decideDestinationWithSuggestedFilename:` (page 1786) will be sent to the delegate when sufficient information has been received to determine the suggested filename for the downloaded file. The delegate will not receive this message if `setDestination:allowOverwrite:` (page 1785) has already been sent to the NSURLDownload instance.
- A `download:didCreateDestination:` (page 1787) message will be sent to the delegate when the NSURLDownload instance creates the file on disk.
- If NSURLDownload determines that the downloaded file is in a format that it is able to decode (MacBinary, Binhex or gzip), the delegate will receive a `download:shouldDecodeSourceDataOfMIMEType:` (page 1790). The delegate should return `YES` to decode the data, `NO` otherwise.
- Unless an NSURLDownload instance receives a `cancel` (page 1782) message, the delegate will receive one and only one `downloadDidFinish:` (page 1792) or `download:didFailWithError:` (page 1787) message, but never both. In addition, once either of messages are sent, the delegate will receive no further messages for the given NSURLDownload.

Tasks

Creating a Download Instance

`- initWithRequest:delegate:` (page 1783)
Returns an initialized URL download for a URL request and begins to download the data for the request.

Resuming Partial Downloads

`+ canResumeDownloadDecodedWithEncodingMIMEType:` (page 1782)
Returns whether a URL download object can resume a download that was decoded with the specified MIME type.

`- initWithResumeData:delegate:path:` (page 1783)
Returns an initialized NSURLDownload object that will resume downloading the specified data to the specified file and begins the download.

`- resumeData` (page 1784)
Returns the resume data for a download that is not yet complete.

`- setDeletesFileUponFailure:` (page 1785)
Specifies whether the receiver deletes the partially downloaded file when a download stops prematurely.

`- deletesFileUponFailure` (page 1783)
Returns whether the receiver deletes partially downloaded files when a download stops prematurely.

Canceling a Download

- `cancel` (page 1782)
  Cancels the receiver's download and deletes the downloaded file.

Getting Download Properties

- `request` (page 1784)
  Returns the request that initiated the receiver's download.

Setting the Destination Path

- `setDestination:allowOverwrite:` (page 1785)
  Sets the destination path of the downloaded file.

Download progress

- `download:decideDestinationWithSuggestedFilename:` (page 1786) *delegate method*
  The delegate receives this message when `download` has determined a suggested filename for the downloaded file.
- `download:didCancelAuthenticationChallenge:` (page 1786) *delegate method*
  Sent if an authentication challenge is canceled due to the protocol implementation encountering an error.
- `download:didCreateDestination:` (page 1787) *delegate method*
  Sent when the destination file is created.
- `download:didFailWithError:` (page 1787) *delegate method*
  Sent if the download fails or if an I/O error occurs when the file is written to disk.
- `download:didReceiveAuthenticationChallenge:` (page 1788) *delegate method*
  Sent when the URL download must authenticate a challenge in order to download the request.
- `download:didReceiveDataOfLength:` (page 1789) *delegate method*
  Sent as a download object receives data incrementally.
- `download:didReceiveResponse:` (page 1789) *delegate method*
  Sent when a download object has received sufficient load data to construct the NSURLResponse object for the download.
- `download:shouldDecodeSourceDataOfMIMEType:` (page 1790) *delegate method*
  Sent when a download object determines that the downloaded file is encoded to inquire whether the file should be automatically decoded.
- `download:willSendRequest:redirectResponse:` (page 1791) *delegate method*
  Sent when the download object determines that it must change URLs in order to continue loading a request.
- `downloadDidBegin:` (page 1791) *delegate method*
  Sent immediately after a download object begins a download.

- `downloadDidFinish:` (page 1792) *delegate method*
  Sent when a download object has completed downloading successfully and has written its results to disk.
- `download:willResumeWithResponse:fromByte:` (page 1790) *delegate method*
  Sent when a download object has received a response from the server after attempting to resume a download.

Class Methods

canResumeDownloadDecodedWithEncodingMIMEType:

Returns whether a URL download object can resume a download that was decoded with the specified MIME type.

```
+ (BOOL)canResumeDownloadDecodedWithEncodingMIMEType:(NSString *)MIMEType
```

Parameters

MIMEType
The MIME type the caller wants to know about.

Return Value
`YES` if the URL download object can resume a download that was decoded with the specified MIME type, `NO` otherwise.

Discussion
NSURLDownload cannot resume a download that was partially decoded in the gzip format.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSURLDownload.h`

Instance Methods

cancel

Cancels the receiver's download and deletes the downloaded file.

```
- (void)cancel
```

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLDownload.h`

deletesFileUponFailure

Returns whether the receiver deletes partially downloaded files when a download stops prematurely.

```
- (BOOL)deletesFileUponFailure
```

Return Value
`YES` if partially downloaded files should be deleted when a download stops prematurely, `NO` otherwise. The default is `YES`.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setDeletesFileUponFailure:` (page 1785)

Declared In
`NSURLDownload.h`

initWithRequest:delegate:

Returns an initialized URL download for a URL request and begins to download the data for the request.

```
- (id)initWithRequest:(NSURLRequest *)request delegate:(id)delegate
```

Parameters

request
The URL request to download. The *request* object is deep-copied as part of the initialization process. Changes made to *request* after this method returns do not affect the request that is used for the loading process.

delegate
The delegate for the download. This object will receive delegate messages as the download progresses. Delegate messages will be sent on the thread which calls this method. For the download to work correctly the calling thread's run loop must be operating in the default run loop mode.

Return Value
An initialized NSURLDownload object for *request*.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLDownload.h`

initWithResumeData:delegate:path:

Returns an initialized NSURLDownload object that will resume downloading the specified data to the specified file and begins the download.

```
- (id)initWithResumeData:(NSData *)resumeData delegate:(id)delegate path:(NSString
    *)path
```

Parameters

resumeData

Specifies the data to resume downloading.

delegate

The delegate for the download. This object will receive delegate messages as the download progresses. Delegate messages will be sent on the thread which calls this method. For the download to work correctly the calling thread's run loop must be operating in the default run loop mode.

path

The location for the downloaded data.

Return Value
An initialized NSURLDownload object.

Availability
Available in Mac OS X v10.4 and later.

See Also
`resumeData` (page 1784)

Declared In
`NSURLDownload.h`

request

Returns the request that initiated the receiver's download.

```
- (NSURLRequest *)request
```

Return Value
The URL request that initiated the receiver's download.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLDownload.h`

resumeData

Returns the resume data for a download that is not yet complete.

```
- (NSData *)resumeData
```

Return Value
The resume data for a download that is not yet complete. This data represents the necessary state information that an `NSURLDownload` object needs to resume a download. The resume data can later be used when initializing a download with `initWithResumeData:delegate:path:` (page 1783). Returns `nil` if the download is not able to be resumed.

Discussion
Resume data will only be returned if the protocol of the download as well as the server support resuming. In order to later resume a download you must call `setDeletesFileUponFailure:` (page 1785) passing `NO` so the partially downloaded data is not deleted when the initial connection is lost or canceled.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSURLDownload.h`

setDeletesFileUponFailure:

Specifies whether the receiver deletes the partially downloaded file when a download stops prematurely.

```
- (void)setDeletesFileUponFailure:(BOOL)deletesFileUponFailure
```

Parameters

deletesFileUponFailure
`YES` if partially downloaded files should be deleted when a download stops prematurely, `NO` otherwise. The default is `YES`.

Discussion
To allow the download to be resumed in case the download ends prematurely you should call this method as soon as possible after starting the download.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- deletesFileUponFailure` (page 1783)

Declared In
`NSURLDownload.h`

setDestination:allowOverwrite:

Sets the destination path of the downloaded file.

```
- (void)setDestination:(NSString *)path allowOverwrite:(BOOL)allowOverwrite
```

Parameters

path
The path for the downloaded file.

allowOverwrite
`YES` if an existing file at *path* can be replaced, `NO` otherwise.

Discussion
If *allowOverwrite* is `NO` and a file already exists at *path*, a unique filename will be created for the downloaded file by appending a number to the filename. The delegate can implement `download:didCreateDestination:` (page 1787) to determine the filename used when the file is written to disk.

Special Considerations

An `NSURLDownload` instance ignores multiple calls to this method.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also

- `download:decideDestinationWithSuggestedFilename:` (page 1786)
- `download:didCreateDestination:` (page 1787)

Declared In

`NSURLDownload.h`

Delegate Methods

download:decideDestinationWithSuggestedFilename:

The delegate receives this message when *download* has determined a suggested filename for the downloaded file.

```
- (void)download:(NSURLDownload *)download
    decideDestinationWithSuggestedFilename:(NSString *)filename
```

Parameters

download
The URL download object sending the message.

filename
The suggested filename for the download.

Discussion

The suggested filename is either derived from the last path component of the URL and the MIME type or, if the download was encoded, from the encoding. If the delegate wishes to modify the path, it should send `setDestination:allowOverwrite:` (page 1785) to *download*.

Special Considerations

The delegate will not receive this message if `setDestination:allowOverwrite:` has already been called for the download.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In

`NSURLDownload.h`

download:didCancelAuthenticationChallenge:

Sent if an authentication challenge is canceled due to the protocol implementation encountering an error.

```
- (void)download:(NSURLDownload *)download
    didCancelAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
```

Parameters

download

The URL download object sending the message.

challenge

The authentication challenge that caused the download object to cancel the download.

Discussion

If the delegate receives this message the download will fail and the delegate will receive a `download:didFailWithError:` (page 1787) message.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

`NSURLDownload.h`

download:didCreateDestination:

Sent when the destination file is created.

```
- (void)download:(NSURLDownload *)download didCreateDestination:(NSString *)path
```

Parameters

download

The URL download object sending the message.

path

The path to the destination file.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

`NSURLDownload.h`

download:didFailWithError:

Sent if the download fails or if an I/O error occurs when the file is written to disk.

```
- (void)download:(NSURLDownload *)download didFailWithError:(NSError *)error
```

Parameters

download

The URL download object sending the message.

error

The error that caused the failure of the download.

Discussion
Any partially downloaded file will be deleted.

Special Considerations

Once the delegate receives this message, it will receive no further messages for *download*.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSURLDownload.h

download:didReceiveAuthenticationChallenge:

Sent when the URL download must authenticate a challenge in order to download the request.

```
- (void)download:(NSURLDownload *)download
    didReceiveAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
```

Parameters

download
The URL download object sending the message.

challenge
The URL authentication challenge that must be authenticated in order to download the request.

Discussion
This method gives the delegate the opportunity to determine the course of action taken for the challenge: provide credentials, continue without providing credentials or cancel the authentication challenge and the download.

The delegate can determine the number of previous authentication challenges by sending the message `previousFailureCount` (page 1739) to *challenge*.

If the previous failure count is 0 and the value returned by `proposedCredential` (page 1740) is `nil`, the delegate can create a new NSURLCredential object, providing a user name and password, and send a `useCredential:forAuthenticationChallenge:` (page 2126) message to `[challenge sender]`, passing the credential and *challenge* as parameters. If `proposedCredential` is not `nil`, the value is a credential from the URL or the shared credential storage that can be provided to the user as feedback.

The delegate may decide to abandon further attempts at authentication at any time by sending `[challenge sender]` a `continueWithoutCredentialForAuthenticationChallenge:` (page 2126) or a `cancelAuthenticationChallenge:` (page 2126) message. The specific action will be implementation dependent.

If the delegate implements this method, the download will suspend until `[challenge sender]` is sent one of the following messages: `useCredential:forAuthenticationChallenge:` (page 2126), `continueWithoutCredentialForAuthenticationChallenge:` (page 2126) or `cancelAuthenticationChallenge:` (page 2126).

If the delegate does not implement this method the default implementation is used. If a valid credential for the request is provided as part of the URL, or is available from the NSURLCredentialStorage the `[challenge sender]` is sent a `useCredential:forAuthenticationChallenge:` (page 2126) with the credential. If the

challenge has no credential or the credentials fail to authorize access, then `continueWithoutCredentialForAuthenticationChallenge:` (page 2126) is sent to `[challenge sender]` instead.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLDownload.h`

download:didReceiveDataOfLength:

Sent as a download object receives data incrementally.

```
- (void)download:(NSURLDownload *)download didReceiveDataOfLength:(NSUInteger)length
```

Parameters

download
The URL download object sending the message.

length
The amount of data received in this increment of the download, measured in bytes.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLDownload.h`

download:didReceiveResponse:

Sent when a download object has received sufficient load data to construct the NSURLResponse object for the download.

```
- (void)download:(NSURLDownload *)download didReceiveResponse:(NSURLResponse
    *)response
```

Parameters

download
The URL download object sending the message.

response
The URL response object received as part of the download. *response* is immutable and will not be modified after this method is called.

Discussion
In some rare cases, multiple responses may be received for a single download. In this case, the client should assume that each new response resets the download progress to 0 and should check the new response for the expected content length.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In
NSURLDownload.h

download:shouldDecodeSourceDataOfMIMEType:

Sent when a download object determines that the downloaded file is encoded to inquire whether the file should be automatically decoded.

```
- (BOOL)download:(NSURLDownload *)download shouldDecodeSourceDataOfMIMEType:(NSString
    *)encodingType
```

Parameters

download

The URL download object sending the message.

encodingType

The type of encoding used by the downloaded file. The supported encoding formats are MacBinary ("application/macbinary"), Binhex ("application/mac-binhex40") and gzip ("application/gzip").

Return Value
YES to decode the file, NO otherwise.

Special Considerations

The delegate may receive this message more than once if the file has been encoded multiple times. This method is not called if the downloaded file is not encoded.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSURLDownload.h

download:willResumeWithResponse:fromByte:

Sent when a download object has received a response from the server after attempting to resume a download.

```
- (void)download:(NSURLDownload *)download willResumeWithResponse:(NSURLResponse
    *)response fromByte:(long long)startingByte
```

Parameters

download

The URL download object sending the message.

response

The URL response received from the server in response to an attempt to resume a download.

startingByte

The location of the start of the resumed data, in bytes.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSURLDownload.h

download:willSendRequest:redirectResponse:

Sent when the download object determines that it must change URLs in order to continue loading a request.

```
- (NSURLRequest *)download:(NSURLDownload *)download willSendRequest:(NSURLRequest
    *)request redirectResponse:(NSURLResponse *)redirectResponse
```

Parameters

download
The URL download object sending the message.

request
The proposed redirected request. The delegate should inspect the redirected request to verify that it meets its needs, and create a copy with new attributes to return to the connection if necessary.

redirectResponse
The URL response that caused the redirect. May be *nil* in cases where this method is not being sent as a result of involving the delegate in redirect processing.

Return Value
The actual URL request to use in light of the redirection response. The delegate may copy and modify *request* as necessary to change its attributes, return *request* unmodified, or return nil.

Discussion
If the delegate wishes to cancel the redirect, it should call the *download* object's cancel (page 1782) method. Alternatively, the delegate method can return nil to cancel the redirect, and the download will continue to process. This has special relevance in the case where *redirectResponse* is not nil. In this case, any data that is loaded for the download will be sent to the delegate, and the delegate will receive a downloadDidFinish: (page 1792) or download:didFailWithError: (page 1787) message, as appropriate.

Special Considerations

The delegate can receive this message as a result of transforming a request's URL to its canonical form, or for protocol-specific reasons, such as an HTTP redirect. The delegate implementation should be prepared to receive this message multiple times.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSURLDownload.h

downloadDidBegin:

Sent immediately after a download object begins a download.

```
- (void)downloadDidBegin:(NSURLDownload *)download
```

Parameters

download
The URL download object sending the message.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLDownload.h`

downloadDidFinish:

Sent when a download object has completed downloading successfully and has written its results to disk.

```
- (void)downloadDidFinish:(NSURLDownload *)download
```

Parameters

download

The URL download object sending the message.

Discussion
The delegate will receive no further messages for *download*.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLDownload.h`

NSURLHandle Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSURLHandle.h

`NSURLHandle` is deprecated in Mac OS X v10.4 and later. Applications that are intended for deployment on Mac OS X v10.3 or later should use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.

Overview

`NSURLHandle` declares the programmatic interface for an object that accesses and manages resource data indicated by an `NSURL` object. A single `NSURLHandle` can service multiple equivalent `NSURL` objects, but only if these URLs map to the same resource.

Cocoa provides private concrete subclasses to handle HTTP and file URL schemes. If you want to implement support for additional URL schemes, you would do so by creating a subclass of `NSURLHandle`. You can use `NSURL` and `NSURLHandle` to download from FTP sites without subclassing.

Tasks

Constructing NSURLHandles

`+ cachedHandleForURL:` (page 1795) Deprecated in Mac OS X v10.4 and later

Returns the URL handle from the cache that has serviced the specified URL or another identical URL. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

`- initWithURL:cached:` (page 1801) Deprecated in Mac OS X v10.4 and later

Initializes a newly created URL handle with the specified URL. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

Managing Subclasses

`+ canInitWithURL:` (page 1796) Deprecated in Mac OS X v10.4 and later
Returns whether a URL handle can be initialized with a given URL. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

`+ registerURLHandleClass:` (page 1796) Deprecated in Mac OS X v10.4 and later
Registers a subclass of `NSURLHandle` as an available subclass for handling URLs (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

`+ URLHandleClassForURL:` (page 1797) Deprecated in Mac OS X v10.4 and later
Returns the class of the URL handle that will be used for a specified URL. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

Managing Clients

`- addClient:` (page 1797) Deprecated in Mac OS X v10.4 and later
Adds a client of the URL handle. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

`- removeClient:` (page 1803) Deprecated in Mac OS X v10.4 and later
Removes *`client`* as an `NSURLHandleClient` of the receiver. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

Setting and Getting Resource Properties

`- propertyForKey:` (page 1802) Deprecated in Mac OS X v10.4 and later
Returns the property for the specified key. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

`- propertyForKeyIfAvailable:` (page 1803) Deprecated in Mac OS X v10.4 and later
Returns the property for the specified key only if the value is already available; that is, the client doesn't need to do any work. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

`- writeProperty:forKey:` (page 1804) Deprecated in Mac OS X v10.4 and later
Sets the property of the receiver's resource for a specified key to the specified value. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

Loading Resource Data

`- availableResourceData` (page 1798) Deprecated in Mac OS X v10.4 and later
Immediately returns the currently available resource data managed by the URL handle. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

`- backgroundLoadDidFailWithReason:` (page 1798) Deprecated in Mac OS X v10.4 and later
Called when a background load fails. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

`- beginLoadInBackground` (page 1798) Deprecated in Mac OS X v10.4 and later
Called when a background load begins. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

- `cancelLoadInBackground` (page 1799) Deprecated in Mac OS X v10.4 and later
  Called to cancel a load currently in progress. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)
- `didLoadBytes:loadComplete:` (page 1799) Deprecated in Mac OS X v10.4 and later
  Appends new data to the receiver's resource data. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)
- `endLoadInBackground` (page 1800) Deprecated in Mac OS X v10.4 and later
  Halts any background loading. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)
- `expectedResourceDataSize` (page 1800) Deprecated in Mac OS X v10.4 and later
  Returns the expected length of the resource data if it is provided by the server. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)
- `failureReason` (page 1800) Deprecated in Mac OS X v10.4 and later
  Returns a string describing the reason a load failed. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)
- `flushCachedData` (page 1801) Deprecated in Mac OS X v10.4 and later
  Flushes any cached data for the URL served by this URL handle. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)
- `loadInBackground` (page 1801) Deprecated in Mac OS X v10.4 and later
  Loads the receiver's data in the background. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)
- `loadInForeground` (page 1802) Deprecated in Mac OS X v10.4 and later
  Loads the receiver's data synchronously. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)
- `resourceData` (page 1803) Deprecated in Mac OS X v10.4 and later
  Returns the resource data managed by the receiver, loading it if necessary. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)
- `status` (page 1804) Deprecated in Mac OS X v10.4 and later
  Returns the status of the receiver. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

Writing Resource Data

- `writeData:` (page 1804) Deprecated in Mac OS X v10.4 and later
  Attempts to write a specified set of data to the location specified by the receiver's URL. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

Class Methods

cachedHandleForURL:

Returns the URL handle from the cache that has serviced the specified URL or another identical URL. (Deprecated in Mac OS X v10.4 and later. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

```
+ (NSURLHandle *)cachedHandleForURL:(NSURL *)aURL
```

Parameters

aURL

The URL whose cached URL handle is desired.

Return Value

The URL handle from the cache that has serviced *aURL* or another identical URL. Returns `nil` if there is no such handle.

Discussion

Subclasses of `NSURLHandle` must override this method.

Availability

Deprecated in Mac OS X v10.4 and later.

Declared In

`NSURLHandle.h`

canInitWithURL:

Returns whether a URL handle can be initialized with a given URL. (Deprecated in Mac OS X v10.4 and later. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

```
+ (BOOL)canInitWithURL:(NSURL *)aURL
```

Parameters

aURL

The URL in question.

Return Value

`YES` if a URL handle can be initialized with *aURL*, `NO` otherwise.

Discussion

Subclasses of `NSURLHandle` must override this method to identify which URLs they can service.

Availability

Deprecated in Mac OS X v10.4 and later.

Declared In

`NSURLHandle.h`

registerURLHandleClass:

Registers a subclass of `NSURLHandle` as an available subclass for handling URLs (Deprecated in Mac OS X v10.4 and later. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

```
+ (void)registerURLHandleClass:(Class)aURLHandleSubclass
```

Parameters

aURLHandleSubclass

The new subclass to register as an available subclass.

Availability

Deprecated in Mac OS X v10.4 and later.

Declared In
NSURLHandle.h

URLHandleClassForURL:

Returns the class of the URL handle that will be used for a specified URL. (Deprecated in Mac OS X v10.4 and later. Use NSURLConnection or NSURLDownload instead; see *URL Loading System*.)

```
+ (Class)URLHandleClassForURL:(NSURL *)aURL
```

Parameters

aURL

The URL in question.

Return Value
The class of the URL handle that will be used for *aURL*.

Discussion
Subclasses of NSURLHandle must be registered via the registerURLHandleClass: (page 1796) method. The subclass is determined by asking the list of registered subclasses if it canInitWithURL: (page 1796); the first class to respond YES is selected.

Availability
Deprecated in Mac OS X v10.4 and later.

Declared In
NSURLHandle.h

Instance Methods

addClient:

Adds a client of the URL handle. (Deprecated in Mac OS X v10.4 and later. Use NSURLConnection or NSURLDownload instead; see *URL Loading System*.)

```
- (void)addClient:(id < NSURLHandleClient >)client
```

Parameters

client

An object conforming to the NSURLHandleClient protocol.

Availability
Deprecated in Mac OS X v10.4 and later.

Declared In
NSURLHandle.h

availableResourceData

Immediately returns the currently available resource data managed by the URL handle. (Deprecated in Mac OS X v10.4 and later. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

```
- (NSData *)availableResourceData
```

Return Value
The currently available resource data managed by the URL handle. Returns `nil` if a previous attempt to load the data failed.

Availability
Deprecated in Mac OS X v10.4 and later.

Declared In
`NSURLHandle.h`

backgroundLoadDidFailWithReason:

Called when a background load fails. (Deprecated in Mac OS X v10.4 and later. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

```
- (void)backgroundLoadDidFailWithReason:(NSString *)reason
```

Parameters

reason
The status message indicating why the background load failed.

Discussion
This method is provided mainly for subclasses that wish to take some action before passing along the failure notification to the URL client. This method should invoke `super`'s implementation before returning.

Availability
Deprecated in Mac OS X v10.4 and later.

Declared In
`NSURLHandle.h`

beginLoadInBackground

Called when a background load begins. (Deprecated in Mac OS X v10.4 and later. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

```
- (void)beginLoadInBackground
```

Discussion
This method is provided mainly for subclasses that wish to take advantage of the superclass failure-reporting mechanism.

Availability
Deprecated in Mac OS X v10.4 and later.

See Also
`- endLoadInBackground` (page 1800)

- loadInBackground (page 1801)

Declared In
NSURLHandle.h

cancelLoadInBackground

Called to cancel a load currently in progress. (Deprecated in Mac OS X v10.4 and later. Use NSURLConnection or NSURLDownload instead; see *URL Loading System*.)

```
- (void)cancelLoadInBackground
```

Discussion
This method is provided mainly for subclasses that wish to take some action before a background load is canceled. This method should invoke super's implementation before returning.

Availability
Deprecated in Mac OS X v10.4 and later.

See Also
- endLoadInBackground (page 1800)

Declared In
NSURLHandle.h

didLoadBytes:loadComplete:

Appends new data to the receiver's resource data. (Deprecated in Mac OS X v10.4 and later. Use NSURLConnection or NSURLDownload instead; see *URL Loading System*.)

```
- (void)didLoadBytes:(NSData *)newBytes loadComplete:(BOOL)done
```

Parameters

newBytes
The newly loaded bytes.

done
YES if newBytes contains the last piece of data for the URL, NO otherwise.

Discussion
You should call this method when loading the resource data in the background.

Availability
Deprecated in Mac OS X v10.4 and later.

See Also
- loadInBackground (page 1801)

Declared In
NSURLHandle.h

endLoadInBackground

Halts any background loading. (Deprecated in Mac OS X v10.4 and later. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

```
- (void)endLoadInBackground
```

Discussion
This method is called by `cancelLoadInBackground` (page 1799).

Availability
Deprecated in Mac OS X v10.4 and later.

See Also
- `beginLoadInBackground` (page 1798)
- `cancelLoadInBackground` (page 1799)
- `loadInBackground` (page 1801)

Declared In
`NSURLHandle.h`

expectedResourceDataSize

Returns the expected length of the resource data if it is provided by the server. (Deprecated in Mac OS X v10.4 and later. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

```
- (long long)expectedResourceDataSize
```

Return Value
The expected size of the resource data, in bytes. A negative value if the length is unknown.

Discussion
This information can be queried before all the data has arrived.

Availability
Deprecated in Mac OS X v10.4 and later.
Available in Mac OS X v10.3 and later.

Declared In
`NSURLHandle.h`

failureReason

Returns a string describing the reason a load failed. (Deprecated in Mac OS X v10.4 and later. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

```
- (NSString *)failureReason
```

Return Value
A string describing the reason a load failed. If the load has not failed, returns `nil`.

Availability
Deprecated in Mac OS X v10.4 and later.

Declared In
NSURLHandle.h

flushCachedData

Flushes any cached data for the URL served by this URL handle. (Deprecated in Mac OS X v10.4 and later. Use NSURLConnection or NSURLDownload instead; see *URL Loading System*.)

```
- (void)flushCachedData
```

Availability
Deprecated in Mac OS X v10.4 and later.

Declared In
NSURLHandle.h

initWithURL:cached:

Initializes a newly created URL handle with the specified URL. (Deprecated in Mac OS X v10.4 and later. Use NSURLConnection or NSURLDownload instead; see *URL Loading System*.)

```
- (id)initWithURL:(NSURL *)aURL cached:(BOOL)willCache
```

Parameters

aURL
The URL for the new handle.

willCache
YES if the URL handle should cache its data and respond to requests from equivalent URLs for the cached data, NO otherwise.

Discussion
Subclasses of NSURLHandle must override this method.

Availability
Deprecated in Mac OS X v10.4 and later.

Declared In
NSURLHandle.h

loadInBackground

Loads the receiver's data in the background. (Deprecated in Mac OS X v10.4 and later. Use NSURLConnection or NSURLDownload instead; see *URL Loading System*.)

```
- (void)loadInBackground
```

Discussion
Each subclass determines its own loading policy. Clients should not assume that multiple background loads can proceed simultaneously. For example, a subclass may maintain only one thread for background loading, so only one background loading operation can be in progress at a time. If multiple background loads are requested, the later requests will wait in a queue until earlier requests are handled.

Availability
Deprecated in Mac OS X v10.4 and later.

See Also
- `beginLoadInBackground` (page 1798)

Declared In
`NSURLHandle.h`

loadInForeground

Loads the receiver's data synchronously. (Deprecated in Mac OS X v10.4 and later. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

```
- (NSData *)loadInForeground
```

Return Value
The loaded data.

Discussion
Called by `resourceData` (page 1803). Subclasses of NSURLHandle must override this method.

Availability
Deprecated in Mac OS X v10.4 and later.

Declared In
`NSURLHandle.h`

propertyForKey:

Returns the property for the specified key. (Deprecated in Mac OS X v10.4 and later. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

```
- (id)propertyForKey:(NSString *)propertyKey
```

Parameters
propertyKey
The key of the desired property.

Return Value
The value associated with *propertyKey*. Returns `nil` if there is no such key.

Discussion
Subclasses of NSURLHandle must override this method.

Availability
Deprecated in Mac OS X v10.4 and later.

See Also
- `writeProperty:forKey:` (page 1804)
- `propertyForKeyIfAvailable:` (page 1803)

Declared In
`NSURLHandle.h`

propertyForKeyIfAvailable:

Returns the property for the specified key only if the value is already available; that is, the client doesn't need to do any work. (Deprecated in Mac OS X v10.4 and later. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

```
- (id)propertyForKeyIfAvailable:(NSString *)propertyKey
```

Parameters

propertyKey

The key of the desired property.

Return Value

The value associated with *propertyKey*. Returns `nil` if there is no such key or if the client would have to do work to fetch the property.

Availability

Deprecated in Mac OS X v10.4 and later.

Declared In

`NSURLHandle.h`

removeClient:

Removes *client* as an `NSURLHandleClient` of the receiver. (Deprecated in Mac OS X v10.4 and later. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

```
- (void)removeClient:(id < NSURLHandleClient >)client
```

Parameters

client

An object conforming to the `NSURLHandleClient` protocol.

Availability

Deprecated in Mac OS X v10.4 and later.

Declared In

`NSURLHandle.h`

resourceData

Returns the resource data managed by the receiver, loading it if necessary. (Deprecated in Mac OS X v10.4 and later. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

```
- (NSData *)resourceData
```

Return Value

The resource data managed by the receiver.

Discussion

Blocks until all data is available.

Availability

Deprecated in Mac OS X v10.4 and later.

Declared In
NSURLHandle.h

status

Returns the status of the receiver. (Deprecated in Mac OS X v10.4 and later. Use NSURLConnection or NSURLDownload instead; see *URL Loading System*.)

```
- (NSURLHandleStatus)status
```

Return Value
The status of the receiver. Possible return statuses are described in "Constants" (page 1805).

Availability
Deprecated in Mac OS X v10.4 and later.

Declared In
NSURLHandle.h

writeData:

Attempts to write a specified set of data to the location specified by the receiver's URL. (Deprecated in Mac OS X v10.4 and later. Use NSURLConnection or NSURLDownload instead; see *URL Loading System*.)

```
- (BOOL)writeData:(NSData *)data
```

Parameters
data
The data to write.

Return Value
YES if successful, NO otherwise.

Discussion
Must be overridden by subclasses.

Availability
Deprecated in Mac OS X v10.4 and later.

Declared In
NSURLHandle.h

writeProperty:forKey:

Sets the property of the receiver's resource for a specified key to the specified value. (Deprecated in Mac OS X v10.4 and later. Use NSURLConnection or NSURLDownload instead; see *URL Loading System*.)

```
- (BOOL)writeProperty:(id)propertyValue forKey:(NSString *)propertyKey
```

Parameters
propertyValue
The new value for the property.

propertyKey
The key of the desired property.

Return Value
`YES` if the modification was successful, `NO` otherwise.

Discussion
Must be overridden by subclasses.

Availability
Deprecated in Mac OS X v10.4 and later.

See Also
`- propertyForKey:` (page 1802)

Declared In
`NSURLHandle.h`

Constants

NSURLHandleStatus

These following constants are defined by `NSURLHandle` and are returned by `status` (page 1804).

```
typedef enum {
    NSURLHandleNotLoaded = 0,
    NSURLHandleLoadSucceeded,
    NSURLHandleLoadInProgress,
    NSURLHandleLoadFailed
} NSURLHandleStatus;
```

Constants
`NSURLHandleNotLoaded`

The resource data has not been loaded. (**Deprecated.** Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

Available in Mac OS X v10.0 and later.

Declared in `NSURLHandle.h`.

`NSURLHandleLoadSucceeded`

The resource data was successfully loaded. (**Deprecated.** Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

Available in Mac OS X v10.0 and later.

Declared in `NSURLHandle.h`.

`NSURLHandleLoadInProgress`

The resource data is in the process of loading. (**Deprecated.** Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

Available in Mac OS X v10.0 and later.

Declared in `NSURLHandle.h`.

`NSURLHandleLoadFailed`

The resource data failed to load. (**Deprecated.** Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

Available in Mac OS X v10.0 and later.

Declared in `NSURLHandle.h`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSURLHandle.h`

NSURLProtectionSpace Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSURLProtectionSpace.h
Availability	Available in Mac OS X v10.2 with Safari 1.0 installed. Available in Mac OS X v10.2.7 and later.
Companion guide	URL Loading System

Overview

NSURLProtectionSpace represents a server or an area on a server, commonly referred to as a realm, that requires authentication. An NSURLProtectionSpace's credentials apply to any requests within that protection space.

Adopted Protocols

NSCopying
- `copyWithZone:` (page 2042)

Tasks

Creating a Protection Space

- `initWithHost:port:protocol:realm:authenticationMethod:` (page 1809)
  Initializes a protection space object.
- `initWithProxyHost:port:type:realm:authenticationMethod:` (page 1809)
  Initializes a protection space object representing a proxy server.

Getting Protection Space Properties

- `authenticationMethod` (page 1808)
  Returns the authentication method used by the receiver.
- `host` (page 1808)
  Returns the receiver's host.
- `isProxy` (page 1810)
  Returns whether the receiver represents a proxy server.
- `port` (page 1810)
  Returns the receiver's port.
- `protocol` (page 1811)
  Returns the receiver's protocol.
- `proxyType` (page 1811)
  Returns the receiver's proxy type.
- `realm` (page 1811)
  Returns the receiver's authentication realm
- `receivesCredentialSecurely` (page 1812)
  Returns whether the credentials for the protection space can be sent securely.

Instance Methods

authenticationMethod

Returns the authentication method used by the receiver.

```
- (NSString *)authenticationMethod
```

Return Value
The authentication method used by the receiver. The supported authentication methods are listed in "Constants" (page 1812).

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLProtectionSpace.h`

host

Returns the receiver's host.

```
- (NSString *)host
```

Return Value
The receiver's host.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSURLProtectionSpace.h

initWithHost:port:protocol:realm:authenticationMethod:

Initializes a protection space object.

```
- (id)initWithHost:(NSString *)host port:(NSInteger)port protocol:(NSString
    *)protocol realm:(NSString *)realm authenticationMethod:(NSString
    *)authenticationMethod
```

Parameters

host
The host name for the protection space object.

port
The port for the protection space object. If *port* is 0 the default port for the specified protocol is used, for example, port 80 for HTTP. Note that servers can, and do, treat these values differently.

protocol
The protocol for the protection space object. The value of *protocol* is equivalent to the scheme for a URL in the protection space, for example, "http", "https", "ftp", etc.

realm
A string indicating a protocol specific subdivision of the host. *realm* may be `nil` if there is no specified realm or if the protocol doesn't support realms.

authenticationMethod
The type of authentication to use. *authenticationMethod* should be set to one of the values in "Constants" (page 1812) or `nil` to use the default, `NSURLAuthenticationMethodDefault`.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
- initWithProxyHost:port:type:realm:authenticationMethod: (page 1809)

Declared In
NSURLProtectionSpace.h

initWithProxyHost:port:type:realm:authenticationMethod:

Initializes a protection space object representing a proxy server.

```
- (id)initWithProxyHost:(NSString *)host port:(NSInteger)port type:(NSString
    *)proxyType realm:(NSString *)realm authenticationMethod:(NSString
    *)authenticationMethod
```

Parameters

host

The host of the proxy server for the protection space object.

port

The port for the protection space object. If *port* is 0 the default port for the specified proxy type is used, for example, port 80 for HTTP. Note that servers can, and do, treat these values differently.

proxyType

The type of proxy server. The value of *proxyType* should be set to one of the values specified in "Constants" (page 1812).

realm

A string indicating a protocol specific subdivision of the host. *realm* may be `nil` if there is no specified realm or if the protocol doesn't support realms.

authenticationMethod

The type of authentication to use. *authenticationMethod* should be set to one of the values in "Constants" (page 1812) or `nil` to use the default, `NSURLAuthenticationMethodDefault`.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

See Also

`- initWithHost:port:protocol:realm:authenticationMethod:` (page 1809)

Declared In

`NSURLProtectionSpace.h`

isProxy

Returns whether the receiver represents a proxy server.

```
- (BOOL)isProxy
```

Return Value

`YES` if the receiver represents a proxy server, `NO` otherwise.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

`NSURLProtectionSpace.h`

port

Returns the receiver's port.

```
- (NSInteger)port
```

Return Value

The receiver's port.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLProtectionSpace.h`

protocol

Returns the receiver's protocol.

```
- (NSString *)protocol
```

Return Value
The receiver's protocol, or `nil` if the receiver represents a proxy protection space.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLProtectionSpace.h`

proxyType

Returns the receiver's proxy type.

```
- (NSString *)proxyType
```

Return Value
The receiver's proxy type, or `nil` if the receiver does not represent a proxy protection space. The supported proxy types are listed in "Constants" (page 1812).

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLProtectionSpace.h`

realm

Returns the receiver's authentication realm

```
- (NSString *)realm
```

Return Value
The receiver's authentication realm, or `nil` if no realm has been set.

Discussion
A realm is generally only specified for HTTP and HTTPS authentication.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLProtectionSpace.h`

receivesCredentialSecurely

Returns whether the credentials for the protection space can be sent securely.

```
- (BOOL)receivesCredentialSecurely
```

Return Value
`YES` if the credentials for the protection space represented by the receiver can be sent securely, `NO` otherwise.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLProtectionSpace.h`

Constants

NSURLProtectionSpace Proxy Types

These constants describe the supported proxy types used in `initWithProxyHost:port:type:realm:authenticationMethod:` (page 1809) and returned by `proxyType` (page 1811).

```
extern NSString *NSURLProtectionSpaceHTTPProxy;
extern NSString *NSURLProtectionSpaceHTTPSProxy;
extern NSString *NSURLProtectionSpaceFTPProxy;
extern NSString *NSURLProtectionSpaceSOCKSProxy;
```

Constants

`NSURLProtectionSpaceHTTPProxy`
The proxy type for HTTP proxies.
Available in Mac OS X v10.2 and later.
Declared in `NSURLProtectionSpace.h`.

`NSURLProtectionSpaceHTTPSProxy`
The proxy type for HTTPS proxies.
Available in Mac OS X v10.2 and later.
Declared in `NSURLProtectionSpace.h`.

`NSURLProtectionSpaceFTPProxy`
The proxy type for FTP proxies.

Available in Mac OS X v10.2 and later.

Declared in `NSURLProtectionSpace.h`.

`NSURLProtectionSpaceSOCKSProxy`
The proxy type for SOCKS proxies.

Available in Mac OS X v10.2 and later.

Declared in `NSURLProtectionSpace.h`.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLProtectionSpace.h`

NSURLProtectionSpace Authentication Methods

These constants describe the available authentication methods used in `initWithHost:port:protocol:realm:authenticationMethod:` (page 1809), `initWithProxyHost:port:type:realm:authenticationMethod:` (page 1809) and returned by `authenticationMethod` (page 1808).

```
extern NSString *NSURLAuthenticationMethodDefault;
extern NSString *NSURLAuthenticationMethodHTTPBasic;
extern NSString *NSURLAuthenticationMethodHTTPDigest;
extern NSString *NSURLAuthenticationMethodHTMLForm;
```

Constants
`NSURLAuthenticationMethodDefault`
Use the default authentication method for a protocol.

Available in Mac OS X v10.2 and later.

Declared in `NSURLProtectionSpace.h`.

`NSURLAuthenticationMethodHTTPBasic`
Use HTTP basic authentication for this protection space.

This is equivalent to `NSURLAuthenticationMethodDefault` for HTTP.

Available in Mac OS X v10.2 and later.

Declared in `NSURLProtectionSpace.h`.

`NSURLAuthenticationMethodHTTPDigest`
Use HTTP digest authentication for this protection space.

Available in Mac OS X v10.2 and later.

Declared in `NSURLProtectionSpace.h`.

`NSURLAuthenticationMethodHTMLForm`
Use HTML form authentication for this protection space.

This authentication method can apply to any protocol.

Available in Mac OS X v10.2 and later.

Declared in `NSURLProtectionSpace.h`.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLProtectionSpace.h`

NSURLProtocol Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSURLProtocol.h
Availability	Available in Mac OS X v10.2 with Safari 1.0 installed. Available in Mac OS X v10.2.7 and later.
Companion guide	URL Loading System

Overview

`NSURLProtocol` is an abstract class that provides the basic structure for performing protocol-specific loading of URL data. Concrete subclasses handle the specifics associated with one or more protocols or URL schemes.

An application should never need to directly instantiate an `NSURLProtocol` subclass. The instance of the appropriate `NSURLProtocol` subclass for an `NSURLRequest` is created by `NSURLConnection` when a download is started.

The `NSURLProtocolClient` protocol describes the methods an implementation uses to drive the URL loading system from a `NSURLProtocol` subclass.

To support customization of protocol-specific requests, protocol implementors are encouraged to provide categories on `NSURLRequest` and `NSMutableURLRequest`. Protocol implementors who need to extend the capabilities of `NSURLRequest` and `NSMutableURLRequest` in this way can store and retrieve protocol-specific request data by using `NSURLProtocol`'s class methods `propertyForKey:inRequest:` (page 1818) and `setProperty:forKey:inRequest:` (page 1820).

An essential responsibility for a protocol implementor is creating a `NSURLResponse` for each request it processes successfully. A protocol implementor may wish to create a custom, mutable `NSURLResponse` class to provide protocol specific information.

Tasks

Creating Protocol Objects

`- initWithRequest:cachedResponse:client:` (page 1822)
Initializes an `NSURLProtocol` object.

Registering and Unregistering Protocol Classes

`+ registerClass:` (page 1819)
Attempts to register a subclass of `NSURLProtocol`, making it visible to the URL loading system.

`+ unregisterClass:` (page 1821)
Unregisters the specified subclass of `NSURLProtocol`.

Getting and Setting Request Properties

`+ propertyForKey:inRequest:` (page 1818)
Returns the property associated with the specified key in the specified request.

`+ setProperty:forKey:inRequest:` (page 1820)
Sets the property associated with the specified key in the specified request.

`+ removePropertyForKey:inRequest:` (page 1819)
Removes the property associated with the specified key in the specified request.

Determining If a Subclass Can Handle a Request

`+ canInitWithRequest:` (page 1817)
Returns whether the protocol subclass can handle the specified request.

Providing a Canonical Version of a Request

`+ canonicalRequestForRequest:` (page 1817)
Returns a canonical version of the specified request.

Determining If Requests Are Cache Equivalent

`+ requestIsCacheEquivalent:toRequest:` (page 1820)
Returns whether two requests are equivalent for cache purposes.

Starting and Stopping Downloads

- `startLoading` (page 1823)
  Starts protocol-specific loading of the request.
- `stopLoading` (page 1823)
  Stops protocol-specific loading of the request.

Getting Protocol Attributes

- `cachedResponse` (page 1821)
  Returns the receiver's cached response.
- `client` (page 1822)
  Returns the object the receiver uses to communicate with the URL loading system.
- `request` (page 1822)
  Returns the receiver's request.

Class Methods

canInitWithRequest:

Returns whether the protocol subclass can handle the specified request.

```
+ (BOOL)canInitWithRequest:(NSURLRequest *)request
```

Parameters

request
  The request to be handled.

Return Value
`YES` if the protocol subclass can handle *request*, otherwise `NO`.

Discussion
A subclass should inspect *request* and determine whether or not the implementation can perform a load with that request.

This is an abstract method and subclasses must provide an implementation.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLProtocol.h`

canonicalRequestForRequest:

Returns a canonical version of the specified request.

```
+ (NSURLRequest *)canonicalRequestForRequest:(NSURLRequest *)request
```

Parameters

request

The request whose canonical version is desired.

Return Value

The canonical form of *request*.

Discussion

It is up to each concrete protocol implementation to define what "canonical" means. A protocol should guarantee that the same input request always yields the same canonical form.

Special consideration should be given when implementing this method, because the canonical form of a request is used to lookup objects in the URL cache, a process which performs equality checks between `NSURLRequest` objects.

This is an abstract method and subclasses must provide an implementation.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

`NSURLProtocol.h`

propertyForKey:inRequest:

Returns the property associated with the specified key in the specified request.

```
+ (id)propertyForKey:(NSString *)key inRequest:(NSURLRequest *)request
```

Parameters

key

The key of the desired property.

request

The request whose properties are to be queried.

Return Value

The property associated with *key*, or `nil` if no property has been stored for *key*.

Discussion

This method provides an interface for protocol implementors to access protocol-specific information associated with `NSURLRequest` objects.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

See Also

`+ setProperty:forKey:inRequest:` (page 1820)

Declared In

`NSURLProtocol.h`

registerClass:

Attempts to register a subclass of `NSURLProtocol`, making it visible to the URL loading system.

```
+ (BOOL)registerClass:(Class)protocolClass
```

Parameters

protocolClass

The subclass of NSURLProtocol to register.

Return Value

`YES` if the registration is successful, `NO` otherwise. The only failure condition is if *protocolClass* is not a subclass of NSURLProtocol.

Discussion

When the URL loading system begins to load a request, each registered protocol class is consulted in turn to see if it can be initialized with the specified request. The first `NSURLProtocol` subclass to return `YES` when sent a `canInitWithRequest:` (page 1817) message is used to perform the URL load. There is no guarantee that all registered protocol classes will be consulted.

Classes are consulted in the reverse order of their registration. A similar design governs the process to create the canonical form of a request with `canonicalRequestForRequest:` (page 1817).

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

See Also

`+ unregisterClass:` (page 1821)

Declared In

`NSURLProtocol.h`

removePropertyForKey:inRequest:

Removes the property associated with the specified key in the specified request.

```
+ (void)removePropertyForKey:((NSString *)key inRequest:(NSMutableURLRequest
    *)request
```

Parameters

key

The key whose value should be removed.

request

The request from which to remove the property value.

Discussion

This method is used to provide an interface for protocol implementors to customize protocol-specific information associated with `NSMutableURLRequest` objects.

Availability

Available in Mac OS X v10.5 and later.

See Also

`+ propertyForKey:inRequest:` (page 1818)

Declared In
NSURLProtocol.h

requestIsCacheEquivalent:toRequest:

Returns whether two requests are equivalent for cache purposes.

```
+ (BOOL)requestIsCacheEquivalent:(NSURLRequest *)aRequest toRequest:(NSURLRequest
    *)bRequest
```

Parameters

aRequest
The request to compare with *bRequest*.

bRequest
The request to compare with *aRequest*.

Return Value
YES if *aRequest* and *bRequest* are equivalent for cache purposes, NO otherwise. Requests are considered equivalent for cache purposes if and only if they would be handled by the same protocol and that protocol declares them equivalent after performing implementation-specific checks.

Discussion
The NSURLProtocol implementation of this method compares the URLs of the requests to determine if the requests should be considered equivalent. Subclasses can override this method to provide protocol-specific comparisons.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSURLProtocol.h

setProperty:forKey:inRequest:

Sets the property associated with the specified key in the specified request.

```
+ (void)setProperty:(id)value forKey:(NSString *)key inRequest:(NSMutableURLRequest
    *)request
```

Parameters

value
The value to set for the specified property.

key
The key for the specified property.

request
The request for which to create the property.

Discussion
This method is used to provide an interface for protocol implementors to customize protocol-specific information associated with NSMutableURLRequest objects.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
`+ propertyForKey:inRequest:` (page 1818)

Declared In
`NSURLProtocol.h`

unregisterClass:

Unregisters the specified subclass of `NSURLProtocol`.

```
+ (void)unregisterClass:(Class)protocolClass
```

Parameters
protocolClass
The subclass of NSURLProtocol to unregister.

Discussion
After this method is invoked, *protocolClass* is no longer consulted by the URL loading system.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
`+ registerClass:` (page 1819)

Declared In
`NSURLProtocol.h`

Instance Methods

cachedResponse

Returns the receiver's cached response.

```
- (NSCachedURLResponse *)cachedResponse
```

Return Value
The receiver's cached response.

Discussion
Subclasses must implement this method.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSURLProtocol.h

client

Returns the object the receiver uses to communicate with the URL loading system.

```
- (id < NSURLProtocolClient >)client
```

Return Value
The object the receiver uses to communicate with the URL loading system.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSURLProtocol.h

initWithRequest:cachedResponse:client:

Initializes an `NSURLProtocol` object.

```
- (id)initWithRequest:(NSURLRequest *)request cachedResponse:(NSCachedURLResponse
    *)cachedResponse client:(id < NSURLProtocolClient >)client
```

Parameters

request
The URL request for the URL protocol object.

cachedResponse
A cached response for the request; may be `nil` if there is no existing cached response for the request.

client
An object that provides an implementation of the NSURLProtocolClient protocol that the receiver uses to communicate with the URL loading system.

Discussion
Subclasses should override this method to do any custom initialization. An application should never explicitly call this method.

This is the designated intializer for `NSURLProtocol`.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSURLProtocol.h

request

Returns the receiver's request.

```
- (NSURLRequest *)request
```

Return Value
The receiver's request.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLProtocol.h`

startLoading

Starts protocol-specific loading of the request.

```
- (void)startLoading
```

Discussion
When this method is called, the subclass implementation should start loading the request, providing feedback to the URL loading system via the `NSURLProtocolClient` protocol.

Subclasses must implement this method.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
`- stopLoading` (page 1823)

Declared In
`NSURLProtocol.h`

stopLoading

Stops protocol-specific loading of the request.

```
- (void)stopLoading
```

Discussion
When this method is called, the subclass implementation should stop loading a request. This could be in response to a cancel operation, so protocol implementations must be able to handle this call while a load is in progress.

Subclasses must implement this method.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
`- startLoading` (page 1823)

Declared In
`NSURLProtocol.h`

NSURLRequest Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSMutableCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSURLRequest.h
Availability	Available in Mac OS X v10.2 with Safari 1.0 installed. Available in Mac OS X v10.2.7 and later.
Companion guide	URL Loading System
Related sample code	URL CacheInfo

Overview

`NSURLRequest` objects represent a URL load request in a manner independent of protocol and URL scheme.

NSURLRequest encapsulates two basic data elements of a load request: the URL to load, and the policy to use when consulting the URL content cache made available by the implementation.

NSURLRequest is designed to be extended to support additional protocols by adding categories that access protocol specific values from a property object using NSURLProtocol's `propertyForKey:inRequest:` (page 1818) and `setProperty:forKey:inRequest:` (page 1820) methods.

The mutable subclass of NSURLRequest is NSMutableURLRequest.

Adopted Protocols

NSCopying
- `copyWithZone:` (page 2042)

NSMutableCopying
- `mutableCopyWithZone:` (page 2094)

Tasks

Creating Requests

+ `requestWithURL:` (page 1827)
  Creates and returns a URL request for a specified URL with default cache policy and timeout value.
- `initWithURL:` (page 1830)
  Returns a URL request for a specified URL with default cache policy and timeout value.
+ `requestWithURL:cachePolicy:timeoutInterval:` (page 1827)
  Creates and returns an initialized URL request with specified values.
- `initWithURL:cachePolicy:timeoutInterval:` (page 1830)
  Returns an initialized URL request with specified values.

Getting Request Properties

- `cachePolicy` (page 1828)
  Returns the receiver's cache policy.
- `mainDocumentURL` (page 1831)
  Returns the main document URL associated with the request.
- `timeoutInterval` (page 1831)
  Returns the receiver's timeout interval, in seconds.
- `URL` (page 1832)
  Returns the request's URL.

Getting HTTP Request Properties

- `allHTTPHeaderFields` (page 1828)
  Returns a dictionary containing all the receiver's HTTP header fields.
- `HTTPBody` (page 1828)
  Returns the receiver's HTTP body data.
- `HTTPBodyStream` (page 1829)
  Returns the receiver's HTTP body stream.
- `HTTPMethod` (page 1829)
  Returns the receiver's HTTP request method.
- `HTTPShouldHandleCookies` (page 1830)
  Returns whether the default cookie handling will be used for this request.
- `valueForHTTPHeaderField:` (page 1832)
  Returns the value of the specified HTTP header field.

Class Methods

requestWithURL:

Creates and returns a URL request for a specified URL with default cache policy and timeout value.

```
+ (id)requestWithURL:(NSURL *)theURL
```

Parameters

theURL

The URL for the new request.

Return Value

The newly created URL request.

Discussion

The default cache policy is `NSURLRequestUseProtocolCachePolicy` and the default timeout interval is 60 seconds.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

See Also

`+ requestWithURL:cachePolicy:timeoutInterval:` (page 1827)

Declared In

`NSURLRequest.h`

requestWithURL:cachePolicy:timeoutInterval:

Creates and returns an initialized URL request with specified values.

```
+ (id)requestWithURL:(NSURL *)theURL cachePolicy:(NSURLRequestCachePolicy)cachePolicy
    timeoutInterval:(NSTimeInterval)timeoutInterval
```

Parameters

theURL

The URL for the new request.

cachePolicy

The cache policy for the new request.

timeoutInterval

The timeout interval for the new request, in seconds.

Return Value

The newly created URL request.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

See Also
`- initWithURL:cachePolicy:timeoutInterval:` (page 1830)

Related Sample Code
URL CacheInfo

Declared In
`NSURLRequest.h`

Instance Methods

allHTTPHeaderFields

Returns a dictionary containing all the receiver's HTTP header fields.

```
- (NSDictionary *)allHTTPHeaderFields
```

Return Value
A dictionary containing all the receiver's HTTP header fields.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
`- valueForHTTPHeaderField:` (page 1832)

Declared In
`NSURLRequest.h`

cachePolicy

Returns the receiver's cache policy.

```
- (NSURLRequestCachePolicy)cachePolicy
```

Return Value
The receiver's cache policy.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLRequest.h`

HTTPBody

Returns the receiver's HTTP body data.

```
- (NSData *)HTTPBody
```

Return Value
The receiver's HTTP body data.

Discussion
This data is sent as the message body of a request, as in an HTTP `POST` request.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLRequest.h`

HTTPBodyStream

Returns the receiver's HTTP body stream.

```
- (NSInputStream *)HTTPBodyStream
```

Return Value
The receiver's HTTP body stream, or `nil` if it has not been set. The returned stream is for examination only, it is not safe to manipulate the stream in any way.

Discussion
The receiver will have either an HTTP body or an HTTP body stream, only one may be set for a request. A HTTP body stream is preserved when copying an NSURLRequest object, but is lost when a request is archived using the NSCoding protocol.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSURLRequest.h`

HTTPMethod

Returns the receiver's HTTP request method.

```
- (NSString *)HTTPMethod
```

Return Value
The receiver's HTTP request method.

Discussion
The default HTTP method is "GET".

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLRequest.h`

HTTPShouldHandleCookies

Returns whether the default cookie handling will be used for this request.

```
- (BOOL)HTTPShouldHandleCookies
```

Return Value
`YES` if the default cookie handling will be used for this request, `NO` otherwise.

Discussion
The default is `YES`.

Special Considerations

In Mac OS X v10.2 with Safari 1.0 the value set by this method is not respected by the framework.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLRequest.h`

initWithURL:

Returns a URL request for a specified URL with default cache policy and timeout value.

```
- (id)initWithURL:(NSURL *)theURL
```

Parameters
theURL
The URL for the request.

Return Value
The initialized URL request.

Discussion
The default cache policy is `NSURLRequestUseProtocolCachePolicy` and the default timeout interval is 60 seconds.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
`- initWithURL:cachePolicy:timeoutInterval:` (page 1830)

Declared In
`NSURLRequest.h`

initWithURL:cachePolicy:timeoutInterval:

Returns an initialized URL request with specified values.

```
- (id)initWithURL:(NSURL *)theURL cachePolicy:(NSURLRequestCachePolicy)cachePolicy
    timeoutInterval:(NSTimeInterval)timeoutInterval
```

Parameters

theURL

The URL for the request.

cachePolicy

The cache policy for the request.

timeoutInterval

The timeout interval for the request, in seconds.

Return Value
The initialized URL request.

Discussion
This is the designated initializer for `NSURLRequest`.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

See Also
`- initWithURL:` (page 1830)

Declared In
`NSURLRequest.h`

mainDocumentURL

Returns the main document URL associated with the request.

```
- (NSURL *)mainDocumentURL
```

Return Value
The main document URL associated with the request.

Discussion
This URL is used for the cookie “same domain as main document” policy.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLRequest.h`

timeoutInterval

Returns the receiver’s timeout interval, in seconds.

```
- (NSTimeInterval)timeoutInterval
```

Return Value
The receiver's timeout interval, in seconds.

Discussion
If during a connection attempt the request remains idle for longer than the timeout interval, the request is considered to have timed out.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLRequest.h`

URL

Returns the request's URL.

```
- (NSURL *)URL
```

Return Value
The request's URL.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLRequest.h`

valueForHTTPHeaderField:

Returns the value of the specified HTTP header field.

```
- (NSString *)valueForHTTPHeaderField:(NSString *)field
```

Parameters

field

The name of the header field whose value is to be returned. In keeping with the HTTP RFC, HTTP header field names are case-insensitive.

Return Value
The value associated with the header field *field*, or `nil` if there is no corresponding header field.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLRequest.h`

Constants

NSURLRequestCachePolicy

These constants are used to specify interaction with the cached responses.

```
enum
{
    NSURLRequestUseProtocolCachePolicy = 0,
    NSURLRequestReloadIgnoringLocalCacheData = 1,
    NSURLRequestReloadIgnoringLocalAndRemoteCacheData =4,
   NSURLRequestReloadIgnoringCacheData = NSURLRequestReloadIgnoringLocalCacheData,
    NSURLRequestReturnCacheDataElseLoad = 2,
    NSURLRequestReturnCacheDataDontLoad = 3,
    NSURLRequestReloadRevalidatingCacheData = 5
};
typedef NSUInteger NSURLRequestCachePolicy;
```

Constants

`NSURLRequestUseProtocolCachePolicy`

Specifies that the caching logic defined in the protocol implementation, if any, is used for a particular URL load request. This is the default policy for URL load requests.

Available in Mac OS X v10.2 and later.

Declared in `NSURLRequest.h`.

`NSURLRequestReloadIgnoringLocalCacheData`

Specifies that the data for the URL load should be loaded from the originating source. No existing cache data should be used to satisfy a URL load request.

Available in Mac OS X v10.5 and later.

Declared in `NSURLRequest.h`.

`NSURLRequestReloadIgnoringLocalAndRemoteCacheData`

Specifies that not only should the local cache data be ignored, but that proxies and other intermediates should be instructed to disregard their caches so far as the protocol allows.

Available in Mac OS X v10.5 and later.

Declared in `NSURLRequest.h`.

`NSURLRequestReloadIgnoringCacheData`

Replaced by `NSURLRequestReloadIgnoringLocalCacheData` (page 1833).

Available in Mac OS X v10.2 and later.

Declared in `NSURLRequest.h`.

`NSURLRequestReturnCacheDataElseLoad`

Specifies that the existing cached data should be used to satisfy the request, regardless of its age or expiration date. If there is no existing data in the cache corresponding the request, the data is loaded from the originating source.

Available in Mac OS X v10.2 and later.

Declared in `NSURLRequest.h`.

`NSURLRequestReturnCacheDataDontLoad`

Specifies that the existing cache data should be used to satisfy a request, regardless of its age or expiration date. If there is no existing data in the cache corresponding to a URL load request, no attempt is made to load the data from the originating source, and the load is considered to have failed. This constant specifies a behavior that is similar to an "offline" mode.

Available in Mac OS X v10.2 and later.

Declared in `NSURLRequest.h`.

`NSURLRequestReloadRevalidatingCacheData`

Specifies that the existing cache data may be used provided the origin source confirms its validity, otherwise the URL is loaded from the origin source.

Available in Mac OS X v10.5 and later.

Declared in `NSURLRequest.h`.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`NSURLRequest.h`

NSURLResponse Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSURLResponse.h
Availability	Available in Mac OS X v10.2 with Safari 1.0 installed. Available in Mac OS X v10.2.7 and later.
Companion guide	URL Loading System
Related sample code	URL CacheInfo

Overview

`NSURLResponse` declares the programmatic interface for an object that accesses the response returned by an `NSURLRequest` instance.

`NSURLResponse` encapsulates the metadata associated with a URL load in a manner independent of protocol and URL scheme.

`NSHTTPURLResponse` is a subclass of `NSURLResponse` that provides methods for accessing information specific to HTTP protocol responses. An `NSHTTPURLResponse` object represents a response to an HTTP URL load request.

Note: `NSURLResponse` objects do not contain the actual bytes representing the content of a URL. See `NSURLConnection` for more information about receiving the content data for a URL load.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 2034)
- `initWithCoder:` (page 2034)

NSCopying

`copyWithZone:` (page 2042)

Tasks

Creating a Response

- `initWithURL:MIMEType:expectedContentLength:textEncodingName:` (page 1837)
  Returns an initialized `NSURLResponse` object with the URL, MIME type, length, and text encoding set to given values.

Getting the Response Properties

- `expectedContentLength` (page 1836)
  Returns the receiver's expected content length
- `suggestedFilename` (page 1838)
  Returns a suggested filename for the response data.
- `MIMEType` (page 1837)
  Returns the receiver's MIME type.
- `textEncodingName` (page 1838)
  Returns the name of the receiver's text encoding provided by the response's originating source.
- `URL` (page 1839)
  Returns the receiver's URL.

Instance Methods

expectedContentLength

Returns the receiver's expected content length

```
- (long long)expectedContentLength
```

Return Value
The receiver's expected content length, or `NSURLResponseUnknownLength` if the length can't be determined.

Discussion
Some protocol implementations report the content length as part of the response, but not all protocols guarantee to deliver that amount of data. Clients should be prepared to deal with more or less data.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLResponse.h`

initWithURL:MIMEType:expectedContentLength:textEncodingName:

Returns an initialized `NSURLResponse` object with the URL, MIME type, length, and text encoding set to given values.

```
- (id)initWithURL:(NSURL *)URL MIMEType:(NSString *)MIMEType
    expectedContentLength:(NSInteger)length textEncodingName:(NSString *)name
```

Parameters

URL

The URL for the new object.

MIMEType

The MIME type.

length

The expected content length.This value should be `-1` if the expected length is undetermined

name

The text encoding name. This value may be `nil`.

Return Value
An initialized `NSURLResponse` object with the URL set to *URL*, the MIME type set to *MIMEType*, length set to *length*, and text encoding name set to *name*.

Discussion
This is the designated initializer for `NSURLResponse`.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLResponse.h`

MIMEType

Returns the receiver's MIME type.

```
- (NSString *)MIMEType
```

Return Value
The receiver's MIME type.

Discussion
The MIME type is often provided by the response's originating source. However, that value may be changed or corrected by a protocol implementation if it can be determined that the response's source reported the information incorrectly.

If the response's originating source does not provide a MIME type, an attempt to guess the MIME type may be made.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSURLResponse.h

suggestedFilename

Returns a suggested filename for the response data.

```
- (NSString *)suggestedFilename
```

Return Value
A suggested filename for the response data.

Discussion
The method tries to create a filename using the following, in order:

1. A filename specified using the content disposition header.
2. The last path component of the URL.
3. The host of the URL.

If the host of URL can't be converted to a valid filename, the filename “unknown” is used.

In most cases, this method appends the proper file extension based on the MIME type. This method will always return a valid filename regardless of whether or not the resource is saved to disk.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSURLResponse.h

textEncodingName

Returns the name of the receiver’s text encoding provided by the response’s originating source.

```
- (NSString *)textEncodingName
```

Return Value
The name of the receiver’s text encoding provided by the response’s originating source, or `nil` if no text encoding was provided by the protocol

Discussion
Clients can convert this string to an `NSStringEncoding` or a `CFStringEncoding` using the methods and functions available in the appropriate framework.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSURLResponse.h

URL

Returns the receiver's URL.

```
- (NSURL *)URL
```

Return Value
The receiver's URL.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLResponse.h`

Constants

Response Length Unknown Error

The following error code is returned by `expectedContentLength` (page 1836).

```
#define NSURLResponseUnknownLength ((long long)-1)
```

Constants
`NSURLResponseUnknownLength`
Returned when the response length cannot be determined in advance of receiving the data from the server. For example, `NSURLResponseUnknownLength` is returned when the server HTTP response does not include a Content-Length header.

Available in Mac OS X v10.2 and later.

Declared in `NSURLResponse.h`.

NSUserDefaults Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSUserDefaults.h
Companion guide	User Defaults Programming Topics for Cocoa
Related sample code	Dicey Quartz Composer QCTV Quartz Composer WWDC 2005 TextEdit Sproing TextEditPlus

Overview

The `NSUserDefaults` class provides a programmatic interface for interacting with the defaults system. The defaults system allows an application to customize its behavior to match a user's preferences. For example, you can allow users to determine what units of measurement your application displays or how often documents are automatically saved. Applications record such preferences by assigning values to a set of parameters in a user's defaults database. The parameters are referred to as defaults since they're commonly used to determine an application's default state at startup or the way it acts by default.

At runtime, you use an `NSUserDefaults` object to read the defaults that your application uses from a user's defaults database. `NSUserDefaults` caches the information to avoid having to open the user's defaults database each time you need a default value. The `synchronize` (page 1861) method, which is automatically invoked at periodic intervals, keeps the in-memory cache in sync with a user's defaults database.

A default's value must be a property list, that is, an instance of (or for collections a combination of instances of): `NSData`, `NSString`, `NSNumber`, `NSDate`, `NSArray`, or `NSDictionary`. If you want to store any other type of object, you should typically archive it to create an instance of `NSData`. For more details, see *User Defaults Programming Topics for Cocoa*.

Values returned from `NSUserDefaults` are *immutable*, even if you set a mutable object as the value. For example, if you set a mutable string as the value for `"MyStringDefault"`, the string you later retrieve using `stringForKey:` (page 1860) will be immutable.

A defaults database is created automatically for each user. The `NSUserDefaults` class does not currently support per-host preferences. To do this, you must use the CFPreferences API (see *Preferences Utilities Reference*). However, `NSUserDefaults` correctly reads per-host preferences, so you can safely mix CFPreferences code with `NSUserDefaults` code.

If your application supports managed environments, you can use an `NSUserDefaults` object to determine which preferences are managed by an administrator for the benefit of the user. Managed environments correspond to computer labs or classrooms where an administrator or teacher may want to configure the systems in a particular way. In these situations, the teacher can establish a set of default preferences and force those preferences on users. If a preference is managed in this manner, applications should prevent users from editing that preference by disabling any appropriate controls.

The `NSUserDefaults` class is thread-safe.

Tasks

Getting the Shared NSUserDefaults Instance

`+ standardUserDefaults` (page 1845)
Returns the shared defaults object.

`+ resetStandardUserDefaults` (page 1844)
Synchronizes any changes made to the shared user defaults object and releases it from memory.

Initializing an NSUserDefaults Object

`- init` (page 1850)
Returns an `NSUserDefaults` object initialized with the defaults for the current user account.

`- initWithUser:` (page 1850)
Returns an `NSUserDefaults` object initialized with the defaults for the specified user account.

Getting a Default Value

`- arrayForKey:` (page 1846)
Returns the array associated with the specified key.

`- boolForKey:` (page 1847)
Returns the Boolean value associated with the specified key.

`- dataForKey:` (page 1847)
Returns the data object associated with the specified key.

`- dictionaryForKey:` (page 1848)
Returns the dictionary object associated with the specified key.

`- floatForKey:` (page 1849)
Returns the floating-point value associated with the specified key.

`- integerForKey:` (page 1851)
Returns the integer value associated with the specified key..

- `objectForKey:` (page 1851)
  Returns the object associated with the first occurrence of the specified default.
- `stringArrayForKey:` (page 1859)
  Returns the array of strings associated with the specified key.
- `stringForKey:` (page 1860)
  Returns the string associated with the specified key.

Setting and Removing Defaults

- `removeObjectForKey:` (page 1855)
  Removes the value of the specified default key in the standard application domain.
- `setBool:forKey:` (page 1856)
  Sets the value of the specified default key to a string containing a Boolean value.
- `setFloat:forKey:` (page 1857)
  Sets the value of the specified default key to a string containing a floating-point value.
- `setInteger:forKey:` (page 1857)
  Sets the value of the specified default key to a string containing an integer value.
- `setObject:forKey:` (page 1858)
  Sets the value of the specified default key in the standard application domain.

Registering Defaults

- `registerDefaults:` (page 1854)
  Adds the contents the specified dictionary to the registration domain.

Maintaining Persistent Domains

- `synchronize` (page 1861)
  Writes any modifications to the persistent domains to disk and updates all unmodified persistent domains to what is on disk.
- `persistentDomainForName:` (page 1853)
  Returns a dictionary containing the keys and values in the specified persistent domain.
- `persistentDomainNames` (page 1854)
  Returns an array of the current persistent domain names.
- `removePersistentDomainForName:` (page 1855)
  Removes the contents of the specified persistent domain from the user's defaults.
- `setPersistentDomain:forName:` (page 1858)
  Sets the dictionary for the specified persistent domain.

Accessing Managed Environment Keys

- `objectIsForcedForKey:` (page 1852)
  Returns a Boolean value indicating whether the specified key is managed by an administrator.

- `objectIsForcedForKey:inDomain:` (page 1853)
  Returns a Boolean value indicating whether the key in the specified domain is managed by an administrator.

Managing the Search List

- `dictionaryRepresentation` (page 1849)
  Returns a dictionary that contains a union of all key-value pairs in the domains in the search list.

Maintaining Volatile Domains

- `removeVolatileDomainForName:` (page 1856)
  Removes the specified volatile domain from the user's defaults.
- `setVolatileDomain:forName:` (page 1859)
  Sets the dictionary for the specified volatile domain.
- `volatileDomainForName:` (page 1861)
  Returns the dictionary for the specified volatile domain.
- `volatileDomainNames` (page 1862)
  Returns an array of the current volatile domain names.

Maintaining Suites

- `addSuiteNamed:` (page 1845)
  Inserts the specified domain name into the receiver's search list.
- `removeSuiteNamed:` (page 1855)
  Removes the specified domain name from the receiver's search list.

Class Methods

resetStandardUserDefaults

Synchronizes any changes made to the shared user defaults object and releases it from memory.

```
+ (void)resetStandardUserDefaults
```

Discussion
A subsequent invocation of `standardUserDefaults` (page 1845) creates a new shared user defaults object with the standard search list.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSUserDefaults.h`

standardUserDefaults

Returns the shared defaults object.

```
+ (NSUserDefaults *)standardUserDefaults
```

Return Value
The shared defaults object.

Discussion
If the shared defaults object does not exist yet, it is created with a search list containing the names of the following domains, in this order:

- `NSArgumentDomain`, consisting of defaults parsed from the application's arguments
- A domain identified by the application's bundle identifier
- `NSGlobalDomain`, consisting of defaults meant to be seen by all applications
- Separate domains for each of the user's preferred languages
- `NSRegistrationDomain`, a set of temporary defaults whose values can be set by the application to ensure that searches will always be successful

The defaults are initialized for the current user. Subsequent modifications to the standard search list remain in effect even when this method is invoked again—the search list is guaranteed to be standard only the first time this method is invoked. The shared instance is provided as a convenience—you can create custom instances using `alloc` along with `initWithUser:` (page 1850) or `init` (page 1850).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Dicey
IKSlideshowDemo
Quartz Composer QCTV
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSUserDefaults.h`

Instance Methods

addSuiteNamed:

Inserts the specified domain name into the receiver's search list.

```
- (void)addSuiteNamed:(NSString *)suiteName
```

Parameters

suiteName
The domain name to insert. This domain is inserted after the application domain.

Discussion
The *suiteName* domain is similar to a bundle identifier string, but is not tied to a particular application or bundle. A suite can be used to hold preferences that are shared between multiple applications.

Availability
Available in Mac OS X v10.0 and later.

See Also
+ standardUserDefaults (page 1845)
- removeSuiteNamed: (page 1855)

Declared In
NSUserDefaults.h

arrayForKey:

Returns the array associated with the specified key.

```
- (NSArray *)arrayForKey:(NSString *)defaultName
```

Parameters
defaultName
A key in the current user's defaults database.

Return Value
The array associated with the specified key, or nil if the key does not exist or its value is not an NSArray object.

Special Considerations
The returned array and its contents are immutable, even if the values you originally set were mutable.

Availability
Available in Mac OS X v10.0 and later.

See Also
- boolForKey: (page 1847)
- dataForKey: (page 1847)
- dictionaryForKey: (page 1848)
- floatForKey: (page 1849)
- integerForKey: (page 1851)
- objectForKey: (page 1851)
- stringArrayForKey: (page 1859)
- stringForKey: (page 1860)

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
NSUserDefaults.h

boolForKey:

Returns the Boolean value associated with the specified key.

```
- (BOOL)boolForKey:(NSString *)defaultName
```

Parameters

defaultName

A key in the current user's defaults database.

Return Value

If a boolean value is associated with *defaultName* in the user defaults, that value is returned. Otherwise, `NO` is returned.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `arrayForKey:` (page 1846)
- `dataForKey:` (page 1847)
- `dictionaryForKey:` (page 1848)
- `floatForKey:` (page 1849)
- `integerForKey:` (page 1851)
- `objectForKey:` (page 1851)
- `stringArrayForKey:` (page 1859)
- `stringForKey:` (page 1860)

Related Sample Code

Sproing

Declared In

`NSUserDefaults.h`

dataForKey:

Returns the data object associated with the specified key.

```
- (NSData *)dataForKey:(NSString *)defaultName
```

Parameters

defaultName

A key in the current user's defaults database.

Return Value

The data object associated with the specified key, or `nil` if the key does not exist or its value is not an `NSData` object.

Special Considerations

The returned data object is immutable, even if the value you originally set was a mutable data object.

Availability

Available in Mac OS X v10.0 and later.

See Also
- `arrayForKey:` (page 1846)
- `boolForKey:` (page 1847)
- `dictionaryForKey:` (page 1848)
- `floatForKey:` (page 1849)
- `integerForKey:` (page 1851)
- `objectForKey:` (page 1851)
- `stringArrayForKey:` (page 1859)
- `stringForKey:` (page 1860)

Related Sample Code
QTQuartzPlayer

Declared In
`NSUserDefaults.h`

dictionaryForKey:

Returns the dictionary object associated with the specified key.

```
- (NSDictionary *)dictionaryForKey:(NSString *)defaultName
```

Parameters

defaultName
A key in the current user's defaults database.

Return Value
The dictionary object associated with the specified key, or `nil` if the key does not exist or its value is not an `NSDictionary` object.

Special Considerations

The returned dictionary and its contents are immutable, even if the values you originally set were mutable.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `arrayForKey:` (page 1846)
- `boolForKey:` (page 1847)
- `dataForKey:` (page 1847)
- `floatForKey:` (page 1849)
- `integerForKey:` (page 1851)
- `objectForKey:` (page 1851)
- `stringArrayForKey:` (page 1859)
- `stringForKey:` (page 1860)

Declared In
`NSUserDefaults.h`

dictionaryRepresentation

Returns a dictionary that contains a union of all key-value pairs in the domains in the search list.

```
- (NSDictionary *)dictionaryRepresentation
```

Return Value
A dictionary containing the keys. The keys are names of defaults and the value corresponding to each key is a property list object (`NSData`, `NSString`, `NSNumber`, `NSDate`, `NSArray`, or `NSDictionary`).

Discussion
As with `objectForKey:` (page 1851), key-value pairs in domains that are earlier in the search list take precedence. The combined result does not preserve information about which domain each entry came from.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
NewsReader

Declared In
`NSUserDefaults.h`

floatForKey:

Returns the floating-point value associated with the specified key.

```
- (float)floatForKey:(NSString *)defaultName
```

Parameters
defaultName
A key in the current user's defaults database.

Return Value
The floating-point value associated with the specified key. If the key does not exist, this method returns `0`.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `arrayForKey:` (page 1846)
- `boolForKey:` (page 1847)
- `dataForKey:` (page 1847)
- `dictionaryForKey:` (page 1848)
- `integerForKey:` (page 1851)
- `objectForKey:` (page 1851)
- `stringArrayForKey:` (page 1859)
- `stringForKey:` (page 1860)

Related Sample Code
MungSaver

Declared In
`NSUserDefaults.h`

init

Returns an `NSUserDefaults` object initialized with the defaults for the current user account.

```
- (id)init
```

Return Value
An initialized `NSUserDefaults` object whose argument and registration domains are already set up.

Discussion
This method does not put anything in the search list. Invoke it only if you've allocated your own `NSUserDefaults` instance instead of using the shared one.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ standardUserDefaults` (page 1845)

Declared In
`NSUserDefaults.h`

initWithUser:

Returns an `NSUserDefaults` object initialized with the defaults for the specified user account.

```
- (id)initWithUser:(NSString *)username
```

Parameters

username
The name of the user account.

Return Value
An initialized `NSUserDefaults` object whose argument and registration domains are already set up. If the current user does not have access to the specified user account, this method returns `nil`.

Discussion
This method does not put anything in the search list. Invoke it only if you've allocated your own `NSUserDefaults` instance instead of using the shared one.

You do not normally use this method to initialize an instance of `NSUserDefaults`. Applications used by a superuser might use this method to update the defaults databases for a number of users. The user who started the application must have appropriate access (read, write, or both) to the defaults database of the new user, or this method returns `nil`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ standardUserDefaults` (page 1845)

Declared In
`NSUserDefaults.h`

integerForKey:

Returns the integer value associated with the specified key..

```
- (NSInteger)integerForKey:(NSString *)defaultName
```

Parameters

defaultName

A key in the current user's defaults database.

Return Value

The integer value associated with the specified key. If the specified key does not exist, this method returns `0`.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `arrayForKey:` (page 1846)
- `boolForKey:` (page 1847)
- `dataForKey:` (page 1847)
- `dictionaryForKey:` (page 1848)
- `floatForKey:` (page 1849)
- `objectForKey:` (page 1851)
- `stringArrayForKey:` (page 1859)
- `stringForKey:` (page 1860)

Related Sample Code

IKSlideshowDemo

MungSaver

NumberInput_IMKit_Sample

Sproing

Declared In

`NSUserDefaults.h`

objectForKey:

Returns the object associated with the first occurrence of the specified default.

```
- (id)objectForKey:(NSString *)defaultName
```

Parameters

defaultName

A key in the current user's defaults database.

Return Value

The object associated with the specified key, or `nil` if the key was not found.

Discussion

This method searches the domains included in the search list in the order they are listed.

Special Considerations

The returned object is immutable, even if the value you originally set was mutable.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `- arrayForKey:` (page 1846)
- `- boolForKey:` (page 1847)
- `- dataForKey:` (page 1847)
- `- dictionaryForKey:` (page 1848)
- `- floatForKey:` (page 1849)
- `- stringArrayForKey:` (page 1859)
- `- stringForKey:` (page 1860)

Related Sample Code

PrefsPane
QTKitPlayer
Quartz Composer QCTV
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In

`NSUserDefaults.h`

objectIsForcedForKey:

Returns a Boolean value indicating whether the specified key is managed by an administrator.

```
- (BOOL)objectIsForcedForKey:(NSString *)key
```

Parameters

key

The key whose status you want to check.

Return Value

`YES` if the value of the specified key is managed by an administrator, otherwise `NO`.

Discussion

This method assumes that the key is a preference associated with the current user and application. For managed keys, the application should disable any user interface that allows the user to modify the value of *key*.

Availability

Available in Mac OS X v10.2 and later.

See Also

- `- objectIsForcedForKey:inDomain:` (page 1853)

Declared In

`NSUserDefaults.h`

objectIsForcedForKey:inDomain:

Returns a Boolean value indicating whether the key in the specified domain is managed by an administrator.

```
- (BOOL)objectIsForcedForKey:(NSString *)key inDomain:(NSString *)domain
```

Parameters

key

The key whose status you want to check.

domain

The domain of the key.

Return Value

`YES` if the key is managed by an administrator in the specified domain, otherwise `NO`.

Discussion

This method assumes that the key is a preference associated with the current user. For managed keys, the application should disable any user interface that allows the user to modify the value of *key*.

Availability

Available in Mac OS X v10.2 and later.

See Also

`- objectIsForcedForKey:` (page 1852)

Declared In

`NSUserDefaults.h`

persistentDomainForName:

Returns a dictionary containing the keys and values in the specified persistent domain.

```
- (NSDictionary *)persistentDomainForName:(NSString *)domainName
```

Parameters

domainName

The domain whose keys and values you want. This value should be equal to your application's bundle identifier.

Return Value

A dictionary containing the keys. The keys are names of defaults and the value corresponding to each key is a property list object (`NSData`, `NSString`, `NSNumber`, `NSDate`, `NSArray`, or `NSDictionary`).

Availability

Available in Mac OS X v10.0 and later.

See Also

`- removePersistentDomainForName:` (page 1855)

`- setPersistentDomain:forName:` (page 1858)

Declared In

`NSUserDefaults.h`

persistentDomainNames

Returns an array of the current persistent domain names.

```
- (NSArray *)persistentDomainNames
```

Return Value
An array of `NSString` objects containing the domain names.

Discussion
You can get the keys and values for each domain by passing the returned domain names to the `persistentDomainForName:` (page 1853) method.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `removePersistentDomainForName:` (page 1855)
- `setPersistentDomain:forName:` (page 1858)

Declared In
`NSUserDefaults.h`

registerDefaults:

Adds the contents the specified dictionary to the registration domain.

```
- (void)registerDefaults:(NSDictionary *)dictionary
```

Parameters
dictionary
The dictionary of keys and values you want to register.

Discussion
If there is no registration domain, one is created using the specified dictionary, and `NSRegistrationDomain` is added to the end of the search list.

The contents of the registration domain are not written to disk; you need to call this method each time your application starts. You can place a plist file in the application's Resources directory and call `registerDefaults:` with the contents that you read in from that file.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
DeskPictAppDockMenu
Dicey
Sproing
TemperatureTester

Declared In
`NSUserDefaults.h`

removeObjectForKey:

Removes the value of the specified default key in the standard application domain.

```
- (void)removeObjectForKey:(NSString *)defaultName
```

Parameters

defaultName

The key whose value you want to remove.

Discussion

Removing a default has no effect on the value returned by the `objectForKey:` (page 1851) method if the same key exists in a domain that precedes the standard application domain in the search list.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- setObject:forKey:` (page 1858)

Related Sample Code

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In

`NSUserDefaults.h`

removePersistentDomainForName:

Removes the contents of the specified persistent domain from the user's defaults.

```
- (void)removePersistentDomainForName:(NSString *)domainName
```

Parameters

domainName

The domain whose keys and values you want. This value should be equal to your application's bundle identifier.

Discussion

When a persistent domain is changed, an `NSUserDefaultsDidChangeNotification` (page 1870) is posted.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- setPersistentDomain:forName:` (page 1858)

Declared In

`NSUserDefaults.h`

removeSuiteNamed:

Removes the specified domain name from the receiver's search list.

```
- (void)removeSuiteNamed:(NSString *)suiteName
```

Parameters

suiteName

The domain name to remove.

Availability
Available in Mac OS X v10.0 and later.

See Also
- addSuiteNamed: (page 1845)

Declared In
NSUserDefaults.h

removeVolatileDomainForName:

Removes the specified volatile domain from the user's defaults.

```
- (void)removeVolatileDomainForName:(NSString *)domainName
```

Parameters

domainName

The volatile domain you want to remove.

Availability
Available in Mac OS X v10.0 and later.

See Also
- setVolatileDomain:forName: (page 1859)

Declared In
NSUserDefaults.h

setBool:forKey:

Sets the value of the specified default key to a string containing a Boolean value.

```
- (void)setBool:(BOOL)value forKey:(NSString *)defaultName
```

Parameters

value

The Boolean value to store in the defaults database.

defaultName

The key with which to associate with the value.

Discussion
Invokes setObject:forKey: (page 1858) as part of its implementation.

Availability
Available in Mac OS X v10.0 and later.

See Also
- boolForKey: (page 1847)

Related Sample Code
Sproing

Declared In
NSUserDefaults.h

setFloat:forKey:

Sets the value of the specified default key to a string containing a floating-point value.

```
- (void)setFloat:(float)value forKey:(NSString *)defaultName
```

Parameters

value
The floating-point value to store in the defaults database.

defaultName
The key with which to associate with the value.

Discussion
Invokes `setObject:forKey:` (page 1858) as part of its implementation.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- floatForKey:` (page 1849)

Declared In
NSUserDefaults.h

setInteger:forKey:

Sets the value of the specified default key to a string containing an integer value.

```
- (void)setInteger:(NSInteger)value forKey:(NSString *)defaultName
```

Parameters

value
The integer value to store in the defaults database.

defaultName
The key with which to associate with the value.

Discussion
Invokes `setObject:forKey:` (page 1858) as part of its implementation.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- integerForKey:` (page 1851)

Related Sample Code
IKSlideshowDemo

MungSaver
Sproing

Declared In
NSUserDefaults.h

setObject:forKey:

Sets the value of the specified default key in the standard application domain.

```
- (void)setObject:(id)value forKey:(NSString *)defaultName
```

Parameters

value

The object to store in the defaults database. A default's value can be only property list objects: NSData, NSString, NSNumber, NSDate, NSArray, or NSDictionary.

defaultName

The key with which to associate with the value.

Discussion
Setting a default has no effect on the value returned by the `objectForKey:` (page 1851) method if the same key exists in a domain that precedes the application domain in the search list.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- removeObjectForKey:` (page 1855)

Related Sample Code
CIVideoDemoGL
PrefsPane
QTAudioExtractionPanel
QTKitPlayer
Quartz Composer QCTV

Declared In
NSUserDefaults.h

setPersistentDomain:forName:

Sets the dictionary for the specified persistent domain.

```
- (void)setPersistentDomain:(NSDictionary *)domain forName:(NSString *)domainName
```

Parameters

domain

The dictionary of keys and values you want to assign to the domain.

domainName

The domain whose keys and values you want to set. This value should be equal to your application's bundle identifier.

Discussion
When a persistent domain is changed, an `NSUserDefaultsDidChangeNotification` (page 1870) is posted.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `persistentDomainForName:` (page 1853)
- `persistentDomainNames` (page 1854)

Declared In
`NSUserDefaults.h`

setVolatileDomain:forName:

Sets the dictionary for the specified volatile domain.

```
- (void)setVolatileDomain:(NSDictionary *)domain forName:(NSString *)domainName
```

Parameters

domain
The dictionary of keys and values you want to assign to the domain.

domainName
The domain whose keys and values you want to set.

Discussion
This method raises an `NSInvalidArgumentException` if a volatile domain with the specified name already exists.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `volatileDomainForName:` (page 1861)
- `volatileDomainNames` (page 1862)

Declared In
`NSUserDefaults.h`

stringArrayForKey:

Returns the array of strings associated with the specified key.

```
- (NSArray *)stringArrayForKey:(NSString *)defaultName
```

Parameters

defaultName
A key in the current user's defaults database.

Return Value
The array of `NSString` objects, or `nil` if the specified default does not exist, the default does not contain an array, or the array does not contain `NSString` objects.

Special Considerations

The returned array and its contents are immutable, even if the values you originally set were mutable.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `arrayForKey:` (page 1846)
- `boolForKey:` (page 1847)
- `dataForKey:` (page 1847)
- `dictionaryForKey:` (page 1848)
- `floatForKey:` (page 1849)
- `integerForKey:` (page 1851)
- `objectForKey:` (page 1851)
- `stringForKey:` (page 1860)

Declared In

`NSUserDefaults.h`

stringForKey:

Returns the string associated with the specified key.

```
- (NSString *)stringForKey:(NSString *)defaultName
```

Parameters

defaultName

A key in the current user's defaults database.

Return Value

The string associated with the specified key, or `nil` if the default does not exist or does not contain a string.

Special Considerations

The returned string is immutable, even if the value you originally set was a mutable string.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `arrayForKey:` (page 1846)
- `boolForKey:` (page 1847)
- `dataForKey:` (page 1847)
- `dictionaryForKey:` (page 1848)
- `floatForKey:` (page 1849)
- `integerForKey:` (page 1851)
- `objectForKey:` (page 1851)
- `stringArrayForKey:` (page 1859)

Related Sample Code

CIVideoDemoGL

Core Animation QuickTime Layer

DeskPictAppDockMenu

Declared In
NSUserDefaults.h

synchronize

Writes any modifications to the persistent domains to disk and updates all unmodified persistent domains to what is on disk.

```
- (BOOL)synchronize
```

Return Value
YES if the data was saved successfully to disk, otherwise NO.

Discussion
Because this method is automatically invoked at periodic intervals, use this method only if you cannot wait for the automatic synchronization (for example, if your application is about to exit) or if you want to update the user defaults to what is on disk even though you have not made any changes.

Availability
Available in Mac OS X v10.0 and later.

See Also
- persistentDomainForName: (page 1853)
- persistentDomainNames (page 1854)
- removePersistentDomainForName: (page 1855)
- setPersistentDomain:forName: (page 1858)

Related Sample Code
CIVideoDemoGL
MungSaver
QTAudioExtractionPanel
QTKitPlayer
Quartz Composer QCTV

Declared In
NSUserDefaults.h

volatileDomainForName:

Returns the dictionary for the specified volatile domain.

```
- (NSDictionary *)volatileDomainForName:(NSString *)domainName
```

Parameters
domainName
The domain whose keys and values you want.

Return Value
The dictionary of keys and values belonging to the domain. The keys in the dictionary are names of defaults, and the value corresponding to each key is a property list object (`NSData`, `NSString`, `NSNumber`, `NSDate`, `NSArray`, or `NSDictionary`).

Availability
Available in Mac OS X v10.0 and later.

See Also
`- removeVolatileDomainForName:` (page 1856)
`- setVolatileDomain:forName:` (page 1859)

Declared In
`NSUserDefaults.h`

volatileDomainNames

Returns an array of the current volatile domain names.

```
- (NSArray *)volatileDomainNames
```

Return Value
An array of `NSString` objects with the volatile domain names.

Discussion
You can get the contents of each domain by passing the returned domain names to the `volatileDomainForName:` (page 1861) method.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- removeVolatileDomainForName:` (page 1856)
`- setVolatileDomain:forName:` (page 1859)

Declared In
`NSUserDefaults.h`

Constants

NSUserDefaults Domains

These constants specify various user defaults domains.

```
extern NSString *NSGlobalDomain;
extern NSString *NSArgumentDomain;
extern NSString *NSRegistrationDomain;
```

Constants

`NSGlobalDomain`

The domain consisting of defaults meant to be seen by all applications.

Available in Mac OS X v10.0 and later.

Declared in `NSUserDefaults.h`.

`NSArgumentDomain`

The domain consisting of defaults parsed from the application's arguments. These are one or more pairs of the form *-default value* included in the command-line invocation of the application.

Available in Mac OS X v10.0 and later.

Declared in `NSUserDefaults.h`.

`NSRegistrationDomain`

The domain consisting of a set of temporary defaults whose values can be set by the application to ensure that searches will always be successful.

Available in Mac OS X v10.0 and later.

Declared in `NSUserDefaults.h`.

Declared In

`NSUserDefaults.h`

Language-Dependent Date/Time Information

The `NSUserDefaults` class provides the following constants as a convenience. They provide access to values of the keys to the locale dictionary, which is discussed in *User Defaults Programming Topics for Cocoa*. (**Deprecated.** These constants are deprecated in Mac OS X v10.5. Where there are direct replacements, you can find typically them in `NSDateFormatter`—for example, `monthSymbols` (page 437), `shortWeekdaySymbols` (page 454), and `AMSymbol` (page 430)—otherwise you should use the patterns described in *Data Formatting Programming Guide for Cocoa*.)

```
extern NSString *NSAMPMDesignation;
extern NSString *NSDateFormatString;
extern NSString *NSDateTimeOrdering;
extern NSString *NSEarlierTimeDesignations;
extern NSString *NSHourNameDesignations;
extern NSString *NSLaterTimeDesignations;
extern NSString *NSMonthNameArray;
extern NSString *NSNextDayDesignations;
extern NSString *NSNextNextDayDesignations;
extern NSString *NSPriorDayDesignations;
extern NSString *NSShortDateFormatString;
extern NSString *NSShortMonthNameArray;
extern NSString *NSShortTimeDateFormatString;
extern NSString *NSShortWeekDayNameArray;
extern NSString *NSThisDayDesignations;
extern NSString *NSTimeDateFormatString;
extern NSString *NSTimeFormatString;
extern NSString *NSWeekDayNameArray;
extern NSString *NSYearMonthWeekDesignations;
```

Constants

`NSAMPMDesignation`

Key for the value that specifies how the morning and afternoon designations are printed, affecting strings that use the `%p` format specifier. (**Deprecated.** Use `AMSymbol` (page 430) or `PMSymbol` (page 437) (`NSDateFormatter`) instead.)

The defaults are “AM” and “PM”.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSDateFormatString`

Key for the format string that specifies how how dates are printed using the date format specifiers. (**Deprecated.** Use the appropriate API from `NSDateFormatter` instead—see *Data Formatting Programming Guide for Cocoa*.)

The default is to use weekday names with full month names and full years, as in “Saturday, March 24, 2001.”

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSDateTimeOrdering`

Key for the string that specifies how to use ambiguous numbers in date strings.

Specify this value as a permutation of the letters M (month), D (day), Y (year), and H (hour). For example, MDYH treats “2/3/01 10” as the 3rd day of February 2001 at 10:00 am, whereas DMYH treats the same value as the 2nd day of March 2001 at 10:00 am. If fewer numbers are specified than are needed, the numbers are prioritized to satisfy day first, then month, and then year. For example, if you supply only the value 12, it means the 12th day of this month in this year because the day must be specified. If you supply “2 12” it means either February 12 or December 2, depending on if the ordering is “MDYH” or “DMYH.”

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSEarlierTimeDesignations`

Key for an array of strings that denote a time in the past. (**Deprecated.** There is no direct replacement. If you need to localize words such as “prior,” you should use a strings file as you would for any other localizable text—see Strings Files.)

These are adjectives that modify values from `NSYearMonthWeekDesignations`. The defaults are “prior,” “last,” “past,” and “ago.”

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSHourNameDesignations`

Key for strings that identify the time of day.

These strings should be bound to an hour. The default is this array of arrays: (0, midnight), (10, morning), (12, noon, lunch), (14, afternoon), (19, dinner).

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSLaterTimeDesignations`

Key for an array of strings that denote a time in the future. (**Deprecated.** There is no direct replacement. If you need to localize words such as “next,” you should use a strings file as you would for any other localizable text—see Strings Files.)

Strings in this array are adjectives that modify a value from `NSYearMonthWeekDesignations`.

The default is an array that contains a single string, "next".

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSMonthNameArray`

Key for the value that specifies the names for the months, affecting strings that use the `%B` format specifier. (**Deprecated.** Use `monthSymbols` (page 437) or—if you are going to display these in the user interface by themselves—`standaloneMonthSymbols` (page 455) (`NSDateFormatter`) instead.)

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSNextDayDesignations`

Key for an array of strings that denote the day after today. (**Deprecated.** There is no direct replacement. If you need to localize words such as "tomorrow," you should use a strings file as you would for any other localizable text—see Strings Files.)

The default is an array that contains a single string, "tomorrow".

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSNextNextDayDesignations`

Key for an array of strings that denote the day after tomorrow. (**Deprecated.** There is no direct replacement. If you need to localize words such as "nextday," you should use a strings file as you would for any other localizable text—see Strings Files.)

The default is an array that contains a single string, "nextday".

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSPriorDayDesignations`

Key for an array of strings that denote the day before today. (**Deprecated.** There is no direct replacement. If you need to localize words such as "yesterday," you should use a strings file as you would for any other localizable text—see Strings Files.)

The default is an array that contains a single string, "yesterday".

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSShortDateFormatString`

Key for a format string that specifies how dates are abbreviated. (**Deprecated.** Use the appropriate API from `NSDateFormatter` instead—see *Data Formatting Programming Guide for Cocoa*.)

The default is to separate the day month and year with slashes and to put the day first, as in 31/10/01.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSShortWeekDayNameArray`

Key for an array of strings that specify the abbreviations for the days of the week, affecting strings that use the %a format specifier. (**Deprecated.** Use `shortWeekdaySymbols` (page 454) or—if you are going to display these in the user interface by themselves—`shortStandaloneWeekdaySymbols` (page 454) (`NSDateFormatter`) instead.)

Sunday should be the first day of the week.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSShortMonthNameArray`

Key for an array of strings that specify the abbreviations for the months, affecting strings that use the `%b` format specifier. (**Deprecated.** Use `shortMonthSymbols` (page 452)or—if you are going to display these in the user interface by themselves—`shortStandaloneMonthSymbols` (page 453) (`NSDateFormatter`) instead.)

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSShortTimeDateFormatString`

Key for a format string that specifies how times and dates are abbreviated. (**Deprecated.** Use the appropriate API from`NSDateFormatter` instead—see *Data Formatting Programming Guide for Cocoa*.)

The default is to use dashes to separate the day, month, and year and to use a 12-hour clock, as in "31-Jan-01 1:30 PM."]

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSThisDayDesignations`

Key for an array of strings that specify what this day is called. (**Deprecated.** There is no direct replacement. If you need to localize words such as “today,” you should use a strings file as you would for any other localizable text—see Strings Files.)

The default is an array containing two strings, "today" and "now".

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSTimeDateFormatString`

Key for the value that specifies how dates with times are printed, affecting strings that use the format specifiers `%c`, `%X`, or `%x`. (**Deprecated.** Use the appropriate API from `NSDateFormatter` instead—see *Data Formatting Programming Guide for Cocoa*.)

The default is to use full month names and days with a 24-hour clock, as in "Sunday, January 01, 2001 23:00:00 Pacific Standard Time."

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSTimeFormatString`

Key for a format string that specifies how dates with times are printed. (**Deprecated.** Use the appropriate API from `NSDateFormatter` instead—see *Data Formatting Programming Guide for Cocoa*.)

The default is to use a 12-hour clock.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSWeekDayNameArray`

Key for an array of strings that specify the names for the days of the week, affecting strings that use the `%A` format specifier. (**Deprecated.** Use `weekdaySymbols` (page 459) or—if you are going to display these in the user interface by themselves—`standaloneWeekdaySymbols` (page 455) (`NSDateFormatter`) instead.)

Sunday should be the first day of the week.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSYearMonthWeekDesignations`

Key for an array of strings that specify the words for year, month, and week in the current locale. (**Deprecated.** There is no direct replacement. If you need to localize words such as “year,” you should use a strings file as you would for any other localizable text—see Strings Files.)

The defaults are “year,” “month,” and “week.”

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

Declared In
`NSUserDefaults.h`

Language-Dependent Numeric Information

The `NSUserDefaults` class provides the following constants as a convenience. They provide access to values of the keys to the locale dictionary, which is discussed in *User Defaults Programming Topics for Cocoa*. (**Deprecated.** These constants are deprecated in Mac OS X v10.5. Where there are replacements, you can

typically find them in `NSNumberFormatter` or `NSLocale`—for example, `currencySymbol` (page 1092), `currencyDecimalSeparator` (page 1091), and `thousandSeparator` (page 1139)—otherwise you should use the patterns described in *Data Formatting Programming Guide for Cocoa*.)

```
extern NSString *NSCurrencySymbol;
extern NSString *NSDecimalDigits;
extern NSString *NSDecimalSeparator;
extern NSString *NSInternationalCurrencyString;
extern NSString *NSNegativeCurrencyFormatString;
extern NSString *NSPositiveCurrencyFormatString;
extern NSString *NSThousandsSeparator;
```

Constants

`NSCurrencySymbol`

A string that specifies the symbol used to denote currency in this language. (**Deprecated.** Use `currencySymbol` (page 1092) (`NSNumberFormatter`) or retrieve the `NSLocaleCurrencySymbol` from the current locale instead.)

The default is "$".

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSDecimalDigits`

Strings that identify the decimal digits in addition to or instead of the ASCII digits.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSDecimalSeparator`

A string that specifies the decimal separator. (**Deprecated.** Use `decimalSeparator` (page 1092) or `currencyDecimalSeparator` (page 1091) (`NSNumberFormatter`) or retrieve the `NSLocaleDecimalSeparator` from the current locale instead.)

The decimal separator separates the ones place from the tenths place. The default is ".".

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSInternationalCurrencyString`

A string containing a three-letter abbreviation for currency, following the ISO 4217 standard. (**Deprecated.** Retrieve the `NSLocaleCurrencySymbol` from the current locale instead.)

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSNegativeCurrencyFormatString`

A format string that specifies how negative numbers are printed when representing a currency value. (**Deprecated.** Use the appropriate API from `NSNumberFormatter` instead—see *Data Formatting Programming Guide for Cocoa*.)

The default is `-$9,999.00`.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSPositiveCurrencyFormatString`

A format string that specifies how positive numbers are printed when representing a currency value. (**Deprecated.** Use the appropriate API from `NSNumberFormatter` instead—see *Data Formatting Programming Guide for Cocoa*.)

The default is `$9,999.00`.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

`NSThousandsSeparator`

A string that specifies the separator character for the thousands place of a decimal number. (**Deprecated.** Retrieve the `NSLocaleGroupingSeparator` from the current locale instead.)

The default is a comma.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSUserDefaults.h`.

Declared In
`NSUserDefaults.h`

Notifications

NSUserDefaultsDidChangeNotification

This notification is posted when a change is made to defaults in a persistent domain.

The notification object is the `NSUserDefaults` object. This notification does not contain a *`userInfo`* dictionary.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSUserDefaults.h`

NSValue Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSValue.h Foundation/NSGeometry.h Foundation/NSRange.h
Companion guide	Number and Value Programming Topics for Cocoa
Related sample code	iSpend QTAudioExtractionPanel QTKitMovieShuffler Quartz Composer WWDC 2005 TextEdit TextEditPlus

Overview

An `NSValue` object is a simple container for a single C or Objective-C data item. It can hold any of the scalar types such as `int`, `float`, and `char`, as well as pointers, structures, and object `id`s. The purpose of this class is to allow items of such data types to be added to collections such as instances of `NSArray` and `NSSet`, which require their elements to be objects. `NSValue` objects are always immutable.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 2034)
- `initWithCoder:` (page 2034)

NSCopying
- `- copyWithZone:` (page 2042)

Tasks

Creating an NSValue

- `initWithBytes:objCType:` (page 1878)
  Initializes and returns an `NSValue` object that contains a given value, which is interpreted as being of a given Objective-C type.
+ `valueWithBytes:objCType:` (page 1873)
  Creates and returns an `NSValue` object that contains a given value, which is interpreted as being of a given Objective-C type.
+ `value:withObjCType:` (page 1873)
  Creates and returns an `NSValue` object that contains a given value which is interpreted as being of a given Objective-C type.
+ `valueWithNonretainedObject:` (page 1874)
  Creates and returns an `NSValue` object that contains a given object.
+ `valueWithPointer:` (page 1875)
  Creates and returns an `NSValue` object that contains a given pointer.
+ `valueWithPoint:` (page 1875)
  Creates and returns an `NSValue` object that contains a given `NSPoint` structure.
+ `valueWithRange:` (page 1876)
  Creates and returns an `NSValue` object that contains a given `NSRange` structure.
+ `valueWithRect:` (page 1876)
  Creates and returns an `NSValue` object that contains a given `NSRect` structure.
+ `valueWithSize:` (page 1877)
  Creates and returns an `NSValue` object that contains a given `NSSize` structure.

Accessing Data

- `getValue:` (page 1877)
  Copies the receiver's value into a given buffer.
- `nonretainedObjectValue` (page 1879)
  Returns the receiver's value as an `id`.
- `objCType` (page 1879)
  Returns a C string containing the Objective-C type of the data contained in the receiver.
- `pointValue` (page 1880)
  Returns an `NSPoint` structure representation of the receiver.
- `pointerValue` (page 1879)
  Returns the receiver's value as a pointer to `void`.
- `rangeValue` (page 1880)
  Returns an `NSRange` structure representation of the receiver.
- `rectValue` (page 1880)
  Returns an `NSRect` structure representation of the receiver.

- sizeValue (page 1881)
  Returns an NSSize structure representation of the receiver.

Comparing Objects

- isEqualToValue: (page 1878)
  Returns a Boolean value that indicates whether the receiver and another value are equal.

Class Methods

value:withObjCType:

Creates and returns an NSValue object that contains a given value which is interpreted as being of a given Objective-C type.

```
+ (NSValue *)value:(const void *)value withObjCType:(const char *)type
```

Parameters

value
  The value for the new NSValue object.

type
  The Objective-C type of *value*. *type* should be created with the Objective-C @encode() compiler directive; it should not be hard-coded as a C string.

Return Value
A new NSValue object that contains *value*, which is interpreted as being of the Objective-C type *type*.

Discussion
This method has the same effect as valueWithBytes:objCType: (page 1873) and may be deprecated in a future release. You should use valueWithBytes:objCType: (page 1873) instead.

Availability
Available in Mac OS X v10.0 and later.

See Also
+ valueWithBytes:objCType: (page 1873)

Related Sample Code
VideoViewer

Declared In
NSValue.h

valueWithBytes:objCType:

Creates and returns an NSValue object that contains a given value, which is interpreted as being of a given Objective-C type.

```
+ (NSValue *)valueWithBytes:(const void *)value objCType:(const char *)type
```

Parameters

value

The value for the new NSValue object.

type

The Objective-C type of *value*. *type* should be created with the Objective-C @encode() compiler directive; it should not be hard-coded as a C string.

Return Value
A new NSValue object that contains *value*, which is interpreted as being of the Objective-C type *type*.

Discussion
See *Number and Value Programming Topics for Cocoa* for other considerations in creating an NSValue object and code examples.

Availability
Available in Mac OS X v10.0 and later.

See Also
- initWithBytes:objCType: (page 1878)

Declared In
NSValue.h

valueWithNonretainedObject:

Creates and returns an NSValue object that contains a given object.

```
+ (NSValue *)valueWithNonretainedObject:(id)anObject
```

Parameters

anObject

The value for the new object.

Return Value
A new NSValue object that contains *anObject*.

Discussion
This method is equivalent to invoking value:withObjCType: (page 1873) in this manner:

```
NSValue *theValue = [NSValue value:&anObject withObjCType:@encode(void *)];
```

This method is useful for preventing an object from being retained when it's added to a collection object (such as an instance of NSArray or NSDictionary).

Availability
Available in Mac OS X v10.0 and later.

See Also
- nonretainedObjectValue (page 1879)

Declared In
NSValue.h

valueWithPoint:

Creates and returns an `NSValue` object that contains a given `NSPoint` structure.

```
+ (NSValue *)valueWithPoint:(NSPoint)aPoint
```

Parameters

aPoint

The value for the new object.

Return Value

A new `NSValue` object that contains the value of *point*.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- pointValue` (page 1880)

Related Sample Code

Aperture Edit Plugin - Borders & Titles
Dicey
ImageMapExample
PDF Annotation Editor
TrackBall

Declared In

`NSGeometry.h`

valueWithPointer:

Creates and returns an `NSValue` object that contains a given pointer.

```
+ (NSValue *)valueWithPointer:(const void *)aPointer
```

Parameters

aPointer

The value for the new object.

Return Value

A new `NSValue` object that contains *aPointer*.

Discussion

This method is equivalent to invoking `value:withObjCType:` (page 1873) in this manner:

```
NSValue *theValue = [NSValue value:&aPointer withObjCType:@encode(void *)];
```

This method does not copy the contents of *aPointer*, so you must not to deallocate the memory at the pointer destination while the `NSValue` object exists. `NSData` objects may be more suited for arbitrary pointers than `NSValue` objects.

Availability

Available in Mac OS X v10.0 and later.

See Also
- `pointerValue` (page 1879)

Declared In
`NSValue.h`

valueWithRange:

Creates and returns an `NSValue` object that contains a given `NSRange` structure.

```
+ (NSValue *)valueWithRange:(NSRange)range
```

Parameters

range
The value for the new object.

Return Value
A new `NSValue` object that contains the value of *range*.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `rangeValue` (page 1880)

Declared In
`NSRange.h`

valueWithRect:

Creates and returns an `NSValue` object that contains a given `NSRect` structure.

```
+ (NSValue *)valueWithRect:(NSRect)rect
```

Parameters

rect
The value for the new object.

Return Value
A new `NSValue` object that contains the value of *rect*.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `rectValue` (page 1880)

Related Sample Code
IBFragmentView
iSpend
QTCoreVideo301
Reducer

Declared In
NSGeometry.h

valueWithSize:

Creates and returns an NSValue object that contains a given NSSize structure.

```
+ (NSValue *)valueWithSize:(NSSize)size
```

Parameters

size
The value for the new object.

Return Value
A new NSValue object that contains the value of *size*.

Availability
Available in Mac OS X v10.0 and later.

See Also
- sizeValue (page 1881)

Related Sample Code
Dicey
ImageMapExample
QTKitMovieShuffler
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
NSGeometry.h

Instance Methods

getValue:

Copies the receiver's value into a given buffer.

```
- (void)getValue:(void *)buffer
```

Parameters

buffer
A buffer into which to copy the receiver's value. *buffer* must be large enough to hold the value.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
VideoViewer

Declared In
NSValue.h

initWithBytes:objCType:

Initializes and returns an NSValue object that contains a given value, which is interpreted as being of a given Objective-C type.

```
- (id)initWithBytes:(const void *)value objCType:(const char *)type
```

Parameters

value

The value for the new NSValue object.

type

The Objective-C type of *value*. *type* should be created with the Objective-C @encode() compiler directive; it should not be hard-coded as a C string.

Return Value
An initialized NSValue object that contains *value*, which is interpreted as being of the Objective-C type *type*. The returned object might be different than the original receiver.

Discussion
See *Number and Value Programming Topics for Cocoa* for other considerations in creating an NSValue object.

This is the designated initializer for the NSValue class.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSValue.h

isEqualToValue:

Returns a Boolean value that indicates whether the receiver and another value are equal.

```
- (BOOL)isEqualToValue:(NSValue *)value
```

Parameters

aValue

The value with which to compare the receiver.

Return Value
YES if the receiver and *aValue* are equal, otherwise NO. For NSValue objects, the class, type, and contents are compared to determine equality.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSValue.h

nonretainedObjectValue

Returns the receiver's value as an `id`.

```
- (id)nonretainedObjectValue
```

Return Value
The receiver's value as an `id`. If the receiver was not created to hold a pointer-sized data item, the result is undefined.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- getValue:` (page 1877)

Declared In
`NSValue.h`

objCType

Returns a C string containing the Objective-C type of the data contained in the receiver.

```
- (const char *)objCType
```

Return Value
A C string containing the Objective-C type of the data contained in the receiver, as encoded by the `@encode()` compiler directive.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSValue.h`

pointerValue

Returns the receiver's value as a pointer to `void`.

```
- (void *)pointerValue
```

Return Value
The receiver's value as a pointer to `void`. If the receiver was not created to hold a pointer-sized data item, the result is undefined.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- getValue:` (page 1877)

Declared In
`NSValue.h`

pointValue

Returns an `NSPoint` structure representation of the receiver.

```
- (NSPoint)pointValue
```

Return Value
An `NSPoint` structure representation of the receiver.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `- rectValue` (page 1880)
- `- sizeValue` (page 1881)

Declared In
`NSGeometry.h`

rangeValue

Returns an `NSRange` structure representation of the receiver.

```
- (NSRange)rangeValue
```

Return Value
An `NSRange` structure representation of the receiver.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `+ valueWithRange:` (page 1876)

Declared In
`NSRange.h`

rectValue

Returns an `NSRect` structure representation of the receiver.

```
- (NSRect)rectValue
```

Return Value
An `NSRect` structure representation of the receiver.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `- pointValue` (page 1880)
- `- sizeValue` (page 1881)

Related Sample Code
IBFragmentView

Declared In
`NSGeometry.h`

sizeValue

Returns an `NSSize` structure representation of the receiver.

```
- (NSSize)sizeValue
```

Return Value
An `NSSize` structure representation of the receiver.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `pointValue` (page 1880)
- `rectValue` (page 1880)

Related Sample Code
Aperture Edit Plugin - Borders & Titles
QTKitAdvancedDocument
QTKitFrameStepper
QTKitMovieShuffler
QTKitTimeCode

Declared In
`NSGeometry.h`

NSValueTransformer Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSValueTransformer.h
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Value Transformer Programming Guide
Related sample code	BindingsJoystick CustomAtomicStoreSubclass NewsReader RGB ValueTransformers TemperatureTester

Overview

`NSValueTransformer` is an abstract class that is used by the Cocoa Bindings technology to transform values from one representation to another.

An application creates a subclass of `NSValueTransformer`, overriding the necessary methods to provide the required custom transformation.

Example

A relatively trivial value transformer takes an object of type `id` and returns a string based on the object's class type. This transformer is not reversible as it's probably unreasonable to transform a class name into an object. The value transformer class you write to accomplish this simple task could look like:

```
@interface ClassNameTransformer: NSValueTransformer {}
@end
@implementation ClassNameTransformer
+ (Class)transformedValueClass { return [NSString class]; }
+ (BOOL)allowsReverseTransformation { return NO; }
- (id)transformedValue:(id)value {
    return (value == nil) ? nil : NSStringFromClass([value class]);
}
@end
```

Tasks

Using Name-based Registry

`+ setValueTransformer:forName:` (page 1885)
Registers the value transformer a given transformer with a given identifier.

`+ valueTransformerForName:` (page 1886)
Returns the value transformer identified by a given identifier.

`+ valueTransformerNames` (page 1886)
Returns an array of all the registered value transformers.

Getting Information About a Transformer

`+ allowsReverseTransformation` (page 1884)
Returns a Boolean value that indicates whether the receiver can reverse a transformation.

`+ transformedValueClass` (page 1885)
Returns the class of the value returned by the receiver for a forward transformation.

Using Transformers

`- transformedValue:` (page 1887)
Returns the result of transforming a given value.

`- reverseTransformedValue:` (page 1887)
Returns the result of the reverse transformation of a given value.

Class Methods

allowsReverseTransformation

Returns a Boolean value that indicates whether the receiver can reverse a transformation.

```
+ (BOOL)allowsReverseTransformation
```

Return Value
`YES` if the receiver supports reverse value transformations, otherwise `NO`.

The default is `NO`.

Discussion
A subclass should override this method to return `YES` if it supports reverse value transformations.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CoreRecipes
DerivedProperty

Declared In
NSValueTransformer.h

setValueTransformer:forName:

Registers the value transformer a given transformer with a given identifier.

```
+ (void)setValueTransformer:(NSValueTransformer *)transformer forName:(NSString
    *)name
```

Parameters

transformer
The transformer to register.

name
The name for *transformer*.

Availability
Available in Mac OS X v10.3 and later.

See Also
+ valueTransformerForName: (page 1886)

Related Sample Code
CoreRecipes
GridCalendar
NewsReader
RGB ValueTransformers
TemperatureTester

Declared In
NSValueTransformer.h

transformedValueClass

Returns the class of the value returned by the receiver for a forward transformation.

```
+ (Class)transformedValueClass
```

Return Value
The class of the value returned by the receiver for a forward transformation.

Discussion
A subclass should override this method to return the appropriate class.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
DerivedProperty
GridCalendar
RGB ValueTransformers
StickiesExample

Declared In
`NSValueTransformer.h`

valueTransformerForName:

Returns the value transformer identified by a given identifier.

```
+ (NSValueTransformer *)valueTransformerForName:(NSString *)name
```

Parameters

name

The transformer identifier.

Return Value
The value transformer identified by *name* in the shared registry, or `nil` if not found.

Discussion
If `valueTransformerForName:` does not find a registered transformer instance for *name*, it will attempt to find a class with the specified name. If a corresponding class is found an instance will be created and initialized using its `init:` method and then automatically registered with `name`.

Availability
Available in Mac OS X v10.3 and later.

See Also
`+ setValueTransformer:forName:` (page 1885)

Related Sample Code
BindingsJoystick
RGB ValueTransformers
TemperatureTester

Declared In
`NSValueTransformer.h`

valueTransformerNames

Returns an array of all the registered value transformers.

```
+ (NSArray *)valueTransformerNames
```

Return Value
An array of all the registered value transformers.

Availability
Available in Mac OS X v10.3 and later.

Declared In
NSValueTransformer.h

Instance Methods

reverseTransformedValue:

Returns the result of the reverse transformation of a given value.

```
- (id)reverseTransformedValue:(id)value
```

Parameters

value

The value to reverse transform.

Return Value
The reverse transformation of *value*.

Discussion
The default implementation raises an exception if `allowsReverseTransformation` (page 1884) returns `NO`; otherwise it will invoke `transformedValue:` (page 1887) with *value*.

A subclass should override this method if they require a reverse transformation that is not the same as simply reapplying the original transform (as would be the case with negation, for example). For example, if a value transformer converts a value in Fahrenheit to Celsius, this method would converts a value from Celsius to Fahrenheit.

Availability
Available in Mac OS X v10.3 and later.

See Also
`- transformedValue:` (page 1887)

Related Sample Code
TemperatureTester

Declared In
NSValueTransformer.h

transformedValue:

Returns the result of transforming a given value.

```
- (id)transformedValue:(id)value
```

Parameters

value

The value to transform.

Return Value
The result of transforming *value*.

The default implementation simply returns *value*.

Discussion
A subclass should override this method to transform and return an object based on *value*.

Availability
Available in Mac OS X v10.3 and later.

See Also
- reverseTransformedValue: (page 1887)

Related Sample Code
BindingsJoystick
CoreRecipes
RGB ValueTransformers
TemperatureTester

Declared In
NSValueTransformer.h

Constants

Named Value Transformers

The following named value transformers are defined by NSValueTransformer:

```
NSString * const NSNegateBooleanTransformerName;
NSString * const NSIsNilTransformerName ;
NSString * const NSIsNotNilTransformerName ;
NSString * const NSUnarchiveFromDataTransformerName ;
NSString * const NSKeyedUnarchiveFromDataTransformerName ;
```

Constants

NSNegateBooleanTransformerName

This value transformer negates a boolean value, transforming YES to NO and NO to YES.

This transformer is reversible.

Available in Mac OS X v10.3 and later.

Declared in NSValueTransformer.h.

NSIsNilTransformerName

This value transformer returns YES if the value is nil.

This transformer is not reversible.

Available in Mac OS X v10.3 and later.

Declared in NSValueTransformer.h.

`NSIsNotNilTransformerName`

This value transformer returns `YES` if the value is non-`nil`.

This transformer is not reversible.

Available in Mac OS X v10.3 and later.

Declared in `NSValueTransformer.h`.

`NSUnarchiveFromDataTransformerName`

This value transformer returns an object created by attempting to unarchive the data in the `NSData` object passed as the value.

The reverse transformation returns an `NSData` instance created by archiving the value. The archived object must implement the `NSCoding` protocol using sequential archiving in order to be unarchived and archived with this transformer.

Available in Mac OS X v10.3 and later.

Declared in `NSValueTransformer.h`.

`NSKeyedUnarchiveFromDataTransformerName`

This value transformer returns an object created by attempting to unarchive the data in the `NSData` object passed as the value. The archived object must be created using keyed archiving in order to be unarchived and archived with this transformer.

The reverse transformation returns an `NSData` instance created by archiving the value using keyed archiving. The archived object must implement the `NSCoding` protocol using keyed archiving in order to be unarchived and archived with this transformer.

Available in Mac OS X v10.5 and later.

Declared in `NSValueTransformer.h`.

Declared In

`NSValueTransformer.h`

NSWhoseSpecifier Class Reference

Inherits from	NSScriptObjectSpecifier : NSObject
Conforms to	NSCoding (NSScriptObjectSpecifier) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSScriptObjectSpecifiers.h
Companion guide	Cocoa Scripting Guide

Overview

`NSWhoseSpecifier` specifies every object in a collection (or every element in a container) that matches the condition defined by a single Boolean expression or multiple Boolean expressions connected by logical operators. `NSWhoseSpecifier` is unique among object specifiers in that its top-level container is typically not the application object but an evaluated object specifier involved in the tested-for condition. An `NSWhoseSpecifier` object encapsulates a "test" object for defining this condition. A test object is instantiated from a subclass of the abstract `NSScriptWhoseTest` class, whose one declared method is `isTrue` (page 1438). See "Boolean Expressions and Logical Operations" in `NSScriptObjectSpecifier` and the descriptions in `NSComparisonMethods` and `NSScriptingComparisonMethods` for more information.

The set of elements specified by an `NSWhoseSpecifier` object can be a subset of those that pass the `NSWhoseSpecifier` object's test. This subset is specified by the various sub-element properties of the `NSWhoseSpecifier` object . Consider as an example the specifier `paragraphs where color of third word is blue`. This would be represented by an `NSWhoseSpecifier` object that uses a test specifier and another object specifier to identify a subset of the objects with the specified property. That is, the specifier's property is `paragraphs`; the test specifier is an index specifier with property `words` and `index 3`; and the qualifier is a key value qualifier for key `color` and value `[NSColor blueColor]`. The test object specifier (`word at index 3`) is evaluated for each object (paragraph) using that object as the container; the resulting objects (if any) are tested with the qualifier (`color blue`).

`NSWhoseSpecifier` is part of Cocoa's built-in script handling. You don't normally subclass it.

Tasks

Initializing a Whose Specifier

- `initWithContainerClassDescription:containerSpecifier:key:test:` (page 1893)
  Returns an `NSWhoseSpecifier` object initialized with the given attributes.

Accessing Information About a Whose Specifier

- `endSubelementIdentifier` (page 1892)
  Returns the end sub-element identifier for the receiver.
- `endSubelementIndex` (page 1893)
  Returns the index position of the last sub-element within the range of objects being tested that passes the receiver's test.
- `setEndSubelementIdentifier:` (page 1894)
  Sets the end sub-element identifier for the specifier to the value of a given sub-element.
- `setEndSubelementIndex:` (page 1894)
  Sets the index position of the last sub-element within the range of objects being tested that pass the specifier's test.
- `setStartSubelementIdentifier:` (page 1894)
  Sets the start sub-element identifier for the specifier.
- `setStartSubelementIndex:` (page 1895)
  Sets the index position of the first sub-element within the range of objects being tested that passes the specifier's test.
- `setTest:` (page 1895)
  Sets the test object that is encapsulated by the receiver.
- `startSubelementIdentifier` (page 1895)
  Returns the start sub-element identifier for the receiver.
- `startSubelementIndex` (page 1896)
  Returns the index position of the first sub-element within the range of objects being tested that pass the receiver's test.
- `test` (page 1896)
  Returns the test object encapsulated by the receiver.

Instance Methods

endSubelementIdentifier

Returns the end sub-element identifier for the receiver.

```
- (NSWhoseSubelementIdentifier)endSubelementIdentifier
```

Return Value
The end sub-element identifier for the receiver, or `NSNoSubelement` if there is none.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptObjectSpecifiers.h`

endSubelementIndex

Returns the index position of the last sub-element within the range of objects being tested that passes the receiver's test.

```
- (NSInteger)endSubelementIndex
```

Return Value
The index position of the last sub-element within the range of objects being tested that passes the receiver's test.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptObjectSpecifiers.h`

initWithContainerClassDescription:containerSpecifier:key:test:

Returns an `NSWhoseSpecifier` object initialized with the given attributes.

```
- (id)initWithContainerClassDescription:(NSScriptClassDescription *)classDescription
    containerSpecifier:(NSScriptObjectSpecifier *)specifier key:(NSString *)property
    test:(NSScriptWhoseTest *)test
```

Parameters

classDescription
Class description for the receiver's container object.

specifier
An object specifier for the receiver's container object.

property
The key for the property for which to test.

test
The test condition.

Return Value
An `NSWhoseSpecifier` object initialized with the given attributes.

Discussion
Invokes the super class's `initWithContainerClassDescription:containerSpecifier:key:` (page 1418) and sets the whose test condition to *test*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptObjectSpecifiers.h

setEndSubelementIdentifier:

Sets the end sub-element identifier for the specifier to the value of a given sub-element.

```
- (void)setEndSubelementIdentifier:(NSWhoseSubelementIdentifier)subelement
```

Parameters

subelement
The end sub-element for the receiver.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptObjectSpecifiers.h

setEndSubelementIndex:

Sets the index position of the last sub-element within the range of objects being tested that pass the specifier's test.

```
- (void)setEndSubelementIndex:(NSInteger)index
```

Parameters

index
The index position of the end sub-element.

Discussion
Used only if the end sub-element identifier is `NSIndexSubelement`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptObjectSpecifiers.h

setStartSubelementIdentifier:

Sets the start sub-element identifier for the specifier.

```
- (void)setStartSubelementIdentifier:(NSWhoseSubelementIdentifier)subelement
```

Parameters

subelement
The start sub-element for the receiver.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptObjectSpecifiers.h`

setStartSubelementIndex:

Sets the index position of the first sub-element within the range of objects being tested that passes the specifier's test.

```
- (void)setStartSubelementIndex:(NSInteger)index
```

Parameters

index
The index position of the start sub-element.

Discussion
Used only if the start sub-element identifier is `NSIndexSubelement`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptObjectSpecifiers.h`

setTest:

Sets the test object that is encapsulated by the receiver.

```
- (void)setTest:(NSScriptWhoseTest *)test
```

Parameters

test
The test object for the receiver.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptObjectSpecifiers.h`

startSubelementIdentifier

Returns the start sub-element identifier for the receiver.

```
- (NSWhoseSubelementIdentifier)startSubelementIdentifier
```

Return Value
The start sub-element identifier for the receiver.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptObjectSpecifiers.h

startSubelementIndex

Returns the index position of the first sub-element within the range of objects being tested that pass the receiver's test.

```
- (NSInteger)startSubelementIndex
```

Return Value
The index position of the first sub-element within the range of objects being tested that pass the receiver's test.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptObjectSpecifiers.h

test

Returns the test object encapsulated by the receiver.

```
- (NSScriptWhoseTest *)test
```

Return Value
The test object encapsulated by the receiver.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptObjectSpecifiers.h

Constants

NSWhoseSubelementIdentifier

`NSWhoseSpecifier` uses these constants to specify sub-elements within the collection of objects being tested that pass the specifier's test.

```
typedef enum {
    NSIndexSubelement  = 0,
    NSEverySubelement  = 1,
    NSMiddleSubelement = 2,
    NSRandomSubelement = 3,
    NSNoSubelement     = 4
} NSWhoseSubelementIdentifier;
```

Constants

`NSIndexSubelement`

An element at a given index that meets the specifier test.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptObjectSpecifiers.h`.

`NSEverySubelement`

Every element that meets the specifier test.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptObjectSpecifiers.h`.

`NSMiddleSubelement`

The middle element that meets the specifier test.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptObjectSpecifiers.h`.

`NSRandomSubelement`

Any element that meets the specifier test.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptObjectSpecifiers.h`.

`NSNoSubelement`

No sub-element met the specifier test. Valid only for specifying the end sub-element.; that is, there is no end, so consider all elements.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptObjectSpecifiers.h`.

Discussion

These constants are used by `startSubelementIdentifier` (page 1895), `setStartSubelementIdentifier:` (page 1894), `endSubelementIdentifier` (page 1892), and `setEndSubelementIdentifier:` (page 1894).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSScriptObjectSpecifiers.h`

NSXMLDocument Class Reference

Inherits from	NSXMLNode : NSObject
Conforms to	NSCopying (NSXMLNode) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSXMLDocument.h
Availability	Available in Mac OS X v10.4 and later.
Companion guide	Tree-Based XML Programming Guide for Cocoa
Related sample code	AlbumToSlideshow CocoaSOAP Core Data HTML Store TimelineToTC

Overview

An instance of `NSXMLDocument` represents an XML document as internalized into a logical tree structure. An `NSXMLDocument` object can have multiple child nodes but only one element, the root element. Any other node must be a `NSXMLNode` object representing a comment or a processing instruction. If you attempt to add any other kind of child node to an `NSXMLDocument` object, such as an attribute, namespace, another document object, or an element other than the root, `NSXMLDocument` raises an exception. If you add a valid child node and that object already has a parent, `NSXMLDocument` raises an exception. An `NSXMLDocument` object may also have document-global attributes, such as XML version, character encoding, referenced DTD, and MIME type.

The initializers of the `NSXMLDocument` class read an external source of XML, whether it be a local file or remote website, parse it, and process it into the tree representation. You can also construct an `NSXMLDocument` programmatically. There are accessor methods for getting and setting document attributes, methods for transforming documents using XSLT, a method for dynamically validating a document, and methods for printing out the content of an `NSXMLDocument` as XML, XHTML, HTML, or plain text.

Subclassing Notes

Methods to Override

To subclass `NSXMLDocument` you need to override the primary initializer, `initWithData:options:error:` (page 1906), and the methods listed below. In most cases, you need only invoke the superclass implementation, adding any subclass-specific code before or after the invocation, as necessary.

- `rootElement` (page 1913)
- `setChildren:` (page 1914)
- `removeChildAtIndex:` (page 1912)
- `insertChild:atIndex:` (page 1908)
- `characterEncoding` (page 1904)
- `setCharacterEncoding:` (page 1913)
- `documentContentKind` (page 1905)
- `setDocumentContentKind:` (page 1914)
- `DTD` (page 1905)
- `setDTD:` (page 1915)
- `MIMEType` (page 1909)
- `setMIMEType:` (page 1915)
- `isStandalone` (page 1909)
- `setStandalone:` (page 1916)
- `version` (page 1918)
- `setURI:` (page 1916)
- `setVersion:` (page 1917)

By default `NSXMLDocument` implements the `NSObject` `isEqual:` (page 2101) method to perform a deep comparison: two `NSXMLDocument` objects are not considered equal unless they have the same name, same child nodes, same attributes, and so on. The comparison does not consider the parent node (and hence the node's location). If you want a different standard of comparison, override `isEqual:`.

Special Considerations

Because of the architecture and data model of NSXML, when it parses and processes a source of XML it cannot know about your subclass unless you override the class method `replacementClassForClass:` (page 1903) to return your custom class in place of an `NSXML` class. If your custom class has no direct `NSXML` counterpart—for example, it is a subclass of `NSXMLNode` that represents CDATA sections—then you can walk the tree after it has been created and insert the new node where appropriate.

Tasks

Initializing NSXMLDocument Objects

- `initWithContentsOfURL:options:error:` (page 1905)
  Initializes and returns an NSXMLDocument object created from the XML or HTML contents of a URL-referenced source
- `initWithData:options:error:` (page 1906)
  Initializes and returns an `NSXMLDocument` object created from an `NSData` object.
- `initWithRootElement:` (page 1907)
  Returns an `NSXMLDocument` object initialized with a single child, the root element.
- `initWithXMLString:options:error:` (page 1907)
  Initializes and returns an `NSXMLDocument` object created from a string containing XML markup text.
+ `replacementClassForClass:` (page 1903)
  Overridden by subclasses to substitute a custom class for an NSXML class that the parser uses to create node instances.

Managing Document Attributes

- `characterEncoding` (page 1904)
  Returns the character encoding used for the XML.
- `setCharacterEncoding:` (page 1913)
  Sets the character encoding of the receiver to *`encoding`*,
- `documentContentKind` (page 1905)
  Returns the kind of document content for output.
- `setDocumentContentKind:` (page 1914)
  Sets the kind of output content for the receiver.
- `DTD` (page 1905)
  Returns an `NSXMLDTD` object representing the internal DTD associated with the receiver.
- `setDTD:` (page 1915)
  Sets the internal DTD to be associated with the receiver.
- `isStandalone` (page 1909)
  Returns whether the receiver represents a standalone XML document—that is, one without an external DTD.
- `setStandalone:` (page 1916)
  Sets a Boolean value that specifies whether the receiver represents a standalone XML document.
- `MIMEType` (page 1909)
  Returns the MIME type for the receiver.
- `setMIMEType:` (page 1915)
  Sets the MIME type of the receiver.
- `URI` (page 1917)
  Returns the URI identifying the source of this document.

- `setURI:` (page 1916)
  Sets the URI identifying the source of this document.
- `version` (page 1918)
  Returns the version of the receiver's XML.
- `setVersion:` (page 1917)
  Sets the version of the receiver's XML.

Managing the Root Element

- `rootElement` (page 1913)
  Returns the root element of the receiver.
- `setRootElement:` (page 1915)
  Set the root element of the receiver.

Adding and Removing Child Nodes

- `addChild:` (page 1904)
  Adds a child node after the last of the receiver's existing children.
- `insertChild:atIndex:` (page 1908)
  Inserts a node object at specified position in the receiver's array of children.
- `insertChildren:atIndex:` (page 1908)
  Inserts an array of children at a specified position in the receiver's array of children.
- `removeChildAtIndex:` (page 1912)
  Removes the child node of the receiver located at a specified position in its array of children.
- `replaceChildAtIndex:withNode:` (page 1912)
  Replaces the child node of the receiver located at a specified position in its array of children with another node.
- `setChildren:` (page 1914)
  Sets the child nodes of the receiver.

Transforming a Document Using XSLT

- `objectByApplyingXSLT:arguments:error:` (page 1910)
  Applies the XSLT pattern rules and templates (specified as a data object) to the receiver and returns a document object containing transformed XML or HTML markup.
- `objectByApplyingXSLTString:arguments:error:` (page 1911)
  Applies the XSLT pattern rules and templates (specified as a string) to the receiver and returns a document object containing transformed XML or HTML markup.
- `objectByApplyingXSLTAtURL:arguments:error:` (page 1910)
  Applies the XSLT pattern rules and templates located at a specified URL to the receiver and returns a document object containing transformed XML markup or an `NSData` object containing plain text, RTF text, and so on.

Writing a Document as XML Data

- `XMLData` (page 1918)
  Returns the XML string representation of the receiver—that is, the entire document—encapsulated in a data object.
- `XMLDataWithOptions:` (page 1919)
  Returns the XML string representation of the receiver—that is, the entire document—encapsulated in a data object.

Validating a Document

- `validateAndReturnError:` (page 1917)
  Validates the document against the governing schema and returns whether the document conforms to the schema.

Class Methods

replacementClassForClass:

Overridden by subclasses to substitute a custom class for an NSXML class that the parser uses to create node instances.

```
+ (Class)replacementClassForClass:(Class)class
```

Parameters

class
  A `Class` object identifying an NSXML class that is to be replaced by your custom class.

Return Value
The substituted class.

Discussion
For example, if you have a custom subclass of `NSXMLElement` that you want to be used in place of `NSXMLElement`, you would make the following override:

```
+ (Class)replacementClassForClass:(Class)currentClass {
    if ( currentClass == [NSXMLElement class] ) {
        return [MyCustomElementClass class];
    }
}
```

This method is invoked before a document is parsed. The substituted class must be a subclass of `NSXMLNode`, `NSXMLDocument`, `NSXMLElement`, `NSXMLDTD`, or `NSXMLDTDNode`.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setRootElement:` (page 1915)

Declared In
NSXMLDocument.h

Instance Methods

addChild:

Adds a child node after the last of the receiver's existing children.

```
- (void)addChild:(NSXMLNode *)child
```

Parameters
child
The NSXMLNode object to be added.

Availability
Available in Mac OS X v10.4 and later.

See Also
- insertChild:atIndex: (page 1908)
- removeChildAtIndex: (page 1912)
- setChildren: (page 1914)

Declared In
NSXMLDocument.h

characterEncoding

Returns the character encoding used for the XML.

```
- (NSString *)characterEncoding
```

Return Value
The character encoding used for the XML, or nil if no encoding is specified.

Discussion
Typically the encoding is specified in the XML declaration of a document that is processed, but it can be set at any time. If the specified encoding does not match the actual encoding, parsing of the document may fail.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setCharacterEncoding: (page 1913)

Declared In
NSXMLDocument.h

documentContentKind

Returns the kind of document content for output.

```
- (NSXMLDocumentContentKind)documentContentKind
```

Discussion
Most of the differences among content kind have to do with the handling of content-less tags such as
. The valid NSXMLDocumentContentKind constants are NSXMLDocumentXMLKind, NSXMLDocumentXHTMLKind, NSXMLDocumentHTMLKind, and NSXMLDocumentTextKind.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setDocumentContentKind: (page 1914)

Declared In
NSXMLDocument.h

DTD

Returns an NSXMLDTD object representing the internal DTD associated with the receiver.

```
- (NSXMLDTD *)DTD
```

Return Value
An NSXMLDTD object representing the internal DTD associated with the receiver or nil if no DTD has been associated.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setDTD: (page 1915)

Declared In
NSXMLDocument.h

initWithContentsOfURL:options:error:

Initializes and returns an NSXMLDocument object created from the XML or HTML contents of a URL-referenced source

```
- (id)initWithContentsOfURL:(NSURL *)url options:(NSUInteger)mask error:(NSError
    **)error
```

Parameters

url

An NSURL object specifying a URL source.

mask

A bit mask for input options. You can specify multiple options by bit-OR'ing them. See "Constants" (page 1919) for a list of valid input options.

error

An error object that, on return, identifies any parsing errors and warnings or connection problems.

Return Value
An initialized `NSXMLDocument` object, or `nil` if initialization fails because of parsing errors or other reasons.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `initWithData:options:error:` (page 1906)
- `initWithRootElement:` (page 1907)
- `initWithXMLString:options:error:` (page 1907)

Declared In
`NSXMLDocument.h`

initWithData:options:error:

Initializes and returns an `NSXMLDocument` object created from an `NSData` object.

```
- (id)initWithData:(NSData *)data options:(NSUInteger)mask error:(NSError **)error
```

Parameters

data

A data object with XML content.

mask

A bit mask for input options. You can specify multiple options by bit-OR'ing them. See "Constants" (page 1919) for a list of valid input options.

error

An error object that, on return, identifies any parsing errors and warnings or connection problems.

Return Value
An initialized `NSXMLDocument` object, or `nil` if initialization fails because of parsing errors or other reasons.

Discussion
This method is the designated initializer for the `NSXMLDocument` class.

If you specify `NSXMLDocumentTidyXML` as one of the options, NSXMLDocument performs several clean-up operations on the document XML (such as removing leading tabs). It does however, respect the `xmlns:space="preserve"` attribute when it attempts to tidy the XML.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `initWithContentsOfURL:options:error:` (page 1905)
- `initWithRootElement:` (page 1907)
- `initWithXMLString:options:error:` (page 1907)

Declared In
`NSXMLDocument.h`

initWithRootElement:

Returns an `NSXMLDocument` object initialized with a single child, the root element.

```
- (id)initWithRootElement:(NSXMLElement *)root
```

Parameters

root

An `NSXMLElement` object representing an XML element.

Return Value
An initialized `NSXMLDocument` object, or `nil` if initialization fails for any reason.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `initWithContentsOfURL:options:error:` (page 1905)
- `initWithData:options:error:` (page 1906)
- `initWithXMLString:options:error:` (page 1907)

Related Sample Code
AlbumToSlideshow

Declared In
`NSXMLDocument.h`

initWithXMLString:options:error:

Initializes and returns an `NSXMLDocument` object created from a string containing XML markup text.

```
- (id)initWithXMLString:(NSString *)string options:(NSUInteger)mask error:(NSError
    **)error
```

Parameters

string

A string object containing XML markup text.

mask

A bit mask for input options. You can specify multiple options by bit-OR'ing them. See "Constants" (page 1919) for a list of valid input options.

error

An error object that, on return, identifies any parsing errors and warnings or connection problems.

Return Value
An initialized `NSXMLDocument` object, or `nil` if initialization fails because of parsing errors or other reasons.

Discussion
The encoding of the document is set to UTF-8.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `initWithContentsOfURL:options:error:` (page 1905)

- initWithData:options:error: (page 1906)
- initWithRootElement: (page 1907)

Related Sample Code
CocoaSOAP

Declared In
NSXMLDocument.h

insertChild:atIndex:

Inserts a node object at specified position in the receiver's array of children.

```
- (void)insertChild:(NSXMLNode *)child atIndex:(NSUInteger)index
```

Parameters

child

The NSXMLNode object to be inserted. The added node must be an NSXMLNode object representing a comment, processing instruction, or the root element.

index

An integer specifying the index of the children array to insert *child*. The indexes of children after the new child are incremented. If *index* is less than zero or greater than the number of children, an out-of-bounds exception is raised.

Availability
Available in Mac OS X v10.4 and later.

See Also
- addChild: (page 1904)
- insertChildren:atIndex: (page 1908)
- removeChildAtIndex: (page 1912)
- replaceChildAtIndex:withNode: (page 1912)

Declared In
NSXMLDocument.h

insertChildren:atIndex:

Inserts an array of children at a specified position in the receiver's array of children.

```
- (void)insertChildren:(NSArray *)children atIndex:(NSUInteger)index
```

Parameters

children

An array of NSXMLNode objects representing comments, processing instructions, or the root element.

index

An integer identifying the location in the receiver's children array for insertion. The indexes of children after the new child are increased by [children count]. If *index* is less than zero or greater than the number of children, an out-of-bounds exception is raised.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `addChild:` (page 1904)
- `removeChildAtIndex:` (page 1912)
- `replaceChildAtIndex:withNode:` (page 1912)
- `setChildren:` (page 1914)

Declared In
`NSXMLDocument.h`

isStandalone

Returns whether the receiver represents a standalone XML document—that is, one without an external DTD.

```
- (BOOL)isStandalone
```

Return Value
`YES` if the receiver represents a standalone XML document, `NO` if the "standalone" declaration was not present in the original document and hasn't been set since.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setStandalone:` (page 1916)

Declared In
`NSXMLDocument.h`

MIMEType

Returns the MIME type for the receiver.

```
- (NSString *)MIMEType
```

Return Value
The MIME type for the receiver (for example, "text/xml").

Discussion
MIME types are assigned by IANA (see http://www.iana.org/assignments/media-types/index.html).

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setMIMEType:` (page 1915)

Declared In
`NSXMLDocument.h`

objectByApplyingXSLT:arguments:error:

Applies the XSLT pattern rules and templates (specified as a data object) to the receiver and returns a document object containing transformed XML or HTML markup.

```
- (id)objectByApplyingXSLT:(NSData *)xslt arguments:(NSDictionary *)arguments
    error:(NSError **)error
```

Parameters

xslt

A data object containing the XSLT pattern rules and templates.

arguments

A dictionary containing `NSString` key-value pairs that are passed as runtime parameters to the XSLT processor. Pass in `nil` if you have no parameters to pass.

Note: Several XML websites discuss XSLT parameters, including O'Reilly Media's http://www.xml.com.

error

If an error occurs, indirectly returns an `NSError` object encapsulating error or warning messages generated by XSLT processing.

Return Value

Depending on intended output, the method returns an `NSXMLDocument` object *or* an `NSData` data containing transformed XML or HTML markup. If the message is supposed to create plain text or RTF, then an `NSData` object is returned, otherwise an XML document object. The method returns `nil` if XSLT processing did not succeed.

Availability

Available in Mac OS X v10.4 and later.

See Also

- `objectByApplyingXSLTAtURL:arguments:error:` (page 1910)
- `objectByApplyingXSLTString:arguments:error:` (page 1911)

Declared In

`NSXMLDocument.h`

objectByApplyingXSLTAtURL:arguments:error:

Applies the XSLT pattern rules and templates located at a specified URL to the receiver and returns a document object containing transformed XML markup or an `NSData` object containing plain text, RTF text, and so on.

```
- (id)objectByApplyingXSLTAtURL:(NSURL *)xsltURL arguments:(NSDictionary *)arguments
    error:(NSError **)error
```

Parameters

xsltURL

An `NSURL` object specifying a valid URL.

arguments

A dictionary containing `NSString` key-value pairs that are passed as runtime parameters to the XSLT processor. Pass in `nil` if you have no parameters to pass.

Note: Several XML websites discuss XSLT parameters, including O'Reilly Media's http://www.xml.com.

error

If an error occurs, indirectly returns an `NSError` object encapsulating error or warning messages generated by XSLT processing or from an attempt to connect to a website identified by the URL.

Return Value

Depending on intended output, the returns an `NSXMLDocument` object *or* an `NSData` data containing transformed XML or HTML markup. If the message is supposed to create plain text or RTF, then an `NSData` object is returned, otherwise an XML document object. The method returns `nil` if XSLT processing did not succeed.

Availability

Available in Mac OS X v10.4 and later.

See Also

- `objectByApplyingXSLT:arguments:error:` (page 1910)
- `objectByApplyingXSLTString:arguments:error:` (page 1911)

Declared In

`NSXMLDocument.h`

objectByApplyingXSLTString:arguments:error:

Applies the XSLT pattern rules and templates (specified as a string) to the receiver and returns a document object containing transformed XML or HTML markup.

```
- (id)objectByApplyingXSLTString:(NSString *)xslt arguments:(NSDictionary
    *)arguments error:(NSError **)error
```

Parameters

xslt

A string object containing the XSLT pattern rules and templates.

arguments

A dictionary containing `NSString` key-value pairs that are passed as runtime parameters to the XSLT processor. Pass in `nil` if you have no parameters to pass.

Note: Several XML websites discuss XSLT parameters, including O'Reilly Media's http://www.xml.com.

error

If an error occurs, indirectly returns an `NSError` object encapsulating error or warning messages generated by XSLT processing.

Return Value
Depending on intended output, the method returns an `NSXMLDocument` object *or* an `NSData` data containing transformed XML or HTML markup. If the message is supposed to create plain text or RTF, then an `NSData` object is returned, otherwise an XML document object. The method returns `nil` if XSLT processing did not succeed.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `objectByApplyingXSLT:arguments:error:` (page 1910)
- `objectByApplyingXSLTAtURL:arguments:error:` (page 1910)

Declared In
`NSXMLDocument.h`

removeChildAtIndex:

Removes the child node of the receiver located at a specified position in its array of children.

```
- (void)removeChildAtIndex:(NSUInteger)index
```

Parameters

index

An integer identifying the position of an child in the receiver's array. If *index* is less than zero or greater than the number of children minus one, an out-of-bounds exception is raised.

Discussion
Subsequent children have their indexes decreased by one. The removed `NSXMLNode` object is autoreleased.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `insertChild:atIndex:` (page 1908)
- `replaceChildAtIndex:withNode:` (page 1912)

Declared In
`NSXMLDocument.h`

replaceChildAtIndex:withNode:

Replaces the child node of the receiver located at a specified position in its array of children with another node.

```
- (void)replaceChildAtIndex:(NSUInteger)index withNode:(NSXMLNode *)node
```

Parameters

index

An integer identifying a position in the receiver's array of children. If *index* is less than zero or greater than the number of children minus one, an out-of-bounds exception is raised.

node

An `NSXMLNode` object to replace the one at *index*; it must represent a comment, a processing instruction, or the root element.

Discussion
The removed `NSXMLNode` object is autoreleased.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `insertChild:atIndex:` (page 1908)
- `removeChildAtIndex:` (page 1912)

Declared In
`NSXMLDocument.h`

rootElement

Returns the root element of the receiver.

```
- (NSXMLElement *)rootElement
```

Return Value
The root element of the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setRootElement:` (page 1915)

Declared In
`NSXMLDocument.h`

setCharacterEncoding:

Sets the character encoding of the receiver to *encoding*,

```
- (void)setCharacterEncoding:(NSString *)encoding
```

Parameters

encoding

A string that specifies an encoding; it must match the name of an IANA character set. See http://www.iana.org/assignments/character-sets for a list of valid encoding specifiers.

Discussion
Typically the encoding is specified in the XML declaration of a document that is processed, but it can be set at any time. If the specified encoding does not match the actual encoding, parsing of the document might fail.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `characterEncoding` (page 1904)

Related Sample Code
AlbumToSlideshow

Declared In
`NSXMLDocument.h`

setChildren:

Sets the child nodes of the receiver.

```
- (void)setChildren:(NSArray *)children
```

Parameters

children

An array of `NSXMLNode` objects. Each of these objects must represent comments, processing instructions, or the root element; otherwise, an exception is raised. Pass in `nil` to remove all children.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `addChild:` (page 1904)
- `insertChildren:atIndex:` (page 1908)

Declared In
`NSXMLDocument.h`

setDocumentContentKind:

Sets the kind of output content for the receiver.

```
- (void)setDocumentContentKind:(NSXMLDocumentContentKind)kind
```

Parameters

kind

An `enum` constant identifying a kind of document content. The valid `NSXMLDocumentContentKind` constants are `NSXMLDocumentXMLKind`, `NSXMLDocumentXHTMLKind`, `NSXMLDocumentHTMLKind`, and `NSXMLDocumentTextKind`.

Discussion
Most of the differences among document-content kind have to do with the handling of content-less tags such as `<br>`.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `documentContentKind` (page 1905)

Declared In
NSXMLDocument.h

setDTD:

Sets the internal DTD to be associated with the receiver.

```
- (void)setDTD:(NSXMLDTD *)documentTypeDeclaration
```

Parameters

documentTypeDeclaration
An NSXMLDTD object representing the internal DTD to be associated with the receiver.

Discussion
When the receiver is written out, this document type declaration appears in the output, just after the XML declaration.

Availability
Available in Mac OS X v10.4 and later.

See Also
- DTD (page 1905)

Declared In
NSXMLDocument.h

setMIMEType:

Sets the MIME type of the receiver.

```
- (void)setMIMEType:(NSString *)MIMEType
```

Parameters

MIMEType
A string object identifying a MIME type, for example, "text/xml". MIME types are assigned by IANA (see http://www.iana.org/assignments/media-types/index.html).

Availability
Available in Mac OS X v10.4 and later.

See Also
- MIMEType (page 1909)

Declared In
NSXMLDocument.h

setRootElement:

Set the root element of the receiver.

```
- (void)setRootElement:(NSXMLNode *)root
```

Parameters

root

An `NSXMLNode` object that is to be the root element.

Discussion

As a side effect, this method removes all other children, including `NSXMLNode` objects representing comments and processing-instructions.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- rootElement` (page 1913)

Declared In

`NSXMLDocument.h`

setStandalone:

Sets a Boolean value that specifies whether the receiver represents a standalone XML document.

```
- (void)setStandalone:(BOOL)standalone
```

Parameters

standalone

`YES` if the receiver represents a standalone XML document, `NO` otherwise.

Discussion

A standalone document does not have an external DTD associated with it.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- isStandalone` (page 1909)

Declared In

`NSXMLDocument.h`

setURI:

Sets the URI identifying the source of this document.

```
- (void)setURI:(NSString *)URI
```

Parameters

URI

A string object representing a URI source, or `nil` to remove the current URI.

Discussion

This attribute is automatically set when the receiver is initialized using `initWithContentsOfURL:options:error:` (page 1905).

See Also
- `URI` (page 1917)

setVersion:

Sets the version of the receiver's XML.

```
- (void)setVersion:(NSString *)version
```

Parameters

version

A string object identifying the version of the XML.

Discussion
Currently, the version should be either "1.0 "or "1.1".

Availability
Available in Mac OS X v10.4 and later.

See Also
- `version` (page 1918)

Related Sample Code
AlbumToSlideshow

Declared In
`NSXMLDocument.h`

URI

Returns the URI identifying the source of this document.

```
- (NSString *)URI
```

Return Value
The URI identifying the source of this document or `nil` if this attribute has not been set.

See Also
- `setURI:` (page 1916)

validateAndReturnError:

Validates the document against the governing schema and returns whether the document conforms to the schema.

```
- (BOOL)validateAndReturnError:(NSError **)error
```

Parameters

error

If validation fails, on return contains an `NSError` object describing the reason or reasons for failure.

Return Value
YES if the validation operation succeeded, otherwise NO.

Discussion
The constants indicating the kind of validation errors are emitted by the underlying parser; see NSXMLParser.h for most of these constants. If the schema is defined with a DTD, this method uses the NSXMLDTD object set for the receiver for validation. If the schema is based on XML Schema, the method uses the URL specified as the value of the xsi:schemaLocation attribute of the root element.

You can validate an XML document when it is first processed by specifying the NSXMLDocumentValidate option when you initialize an NSXMLDocument object with the initWithContentsOfURL:options:error: (page 1905), initWithData:options:error: (page 1906), or initWithXMLString:options:error: (page 1907) methods.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setDTD: (page 1915)

Declared In
NSXMLDocument.h

version

Returns the version of the receiver's XML.

```
- (NSString *)version
```

Return Value
The version of the receiver's XML or nil if the version has not be set.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setVersion: (page 1917)

Declared In
NSXMLDocument.h

XMLData

Returns the XML string representation of the receiver—that is, the entire document—encapsulated in a data object.

```
- (NSData *)XMLData
```

Discussion
This method invokes XMLDataWithOptions: with an option of NSXMLNodeOptionsNone. The encoding used is based on the value returned from characterEncoding (page 1904) or UTF-8 if no valid encoding is returned by that method.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `XMLDataWithOptions:` (page 1919)

Related Sample Code
CocoaSOAP

Declared In
`NSXMLDocument.h`

XMLDataWithOptions:

Returns the XML string representation of the receiver—that is, the entire document—encapsulated in a data object.

```
- (NSData *)XMLDataWithOptions:(NSUInteger)options
```

Parameters

options

One or more options (bit-OR'd if multiple) to affect the output of the document; see "Constants" (page 1919) for the valid output options.

Discussion
The encoding used is based on the value returned from `characterEncoding` (page 1904).

Availability
Available in Mac OS X v10.4 and later.

See Also
- `XMLData` (page 1918)

Related Sample Code
AlbumToSlideshow

Declared In
`NSXMLDocument.h`

Constants

Input and Output Options

Input and output options specifically intended for `NSXMLDocument` objects.

```
NSXMLDocumentTidyHTML = 1 << 9,
NSXMLDocumentTidyXML = 1 << 10,
NSXMLDocumentValidate = 1 << 13,
NSXMLDocumentXInclude = 1 << 16,
NSXMLDocumentIncludeContentTypeDeclaration = 1 << 18,
```

Constants

`NSXMLDocumentTidyHTML`

Formats HTML into valid XHTML during processing of the document.

When tidying, `NSXMLDocument` adds a line break before the close tag of a block-level element (`<p>`, `<div>`, `<h1>`, and so on); it also makes the string value of `<br>` or `<hr>` a line break. These operations make the string value of the HTML `<body>` more readable. After using this option, avoid outputting the document as anything other than the default kind, `NSXMLDocumentXHTMLKind`.

(Input)

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLDocumentTidyXML`

Changes malformed XML into valid XML during processing of the document.

It also eliminates "pretty-printing" formatting, such as leading tab characters. However, it respects the `xmlns:space="preserve"` attribute.

(Input)

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLDocumentValidate`

Validates this document against its DTD (internal or external) or XML Schema.

(Input)

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLDocumentXInclude`

Replaces all XInclude nodes in the document with the nodes referred to.

XInclude allows clients to include parts of another XML document within a document.

(Input)

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLDocumentIncludeContentTypeDeclaration`

Includes a content type declaration for HTML or XHTML in the output of the document.

(Output)

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

Discussion

Because `NSXMLDocument` is a subclass of `NSXMLNode`, you can also use the relevant input and output options described in "Constants" (page 1992) in the `NSXMLNode` class reference. You can specify input options in the `NSXMLDocument` methods `initWithContentsOfURL:options:error:` (page 1905), `initWithData:options:error:` (page 1906), `initWithXMLString:options:error:` (page 1907). The `XMLDataWithOptions:` (page 1919) method takes output options.

Declared In
NSXMLNodeOptions.h

NSXMLDocumentContentKind

Type used to define the kind of document content.

```
typedef NSUInteger NSXMLDocumentContentKind;
```

Discussion
For possible values, see "Document Content Types" (page 1921).

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSXMLDocument.h

Document Content Types

Define document types.

```
enum {
    NSXMLDocumentXMLKind = 0,
    NSXMLDocumentXHTMLKind,
    NSXMLDocumentHTMLKind,
    NSXMLDocumentTextKind
};
```

Constants

`NSXMLDocumentXMLKind`
The default type of document content type, which is XML.
Available in Mac OS X v10.4 and later.
Declared in NSXMLDocument.h.

`NSXMLDocumentXHTMLKind`
The document output is XHTML.
This is set automatically if the NSXMLDocumentTidyHTML option is set and NSXML detects HTML.
Available in Mac OS X v10.4 and later.
Declared in NSXMLDocument.h.

`NSXMLDocumentHTMLKind`
Outputs empty tags in HTML without a close tag, such as
.
Available in Mac OS X v10.4 and later.
Declared in NSXMLDocument.h.

`NSXMLDocumentTextKind`
Outputs the string value of the document by extracting the string values from all text nodes.
Available in Mac OS X v10.4 and later.
Declared in NSXMLDocument.h.

Discussion
You specify one of the `NSXMLDocumentContentKind` constants in `setDocumentContentKind:` (page 1914) to indicate the kind of content required for document output.

Declared In
`NSXMLDocument.h`

NSXMLDTD Class Reference

Inherits from	NSXMLNode : NSObject
Conforms to	NSCopying (NSXMLNode) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/Foundation.h
Availability	Available in Mac OS X v10.4 and later.
Companion guide	Tree-Based XML Programming Guide for Cocoa

Overview

An instance of the `NSXMLDTD` class represents a Document Type Definition. It is held as a property of an `NSXMLDocument` instance, accessed through the `NSXMLDocument` method `DTD` (page 1905) (and set via `setDTD:` (page 1915)).

In the data model, an `NSXMLDTD` object is conceptually similar to namespace and attribute nodes: it is not considered to be a child of the `NSXMLDocument` object although it is closely associated with it. It is at the "root" of a shallow tree consisting primarily of nodes representing DTD declarations. Acceptable child nodes are instances of the `NSXMLDTDNode` class as well as `NSXMLNode` objects representing comment nodes and processing-instruction nodes.

You create an `NSXMLDTD` object in one of three ways:

- By processing an XML document with its own internal (in-line) DTD
- By process a standalone (external) DTD
- Programmatically

Once an `NSXMLDTD` instance is in place, you can add, remove, and change the `NSXMLDTDNode` objects representing various DTD declarations. When you write the document out as XML, the new or modified internal DTD is included (assuming you set the DTD in the `NSXMLDocument` instance). You may also programmatically create an external DTD and write that out to its own file.

Tasks

Initializing an NSXMLDTD Object

- `initWithContentsOfURL:options:error:` (page 1927)
  Initializes and returns an `NSXMLDTD` object created from the DTD declarations in a URL-referenced source.
- `initWithData:options:error:` (page 1928)
  Initializes and returns an `NSXMLDTD` object created from the DTD declarations encapsulated in an `NSData` object

Managing DTD Identifiers

- `setPublicID:` (page 1932)
  Sets the public identifier of the receiver.
- `publicID` (page 1930)
  Returns the receiver's public identifier.
- `setSystemID:` (page 1932)
  Sets the system identifier of the receiver.
- `systemID` (page 1933)
  Returns the receiver's system identifier.

Manipulating Child Nodes

- `addChild:` (page 1925)
  Adds a child node to the end of the list of existing children.
- `insertChild:atIndex:` (page 1929)
  Inserts a child node in the receiver's list of children at a specific location in the list.
- `insertChildren:atIndex:` (page 1929)
  Inserts an array of child nodes at a specified location in the receiver's list of children.
- `removeChildAtIndex:` (page 1930)
  Removes the child node at a particular location in the receiver's list of children.
- `replaceChildAtIndex:withNode:` (page 1931)
  Replaces a child at a particular index with another child.
- `setChildren:` (page 1931)
  Removes all existing children of the receiver and replaces them with an array of new child nodes.

Getting DTD Nodes by Name

+ `predefinedEntityDeclarationForName:` (page 1925)
  Returns a DTD node representing the predefined entity declaration with the specified name.

- `elementDeclarationForName:` (page 1926)
  Returns the DTD node representing an element declaration for a specified element.
- `attributeDeclarationForName:elementName:` (page 1926)
  Returns the DTD node representing an attribute-list declaration for a given attribute and its element.
- `entityDeclarationForName:` (page 1927)
  Returns the DTD node representing the entity declaration for a specified entity.
- `notationDeclarationForName:` (page 1930)
  Returns the DTD node representing the notation declaration identified by the specified notation name.

Class Methods

predefinedEntityDeclarationForName:

Returns a DTD node representing the predefined entity declaration with the specified name.

```
+ (NSXMLDTDNode *)predefinedEntityDeclarationForName:(NSString *)name
```

Parameters

name
  A string identifying a predefined entity declaration.

Return Value
An autoreleased `NSXMLDTDNode` object, or `nil` if there is no match for *name*.

Discussion
The five predefined entity references (or character references) are “<” (less-than sign), “>” (greater-than sign), “&” (ampersand), “"” (quotation mark), and “'” (apostrophe).

Availability
Available in Mac OS X v10.4 and later.

See Also
- `entityDeclarationForName:` (page 1927)

Declared In
`NSXMLDTD.h`

Instance Methods

addChild:

Adds a child node to the end of the list of existing children.

```
- (void)addChild:(NSXMLNode *)child
```

Parameters

child

The node object to add to the existing children.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `insertChild:atIndex:` (page 1929)
- `insertChildren:atIndex:` (page 1929)
- `removeChildAtIndex:` (page 1930)
- `replaceChildAtIndex:withNode:` (page 1931)
- `setChildren:` (page 1931)

Declared In
`NSXMLDTD.h`

attributeDeclarationForName:elementName:

Returns the DTD node representing an attribute-list declaration for a given attribute and its element.

```
- (NSXMLDTDNode *)attributeDeclarationForName:(NSString *)attrName
    elementName:(NSString *)elementName
```

Parameters

attrName

A string object identifying the name of an attribute.

elementName

A string object identifying the name of an element.

Return Value
An autoreleased `NSXMLDTDNode` object, or `nil` if there is no matching attribute-list declaration.

Discussion
For example, in the attribute-list declaration:

```
<!ATTLIST person idnum CDATA "0000">
```

“idnum” would correspond to *attrName* and “person” would correspond to *elementName*.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSXMLDTD.h`

elementDeclarationForName:

Returns the DTD node representing an element declaration for a specified element.

```
- (NSXMLDTDNode *)elementDeclarationForName:(NSString *)elementName
```

Parameters

elementName

A string that is the name of an element.

Return Value

An autoreleased `NSXMLDTDNode` object, or `nil` if there is no match.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`NSXMLDTD.h`

entityDeclarationForName:

Returns the DTD node representing the entity declaration for a specified entity.

```
- (NSXMLDTDNode *)entityDeclarationForName:(NSString *)entityName
```

Parameters

entityName

A string that is the name of an entity.

Return Value

An autoreleased `NSXMLDTDNode` object, or `nil` if there is no match.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`NSXMLDTD.h`

initWithContentsOfURL:options:error:

Initializes and returns an `NSXMLDTD` object created from the DTD declarations in a URL-referenced source.

```
- (id)initWithContentsOfURL:(NSURL *)url options:(NSUInteger)mask error:(NSError
    **)error
```

Parameters

url

An `NSURL` object identifying a URL source.

mask

A bit mask specifying input options; bit-OR multiple options. The current valid options are `NSXMLNodePreserveWhitespace` and `NSXMLNodePreserveEntities`; these constants are described in the "Constants" section of the `NSXMLNode` reference.

error

On return, this parameter holds an `NSError` object describing any errors and warnings related to parsing and remote connection.

Return Value

An initialized `NSXMLDTD` object or `nil` if initialization fails because of parsing errors or other reasons.

Discussion
You use this method to create a stand-alone DTD which you can thereafter query and use for validation. You can associate the DTD created through this message with a document by sending `setDTD:` (page 1915) to an `NSXMLDocument` object.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `initWithData:options:error:` (page 1928)
- `validateAndReturnError:` (page 1917) (NSXMLDocument)

Declared In
`NSXMLDTD.h`

initWithData:options:error:

Initializes and returns an `NSXMLDTD` object created from the DTD declarations encapsulated in an `NSData` object

```
- (id)initWithData:(NSData *)data options:(NSUInteger)mask error:(NSError **)error
```

Parameters

data
A data object containing DTD declarations.

mask
A bit mask specifying input options; bit-OR multiple options. The current valid options are `NSXMLNodePreserveWhitespace` and `NSXMLNodePreserveEntities`; these constants are described in the "Constants" section of the `NSXMLNode` reference.

error
On return, this parameter holds an `NSError` object describing any errors and warnings related to parsing and remote connection.

Return Value
An initialized `NSXMLDTD` object or `nil` if initialization fails because of parsing errors or other reasons.

Discussion
This method is the designated initializer for the `NSXMLDTD` class. You use this method to create a stand-alone DTD which you can thereafter query and use for validation. You can associate the DTD created through this message with a document by sending `setDTD:` (page 1915) to an `NSXMLDocument` object.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `initWithContentsOfURL:options:error:` (page 1927)
- `validateAndReturnError:` (page 1917) (NSXMLDocument)

Declared In
`NSXMLDTD.h`

insertChild:atIndex:

Inserts a child node in the receiver's list of children at a specific location in the list.

```
- (void)insertChild:(NSXMLNode *)child atIndex:(NSUInteger)index
```

Parameters

child

An XML-node object that represents the child to insert.

index

An integer identifying the location in the receiver's list of children to insert *child*. The indices of subsequent children in the list are incremented by one.

Availability

Available in Mac OS X v10.4 and later.

See Also

- addChild: (page 1925)
- insertChildren:atIndex: (page 1929)
- removeChildAtIndex: (page 1930)
- replaceChildAtIndex:withNode: (page 1931)
- setChildren: (page 1931)

Declared In

NSXMLDTD.h

insertChildren:atIndex:

Inserts an array of child nodes at a specified location in the receiver's list of children.

```
- (void)insertChildren:(NSArray *)children atIndex:(NSUInteger)index
```

Parameters

children

An array of NSXMLNode objects to insert as children of the receiver.

index

An integer identifying the location in the list of current children to make the insertion. The indices of subsequent children in the list are incremented by the number of inserted children.

Availability

Available in Mac OS X v10.4 and later.

See Also

- addChild: (page 1925)
- insertChild:atIndex: (page 1929)
- removeChildAtIndex: (page 1930)
- replaceChildAtIndex:withNode: (page 1931)
- setChildren: (page 1931)

Declared In

NSXMLDTD.h

notationDeclarationForName:

Returns the DTD node representing the notation declaration identified by the specified notation name.

```
- (NSXMLDTDNode *)notationDeclarationForName:(NSString *)notationName
```

Parameters

notationName

A string that is the name of a notation.

Return Value

An autoreleased `NSXMLDTDNode` object, or `nil` if there is no match.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`NSXMLDTD.h`

publicID

Returns the receiver's public identifier.

```
- (NSString *)publicID
```

Availability

Available in Mac OS X v10.4 and later.

See Also

`- setPublicID:` (page 1932)

Declared In

`NSXMLDTD.h`

removeChildAtIndex:

Removes the child node at a particular location in the receiver's list of children.

```
- (void)removeChildAtIndex:(NSUInteger)index
```

Parameters

index

An integer identifying the child node to remove. The indices of subsequent children in the list are decremented by one.

Discussion

The removed child node is released.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- addChild:` (page 1925)

`- insertChild:atIndex:` (page 1929)

- insertChildren:atIndex: (page 1929)
- replaceChildAtIndex:withNode: (page 1931)
- setChildren: (page 1931)

Declared In
NSXMLDTD.h

replaceChildAtIndex:withNode:

Replaces a child at a particular index with another child.

```
- (void)replaceChildAtIndex:(NSUInteger)index withNode:(NSXMLNode *)node
```

Parameters

index

An integer identifying the position of a node in the receiver's list of child nodes.

node

An NSXMLNode object to replace the object at *index*.

Discussion
The replaced child node is released.

Availability
Available in Mac OS X v10.4 and later.

See Also
- addChild: (page 1925)
- insertChild:atIndex: (page 1929)
- insertChildren:atIndex: (page 1929)
- removeChildAtIndex: (page 1930)
- setChildren: (page 1931)

Declared In
NSXMLDTD.h

setChildren:

Removes all existing children of the receiver and replaces them with an array of new child nodes.

```
- (void)setChildren:(NSArray *)children
```

Parameters

children

An array of NSXMLNode objects. To remove all existing children, pass in nil.

Discussion
Replaced or removed child nodes are released.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `addChild:` (page 1925)
- `insertChild:atIndex:` (page 1929)
- `insertChildren:atIndex:` (page 1929)
- `removeChildAtIndex:` (page 1930)
- `replaceChildAtIndex:withNode:` (page 1931)

Declared In
`NSXMLDTD.h`

setPublicID:

Sets the public identifier of the receiver.

```
- (void)setPublicID:(NSString *)publicID
```

Parameters

publicID
A string object specifying a public identifier.

Discussion
This identifier should be in the default catalog in `/etc/xml/catalog` or in a path specified by the environment variable `XML_CATALOG_FILES`. When the public ID is set the system ID must also be set.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `publicID` (page 1930)
- `setSystemID:` (page 1932)

Declared In
`NSXMLDTD.h`

setSystemID:

Sets the system identifier of the receiver.

```
- (void)setSystemID:(NSString *)systemID
```

Parameters

systemID
A string object that encapsulates a URL locating a valid DTD.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `systemID` (page 1933)

Declared In
`NSXMLDTD.h`

systemID

Returns the receiver's system identifier.

```
- (NSString *)systemID
```

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setSystemID:` (page 1932)

Declared In
`NSXMLDTD.h`

NSXMLDTDNode Class Reference

Inherits from	NSXMLNode : NSObject
Conforms to	NSCopying (NSXMLNode) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/Foundation.h
Availability	Available in Mac OS X v10.4 and later.
Companion guide	Tree-Based XML Programming Guide for Cocoa

Overview

Instances of the `NSXMLDTDNode` class represent element, attribute-list, entity, and notation declarations in a Document Type Definition. `NSXMLDTDNode` objects are the sole children of a `NSXMLDTD` object (possibly along with comment nodes and processing-instruction nodes). They themselves cannot have any children.

`NSXMLDTDNode` objects can be of four kinds—element, attribute-list, entity, or notation declaration—and can also be of a subkind, as specified by a `NSXMLDTDNodeKind` constant. For example, a DTD entity-declaration node could represent an unparsed entity declaration (`NSXMLEntityUnparsedKind`) rather than a parameter entity declaration (`NSXMLEntityParameterKind`). You can use a DTD node's subkind to help determine how to handle the value of the node.

You can create an `NSXMLDTDNode` object with the `initWithXMLString:` (page 1937) method, the `NSXMLNode` class method `DTDNodeWithXMLString:` (page 1970), or with the NSXMLNode initializer `initWithKind:options:` (page 1979) (in the latter method supplying the appropriate `NSXMLNodeKind` constant).

Setting the object value or string value of an `NSXMLDTDNode` objects affects different parts of different kinds of declaration. See the related programming topic for more information.

Tasks

Initializing an NSXMLDTDNode Object

`- initWithXMLString:` (page 1937)
Returns an `NSXMLDTDNode` object initialized with the DTD declaration in a given string.

Managing the DTD Node Kind

- `DTDKind` (page 1936)
  Returns the receiver's DTD kind.
- `setDTDKind:` (page 1938)
  Sets the receiver's DTD kind.

Managing DTD Identifiers

- `isExternal` (page 1937)
  Returns a Boolean value that indicates whether the receiver represents a declaration from an external DTD (the system ID is set).
- `setNotationName:` (page 1939)
  Sets the notation name associated with the receiver.
- `notationName` (page 1938)
  Returns the name of the notation associated with the receiver.
- `setPublicID:` (page 1939)
  Sets the public identifier associated with the receiver.
- `publicID` (page 1938)
  Returns the public identifier associated with the receiver.
- `setSystemID:` (page 1940)
  Sets the system identifier associated with the receiver.
- `systemID` (page 1940)
  Returns the system identifier associated with the receiver.

Instance Methods

DTDKind

Returns the receiver's DTD kind.

```
- (NSXMLDTDNodeKind)DTDKind
```

Return Value
The receiver's DTD kind. See "Constants" (page 1940) for a list of valid `NSXMLDTDNodeKind` constants.

Discussion
The DTD kind is distinct from a `NSXMLDTDNode` object's node kind (returned by the `NSXMLNode` `kind` (page 1980) method).

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setDTDKind:` (page 1938)

Declared In
NSXMLDTDNode.h

initWithXMLString:

Returns an NSXMLDTDNode object initialized with the DTD declaration in a given string.

```
- (id)initWithXMLString:(NSString *)string
```

Parameters
string
The DTD declaration.

Return Value
An NSXMLDTDNode object initialized with the DTD declaration in *string*. Returns nil if initialization did not succeed, as might occur if the passed-in declaration is malformed.

Discussion
The node kind (NSXMLNode) assigned to the returned object—element, attribute, entity, or notation declaration— is based on the full XML string that is parsed. To assign a subkind, use the setDTDKind: (page 1938) method.

You may also use the DTDNodeWithXMLString: (page 1970) or initWithKind: (page 1979) methods to create NSXMLDTDNode instances. However, you cannot use the latter method to create NSXMLDTDNode instances for attribute-list declarations.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSXMLDTDNode.h

isExternal

Returns a Boolean value that indicates whether the receiver represents a declaration from an external DTD (the system ID is set).

```
- (BOOL)isExternal
```

Return Value
YES if receiver represents a declaration from an external DTD (the system ID is set), otherwise NO.

Discussion
This method is valid only for objects representing entities and notations.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setSystemID: (page 1940)

Declared In
NSXMLDTDNode.h

notationName

Returns the name of the notation associated with the receiver.

```
- (NSString *)notationName
```

Return Value
The name of the notation associated with the receiver.

Discussion
Notations are applicable to unparsed external entities, processing instructions, and some attribute values.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setNotationName:` (page 1939)

Declared In
`NSXMLDTDNode.h`

publicID

Returns the public identifier associated with the receiver.

```
- (NSString *)publicID
```

Return Value
The public identifier associated with the receiver.

Discussion
The public ID is applicable to entities and notations.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSXMLDTDNode.h`

setDTDKind:

Sets the receiver's DTD kind.

```
- (void)setDTDKind:(NSXMLDTDNodeKind)kind
```

Parameters
kind
The receiver's DTD kind. See "Constants" (page 1940) for a list of valid `NSXMLDTDNodeKind` constants.

Discussion
The DTD kind is a finer grain of an `NSXMLDTDNode` object's node kind (returned by the `NSXMLNode kind` (page 1980) method).

Availability
Available in Mac OS X v10.4 and later.

See Also
- `DTDKind` (page 1936)

Declared In
`NSXMLDTDNode.h`

setNotationName:

Sets the notation name associated with the receiver.

```
- (void)setNotationName:(NSString *)notationName
```

Parameters

notationName
The notation name associated with the receiver.

Discussion
This method is valid for entities only.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `notationName` (page 1938)

Declared In
`NSXMLDTDNode.h`

setPublicID:

Sets the public identifier associated with the receiver.

```
- (void)setPublicID:(NSString *)publicID
```

Parameters

publicID
The public identifier associated with the receiver. This identifier should be in the default catalog in `/etc/xml/catalog` or in a path specified by the environment variable `XML_CATALOG_FILES`.

Discussion
This method is valid for entities and notations only. When the public ID is set the system ID must also be set.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `publicID` (page 1938)
- `setSystemID:` (page 1940)

Declared In
`NSXMLDTDNode.h`

setSystemID:

Sets the system identifier associated with the receiver.

```
- (void)setSystemID:(NSString *)systemID
```

Parameters

systemID

The system identifier associated with the receiver. This value must be a valid URI.

Availability
Available in Mac OS X v10.4 and later.

See Also
- systemID (page 1940)

Declared In
NSXMLDTDNode.h

systemID

Returns the system identifier associated with the receiver.

```
- (NSString *)systemID
```

Return Value
The system identifier associated with the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setSystemID: (page 1940)

Declared In
NSXMLDTDNode.h

Constants

NSXMLDTDNodeKind

The type defined for the constants that specify the kind and subkind of DTD declaration represented by an NSXMLDTDNode object. You set the DTD-node kind using the setDTDKind: (page 1938) method.

```
typedef NSUInteger NSXMLDTDNodeKind;
```

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSXMLDTDNode.h

DTD Node Kind Constants

Constants that specify the kind and subkind of DTD declaration represented by an `NSXMLDTDNode` object. You set the DTD-node kind using the `setDTDKind:` (page 1938) method.

```
enum {
        NSXMLEntityGeneralKind = 1,
        NSXMLEntityParsedKind,
        NSXMLEntityUnparsedKind,
        NSXMLEntityParameterKind,
        NSXMLEntityPredefined,

        NSXMLAttributeCDATAKind,
        NSXMLAttributeIDKind,
        NSXMLAttributeIDRefKind,
        NSXMLAttributeIDRefsKind,
        NSXMLAttributeEntityKind,
        NSXMLAttributeEntitiesKind,
        NSXMLAttributeNMTokenKind,
        NSXMLAttributeNMTokensKind,
        NSXMLAttributeEnumerationKind,
        NSXMLAttributeNotationKind,

        NSXMLElementDeclarationUndefinedKind,
        NSXMLElementDeclarationEmptyKind,
        NSXMLElementDeclarationAnyKind,
        NSXMLElementDeclarationMixedKind,
        NSXMLElementDeclarationElementKind
};
```

Constants

`NSXMLEntityGeneralKind`
Identifies a general entity declaration.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLDTDNode.h`.

`NSXMLEntityParsedKind`
Identifies a parsed entity declaration.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLDTDNode.h`.

`NSXMLEntityUnparsedKind`
Identifies an unparsed entity declaration.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLDTDNode.h`.

`NSXMLEntityParameterKind`
Identifies a parameter entity declaration.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLDTDNode.h`.

`NSXMLEntityPredefined`
Identifies a predefined entity declaration.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLDTDNode.h`.

`NSXMLAttributeCDATAKind`
Identifies an attribute-list declaration with a `CDATA` (character data) value type.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLDTDNode.h`.

`NSXMLAttributeIDKind`
Identifies an attribute-list declaration with an `ID` value type (per-document unique element name).

Available in Mac OS X v10.4 and later.

Declared in `NSXMLDTDNode.h`.

`NSXMLAttributeIDRefKind`
Identifies an attribute-list declaration with an `IDREF` value type (refers to element `ID` type).

Available in Mac OS X v10.4 and later.

Declared in `NSXMLDTDNode.h`.

`NSXMLAttributeIDRefsKind`
Identifies an attribute-list declaration with an `IDREFS` value type (refers to multiple elements of `ID` type).

Available in Mac OS X v10.4 and later.

Declared in `NSXMLDTDNode.h`.

`NSXMLAttributeEntityKind`
Identifies an attribute-list declaration with an `ENTITY` value type (refers to unparsed entity declared in document).

Available in Mac OS X v10.4 and later.

Declared in `NSXMLDTDNode.h`.

`NSXMLAttributeEntitiesKind`
Identifies an attribute-list declaration with an `ENTITIES` value type (refers to multiple unparsed entities declared elsewhere in document).

Available in Mac OS X v10.4 and later.

Declared in `NSXMLDTDNode.h`.

`NSXMLAttributeNMTokenKind`
Identifies an attribute-list declaration with a `NMTOKEN` value type (name token).

Available in Mac OS X v10.4 and later.

Declared in `NSXMLDTDNode.h`.

`NSXMLAttributeNMTokensKind`
Identifies an attribute-list declaration with a `NMTOKENS` value type (multiple name tokens)

Available in Mac OS X v10.4 and later.

Declared in `NSXMLDTDNode.h`.

`NSXMLAttributeEnumerationKind`
Identifies an attribute-list declaration with an enumeration value type (list of all possible values).

Available in Mac OS X v10.4 and later.

Declared in `NSXMLDTDNode.h`.

`NSXMLAttributeNotationKind`
Identifies an attribute-list declaration with a `NOTATION` value type (name of declared notation).

Available in Mac OS X v10.4 and later.

Declared in `NSXMLDTDNode.h`.

`NSXMLElementDeclarationUndefinedKind`
Identifies an undefined element declaration.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLDTDNode.h`.

`NSXMLElementDeclarationEmptyKind`
Identifies a declaration (`EMPTY`) of an empty element.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLDTDNode.h`.

`NSXMLElementDeclarationAnyKind`
Identifies an `ANY` element declaration.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLDTDNode.h`.

`NSXMLElementDeclarationMixedKind`
Identifies a declaration of an element with mixed content (`(#PCDATA | child)`).

Available in Mac OS X v10.4 and later.

Declared in `NSXMLDTDNode.h`.

`NSXMLElementDeclarationElementKind`
Identifies a declaration of an element with child elements.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLDTDNode.h`.

Declared In
`NSXMLDTDNode.h`

NSXMLElement Class Reference

Inherits from	NSXMLNode : NSObject
Conforms to	NSCopying (NSXMLNode) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSXMLElement.h
Availability	Available in Mac OS X v10.4 and later.
Companion guide	Tree-Based XML Programming Guide for Cocoa
Related sample code	AlbumToSlideshow Core Data HTML Store NewsReader TimelineToTC

Overview

Instances of the `NSXMLElement` class represent element nodes in an XML tree structure. An `NSXMLElement` object may have child nodes, specifically comment nodes, processing-instruction nodes, text nodes, and other `NSXMLElement` nodes. It may also have attribute nodes and namespace nodes associated with it (however, namespace and attribute nodes are not considered children). Any attempt to add a `NSXMLDocument` node, `NSXMLDTD` node, namespace node, or attribute node as a child raises an exception. If you add a child node to an `NSXMLElement` object and that child already has a parent, `NSXMLElement` raises an exception; the child must be detached or copied first.

Subclassing Notes

You can subclass `NSXMLElement` if you want element nodes with more specialized attributes or behavior, for example, paragraph and font attributes that specify how the string value of the element should appear.

Methods to Override

To subclass `NSXMLElement` you need to override the primary initializer, `initWithName:URI:` (page 1953), and the methods listed below. In most cases, you need only invoke the superclass implementation, adding any subclass-specific code before or after the invocation, as necessary.

addAttribute: (page 1948)	removeNamespaceForPrefix: (page 1958)
removeAttributeForName: (page 1957)	setNamespaces: (page 1962)
setAttributes: (page 1960)	namespaces (page 1956)
attributeForLocalName:URI: (page 1950)	insertChild:atIndex: (page 1954)
attributes (page 1951)	removeChildAtIndex: (page 1958)
addNamespace: (page 1949)	setChildren: (page 1961)

By default NSXMLElement implements the NSObject isEqual: (page 2101) method to perform a deep comparison: two NSXMLDocument objects are not considered equal unless they have the same name, same child nodes, same attributes, and so on. If you want a different standard of comparison, override isEqual:.

Special Considerations

Because of the architecture and data model of NSXML, when it parses and processes a source of XML it cannot know about your subclass unless you override the class method replacementClassForClass: (page 1903) to return your custom class in place of an NSXML class. If your custom class has no direct NSXML counterpart—for example, it is a subclass of NSXMLNode that represents CDATA sections—then you can walk the tree after it has been created and insert the new node where appropriate.

Note that you can safely set the root element of the XML document (using the NSXMLDocument setRootElement: (page 1915)method) to be an instance of your subclass since this method only checks to see if the added node is of an element kind (NSXMLElementKind). These precautions do not apply, of course, if you are creating an XML tree programmatically.

Tasks

Initializing NSXMLElement Objects

- initWithName: (page 1952)
  Returns an NSXMLElement object initialized with the specified name.
- initWithName:stringValue: (page 1953)
  Returns an NSXMLElement object initialized with a specified name and a single text-node child containing a specified value.
- initWithXMLString:error: (page 1954)
  Returns an NSXMLElement object created from a specified string containing XML markup.
- initWithName:URI: (page 1953)
  Returns an NSXMLElement object initialized with the specified name and URI.

Obtaining Child Elements

- `elementsForName:` (page 1952)
  Returns the child element nodes (as `NSXMLElement` objects) of the receiver that have a specified name.
- `elementsForLocalName:URI:` (page 1951)
  Returns the child element nodes (as `NSXMLElement` objects) of the receiver that are matched with the specified local name and URI.

Manipulating Child Elements

- `addChild:` (page 1949)
  Adds a child node at the end of the receiver's current list of children.
- `insertChild:atIndex:` (page 1954)
  Inserts a new child node at a specified location in the receiver's list of child nodes.
- `insertChildren:atIndex:` (page 1955)
  Inserts an array of child nodes at a specified location in the receiver's list of children.
- `removeChildAtIndex:` (page 1958)
  Removes the child node of the receiver identified by a given index.
- `replaceChildAtIndex:withNode:` (page 1959)
  Replaces a child node at a specified location with another child node.
- `setChildren:` (page 1961)
  Sets all child nodes of the receiver at once, replacing any existing children.
- `normalizeAdjacentTextNodesPreservingCDATA:` (page 1957)
  Coalesces adjacent text nodes of the receiver that you have explicitly added, optionally including CDATA sections.

Handling Attributes

- `addAttribute:` (page 1948)
  Adds an attribute node to the receiver.
- `attributeForName:` (page 1950)
  Returns the attribute node of the receiver with the specified name.
- `attributeForLocalName:URI:` (page 1950)
  Returns the attribute node of the receiver that is identified by a local name and URI.
- `attributes` (page 1951)
  Returns the receiver's attributes
- `removeAttributeForName:` (page 1957)
  Removes an attribute node that is identified by its name.
- `setAttributes:` (page 1960)
  Sets all attributes of the receiver at once, replacing any existing attribute nodes.
- `setAttributesAsDictionary:` (page 1961)
  Sets the attributes of the receiver based on the key-value pairs specified in the passed-in dictionary.

Handling Namespaces

- `addNamespace:` (page 1949)
  Adds a namespace node to the receiver.
- `namespaces` (page 1956)
  Returns the namespace nodes of the receiver.
- `namespaceForPrefix:` (page 1956)
  Returns the namespace node with a specified prefix.
- `removeNamespaceForPrefix:` (page 1958)
  Removes a namespace node that is identified by a given prefix.
- `resolveNamespaceForName:` (page 1959)
  Returns the namespace node with the prefix matching the given qualified name.
- `resolvePrefixForNamespaceURI:` (page 1960)
  Returns the prefix associated with the specified URI.
- `setNamespaces:` (page 1962)
  Sets all of the namespace nodes of the receiver at once, replacing any existing namespace nodes.

Instance Methods

addAttribute:

Adds an attribute node to the receiver.

```
- (void)addAttribute:(NSXMLNode *)anAttribute
```

Parameters

anAttribute
An XML node object representing an attribute. If the receiver already has an attribute with the same name, *anAttribute* is not added.

Discussion
The order of multiple attributes is preserved if the `NSXMLPreserveAttributeOrder` option is specified when the element is created.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `attributeForName:` (page 1950)
- `attributes` (page 1951)
- `removeAttributeForName:` (page 1957)
- `setAttributes:` (page 1960)

Related Sample Code
AlbumToSlideshow
Core Data HTML Store

Declared In
NSXMLElement.h

addChild:

Adds a child node at the end of the receiver's current list of children.

```
- (void)addChild:(NSXMLNode *)child
```

Parameters

child

An XML node object to add to the receiver's children.

Discussion
The new node has an index value that is one greater than the last of the current children.

Availability
Available in Mac OS X v10.4 and later.

See Also
- insertChild:atIndex: (page 1954)
- removeChildAtIndex: (page 1958)
- replaceChildAtIndex:withNode: (page 1959)
- setChildren: (page 1961)

Related Sample Code
AlbumToSlideshow
Core Data HTML Store

Declared In
NSXMLElement.h

addNamespace:

Adds a namespace node to the receiver.

```
- (void)addNamespace:(NSXMLNode *)aNamespace
```

Parameters

aNamespace

An XML node object of kind NSXMLNamespaceKind. If the receiver already has a namespace with the same name, *aNamespace* is not added.

Availability
Available in Mac OS X v10.4 and later.

See Also
- namespaces (page 1956)
- namespaceForPrefix: (page 1956)
- removeNamespaceForPrefix: (page 1958)
- resolveNamespaceForName: (page 1959)
- resolvePrefixForNamespaceURI: (page 1960)

- setNamespaces: (page 1962)

Declared In
NSXMLElement.h

attributeForLocalName:URI:

Returns the attribute node of the receiver that is identified by a local name and URI.

```
- (NSXMLNode *)attributeForLocalName:(NSString *)localName URI:(NSString *)URI
```

Parameters

localName

A string specifying the local name of an attribute.

URI

A sting identifying the URI associated with an attribute.

Return Value
An XML node object representing a matching attribute or nil if no such node was found.

Availability
Available in Mac OS X v10.4 and later.

See Also
- attributeForName: (page 1950)
- attributes (page 1951)
- removeAttributeForName: (page 1957)
- setAttributes: (page 1960)

Declared In
NSXMLElement.h

attributeForName:

Returns the attribute node of the receiver with the specified name.

```
- (NSXMLNode *)attributeForName:(NSString *)name
```

Parameters

name

A string specifying the name of an attribute.

Return Value
An XML node object representing a matching attribute or nil if no such node was found.

Discussion
If *name* is a qualified name, then this method invokes attributeForLocalName:URI: (page 1950) with the URI parameter set to the URI associated with the prefix. Otherwise comparison is based on string equality of the qualified or non-qualified name.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `attributes` (page 1951)
- `removeAttributeForName:` (page 1957)
- `setAttributes:` (page 1960)

Related Sample Code
Core Data HTML Store

Declared In
`NSXMLElement.h`

attributes

Returns the receiver's attributes

```
- (NSArray *)attributes
```

Return Value
An array of `NSXMLNode` objects of kind `NSXMLAttributeKind` or `nil` if the receiver has no attribute nodes.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `attributeForLocalName:URI:` (page 1950)
- `attributeForName:` (page 1950)
- `removeAttributeForName:` (page 1957)
- `setAttributes:` (page 1960)

Declared In
`NSXMLElement.h`

elementsForLocalName:URI:

Returns the child element nodes (as `NSXMLElement` objects) of the receiver that are matched with the specified local name and URI.

```
- (NSArray *)elementsForLocalName:(NSString *)localName URI:(NSString *)URI
```

Parameters
localName
A string specifying a local name of an element.

URI
A string specifying a URI associated with an element.

Return Value
An array of `NSXMLElement` objects or nil if no matching children could be found.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `elementsForName:` (page 1952)

Declared In
`NSXMLElement.h`

elementsForName:

Returns the child element nodes (as `NSXMLElement` objects) of the receiver that have a specified name.

```
- (NSArray *)elementsForName:(NSString *)name
```

Parameters

name

A string specifying the name of the child element nodes to find and return. If *name* is a qualified name, then this method invokes `elementsForLocalName:URI:` (page 1951) with the URI parameter set to the URI associated with the prefix. Otherwise comparison is based on string equality of the qualified or non-qualified name.

Return Value
An array of of `NSXMLElement` objects or an empty array if no matching children can be found.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
Core Data HTML Store
NewsReader

Declared In
`NSXMLElement.h`

initWithName:

Returns an `NSXMLElement` object initialized with the specified name.

```
- (id)initWithName:(NSString *)name
```

Parameters

name

A string specifying the name of the element.

Return Value
The initialized `NSXMLElement` object or `nil` if initialization did not succeed.

Discussion
The XML string representation of this object is <*name*></*name*>. This method invokes `initWithName:URI:` (page 1953) with the URI parameter set to `nil`.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `initWithName:stringValue:` (page 1953)

- initWithXMLString:error: (page 1954)

Declared In
NSXMLElement.h

initWithName:stringValue:

Returns an NSXMLElement object initialized with a specified name and a single text-node child containing a specified value.

```
- (id)initWithName:(NSString *)name stringValue:(NSString *)string
```

Parameters

name

A string specifying the name of the element.

string

The string value of the receiver's text node.

Return Value
The initialized NSXMLElement object or nil if initialization did not succeed.

Discussion
The string representation of this object is <*name*>*string*</*name*>.

Availability
Available in Mac OS X v10.4 and later.

See Also
- initWithName:URI: (page 1953)
- initWithName: (page 1952)
- initWithXMLString:error: (page 1954)

Declared In
NSXMLElement.h

initWithName:URI:

Returns an NSXMLElement object initialized with the specified name and URI.

```
- (id)initWithName:(NSString *)name URI:(NSString *)URI
```

Parameters

name

A string that specifies the qualified name of the element.

URI

A string that specifies the namespace URI associated with the element.

Return Value
The initialized NSXMLElement object or nil if initialization did not succeed.

Discussion
You can look up the namespace prefix for this element node based on its URI using `resolvePrefixForNamespaceURI:` (page 1960). This method is the primary initializer for the `NSXMLElement` class.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `initWithName:` (page 1952)
- `initWithName:stringValue:` (page 1953)
- `initWithXMLString:error:` (page 1954)

Declared In
`NSXMLElement.h`

initWithXMLString:error:

Returns an `NSXMLElement` object created from a specified string containing XML markup.

```
- (id)initWithXMLString:(NSString *)string error:(NSError **)error
```

Parameters
string
A string containing XML markup for an element.

error
On return, an `NSError` object that describes any errors or warnings resulting from the parsing of the markup.

Return Value
The initialized `NSXMLElement` object or `nil` if initialization did not succeed.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `initWithName:URI:` (page 1953)
- `initWithName:` (page 1952)
- `initWithName:stringValue:` (page 1953)

Declared In
`NSXMLElement.h`

insertChild:atIndex:

Inserts a new child node at a specified location in the receiver's list of child nodes.

```
- (void)insertChild:(NSXMLNode *)child atIndex:(NSUInteger)index
```

Parameters
child
An XML node object to be inserted as a child of the receiver.

index
An integer identifying a position in the receiver's list of children. An exception is raised if *index* is out of bounds.

Discussion
Insertion of the node increments the indexes of sibling nodes after it.

Availability
Available in Mac OS X v10.4 and later.

See Also
- addChild: (page 1949)
- insertChildren:atIndex: (page 1955)
- removeChildAtIndex: (page 1958)
- replaceChildAtIndex:withNode: (page 1959)
- setChildren: (page 1961)

Declared In
NSXMLElement.h

insertChildren:atIndex:

Inserts an array of child nodes at a specified location in the receiver's list of children.

```
- (void)insertChildren:(NSArray *)children atIndex:(NSUInteger)index
```

Parameters
children
An array of XML node objects to add as children of the receiver.

index
An integer identifying a position in the receiver's list of children. An exception is raised if *index* is out of bounds.

Discussion
Insertion of the node increases the indexes of sibling nodes after it by the count of *children*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- addChild: (page 1949)
- insertChild:atIndex: (page 1954)
- removeChildAtIndex: (page 1958)
- replaceChildAtIndex:withNode: (page 1959)
- setChildren: (page 1961)

Declared In
NSXMLElement.h

namespaceForPrefix:

Returns the namespace node with a specified prefix.

```
- (NSXMLNode *)namespaceForPrefix:(NSString *)name
```

Parameters

name

A string specifying a namespace prefix.

Return Value

An `NSXMLNode` object of kind `NSXMLNamespaceKind` or `nil` if there is no namespace node with that prefix.

Availability

Available in Mac OS X v10.4 and later.

See Also

- `addNamespace:` (page 1949)
- `namespaces` (page 1956)
- `removeNamespaceForPrefix:` (page 1958)
- `resolveNamespaceForName:` (page 1959)
- `resolvePrefixForNamespaceURI:` (page 1960)
- `setNamespaces:` (page 1962)

Declared In

`NSXMLElement.h`

namespaces

Returns the namespace nodes of the receiver.

```
- (NSArray *)namespaces
```

Return Value

An array of `NSXMLNode` objects of kind `NSXMLNamespaceKind`. Returns `nil` if the receiver has no namespace nodes.

Availability

Available in Mac OS X v10.4 and later.

See Also

- `addNamespace:` (page 1949)
- `namespaceForPrefix:` (page 1956)
- `removeNamespaceForPrefix:` (page 1958)
- `resolveNamespaceForName:` (page 1959)
- `resolvePrefixForNamespaceURI:` (page 1960)
- `setNamespaces:` (page 1962)

Declared In

`NSXMLElement.h`

normalizeAdjacentTextNodesPreservingCDATA:

Coalesces adjacent text nodes of the receiver that you have explicitly added, optionally including CDATA sections.

```
- (void)normalizeAdjacentTextNodesPreservingCDATA:(BOOL)preserve
```

Parameters

preserve

YES if CDATA sections are left alone as text nodes, NO otherwise.

Discussion

A text node with a value of an empty string is removed. When you process an input source of XML, adjacent text nodes are automatically normalized. You should invoke this method (with *preserve* as NO) before using the NSXMLNode methods objectsForXQuery:constants:error: (page 1983) or nodesForXPath:error: (page 1982).

Availability

Available in Mac OS X v10.4 and later.

See Also

- setChildren: (page 1961)

Declared In

NSXMLElement.h

removeAttributeForName:

Removes an attribute node that is identified by its name.

```
- (void)removeAttributeForName:(NSString *)attrName
```

Parameters

attrName

A string specifying the name of an attribute.

Discussion

The removed XML node object is released.

Availability

Available in Mac OS X v10.4 and later.

See Also

- addAttribute: (page 1948)
- attributeForName: (page 1950)
- attributes (page 1951)
- removeAttributeForName: (page 1957)
- setAttributes: (page 1960)

Declared In

NSXMLElement.h

removeChildAtIndex:

Removes the child node of the receiver identified by a given index.

```
- (void)removeChildAtIndex:(NSUInteger)nodeIndex
```

Parameters

nodeIndex

An integer identifying the node in the receiver's list of children to remove. An exception is raised if *index* is out of bounds.

Discussion

The XML node object is released upon removal. The indices of subsequent children are decremented by one.

Availability

Available in Mac OS X v10.4 and later.

See Also

- addChild: (page 1949)
- insertChild:atIndex: (page 1954)
- replaceChildAtIndex:withNode: (page 1959)
- setChildren: (page 1961)

Related Sample Code

Core Data HTML Store

Declared In

NSXMLElement.h

removeNamespaceForPrefix:

Removes a namespace node that is identified by a given prefix.

```
- (void)removeNamespaceForPrefix:(NSString *)name
```

Parameters

name

A string that is the prefix for a namespace.

Discussion

The removed XML node object is removed.

Availability

Available in Mac OS X v10.4 and later.

See Also

- addNamespace: (page 1949)
- namespaces (page 1956)
- namespaceForPrefix: (page 1956)
- setNamespaces: (page 1962)

Declared In

NSXMLElement.h

replaceChildAtIndex:withNode:

Replaces a child node at a specified location with another child node.

```
- (void)replaceChildAtIndex:(NSUInteger)index withNode:(NSXMLNode *)node
```

Parameters

index

An integer identifying a position in the receiver's list of children. An exception is raised if *index* is out of bounds.

node

An XML node object that will replace the current child.

Discussion

The replaced XML node object is released upon removal.

Availability

Available in Mac OS X v10.4 and later.

See Also

- `addChild:` (page 1949)
- `insertChild:atIndex:` (page 1954)
- `insertChildren:atIndex:` (page 1955)
- `removeChildAtIndex:` (page 1958)
- `setChildren:` (page 1961)

Declared In

`NSXMLElement.h`

resolveNamespaceForName:

Returns the namespace node with the prefix matching the given qualified name.

```
- (NSXMLNode *)resolveNamespaceForName:(NSString *)name
```

Parameters

name

A string that is the qualified name for a namespace (a qualified name is prefix plus local name).

Return Value

An `NSXMLNode` object of kind `NSXMLNamespaceKind` or `nil` if there is no matching namespace node.

Discussion

The method looks in the entire namespace chain for the prefix.

Availability

Available in Mac OS X v10.4 and later.

See Also

- `addNamespace:` (page 1949)
- `namespaces` (page 1956)
- `namespaceForPrefix:` (page 1956)
- `resolvePrefixForNamespaceURI:` (page 1960)

- setNamespaces: (page 1962)

Declared In
NSXMLElement.h

resolvePrefixForNamespaceURI:

Returns the prefix associated with the specified URI.

- (NSString *)resolvePrefixForNamespaceURI:(NSString *)*namespaceURI*

Parameters

namespaceURI

A string identifying the URI associated with the namespace.

Return Value
A string that is the matching prefix or nil if it finds no matching prefix.

Discussion
The method looks in the entire namespace chain for the URI.

Availability
Available in Mac OS X v10.4 and later.

See Also
- addNamespace: (page 1949)
- namespaces (page 1956)
- namespaceForPrefix: (page 1956)
- resolveNamespaceForName: (page 1959)
- setNamespaces: (page 1962)

Declared In
NSXMLElement.h

setAttributes:

Sets all attributes of the receiver at once, replacing any existing attribute nodes.

- (void)setAttributes:(NSArray *)*attributes*

Parameters

attributes

An array of NSXMLNode objects of kind NSXMLAttributeKind. If there are attribute nodes with the same name, the first attribute with that name is used. Send this message with *attributes* as nil to remove all attributes.

Discussion
To set attributes in an element node using an NSDictionary object as the input parameter, see setAttributesAsDictionary: (page 1961).

Availability
Available in Mac OS X v10.4 and later.

See Also
- `addAttribute:` (page 1948)
- `attributeForName:` (page 1950)
- `attributes` (page 1951)
- `removeAttributeForName:` (page 1957)

Declared In
`NSXMLElement.h`

setAttributesAsDictionary:

Sets the attributes of the receiver based on the key-value pairs specified in the passed-in dictionary.

```
- (void)setAttributesAsDictionary:(NSDictionary *)attributes
```

Parameters

attributes

A dictionary of key-value pairs where the attribute name is the key and the object value of the attribute is the dictionary value.

Discussion
The method uses these names and object values to create `NSXMLNode` objects of kind `NSXMLAttributeKind`. Existing attributes are removed.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `addAttribute:` (page 1948)
- `attributes` (page 1951)
- `removeAttributeForName:` (page 1957)
- `setAttributes:` (page 1960)

Declared In
`NSXMLElement.h`

setChildren:

Sets all child nodes of the receiver at once, replacing any existing children.

```
- (void)setChildren:(NSArray *)children
```

Parameters

children

An array of `NSXMLElement` objects or `NSXMLNode` objects of kinds `NSXMLElementKind`, `NSXMLProcessingInstructionKind`, `NSXMLTextKind`, or `NSXMLCommentKind`.

Discussion
Send this message with *children* as `nil` to remove all child nodes.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `- insertChild:atIndex:` (page 1954)
- `- insertChildren:atIndex:` (page 1955)
- `- removeChildAtIndex:` (page 1958)
- `- replaceChildAtIndex:withNode:` (page 1959)
- `- setChildren:` (page 1961)

Related Sample Code
Core Data HTML Store

Declared In
`NSXMLElement.h`

setNamespaces:

Sets all of the namespace nodes of the receiver at once, replacing any existing namespace nodes.

```
- (void)setNamespaces:(NSArray *)namespaces
```

Parameters

namespaces

An array of `NSXMLNode` objects of kind `NSXMLNamespaceKind`. If there are namespace nodes with the same prefix, the first attribute with that prefix is used. Send this message with *namespaces* as `nil` to remove all namespace nodes.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `- addNamespace:` (page 1949)
- `- namespaces` (page 1956)
- `- namespaceForPrefix:` (page 1956)
- `- removeNamespaceForPrefix:` (page 1958)
- `- resolveNamespaceForName:` (page 1959)
- `- resolvePrefixForNamespaceURI:` (page 1960)

Declared In
`NSXMLElement.h`

NSXMLNode Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	NSXML/NSXMLNode.h
Availability	Available in Mac OS X v10.4 and later.
Companion guide	Tree-Based XML Programming Guide for Cocoa
Related sample code	AlbumToSlideshow CocoaSOAP Core Data HTML Store TimelineToTC

Overview

Objects of the `NSXMLNode` class are nodes in the abstract, logical tree structure that represents an XML document. Node objects can be of different kinds, corresponding to the following markup constructs in an XML document: element, attribute, text, processing instruction, namespace, and comment. In addition, a document-node object (specifically, an instance of `NSXMLDocument`) represents an XML document in its entirety. `NSXMLNode` objects can also represent document type declarations as well as declarations in Document Type Definitions (DTDs). Class factory methods of `NSXMLNode` enable you to create nodes of each kind. Only document, element, and DTD nodes may have child nodes.

Among the NSXML family of classes—that is, the Foundation classes with the prefix "NSXML" (excluding `NSXMLParser`)—the `NSXMLNode` class is the base class. Inheriting from it are the classes `NSXMLElement`, `NSXMLDocument`, `NSXMLDTD`, and `NSXMLDTDNode`. `NSXMLNode` specifies the interface common to all XML node objects and defines common node behavior and attributes, for example hierarchy level, node name and value, tree traversal, and the ability to emit representative XML markup text.

Subclassing Notes

You can subclass `NSXMLNode` if you want nodes of kinds different from the supported ones, You can also create a subclass with more specialized attributes or behavior than `NSXMLNode`.

Methods to Override

To subclass `NSXMLNode` you need to override the primary initializer, `initWithKind:options:` (page 1979), and the methods listed below. In most cases, you need only invoke the superclass implementation, adding any subclass-specific code before or after the invocation, as necessary.

`kind` (page 1980)	`parent` (page 1985)
`name` (page 1981)	`childAtIndex:` (page 1976)
`setName:` (page 1987)	`childCount` (page 1977)
`objectValue` (page 1984)	`children` (page 1977)
`setObjectValue:` (page 1987)	`detach` (page 1978)
`stringValue` (page 1989)	`localName` (page 1981)
`setStringValue:resolvingEntities:` (page 1988)	`prefix` (page 1985)
`index` (page 1978)	`URI` (page 1990)

By default `NSXMLNode` implements the `NSObject` `isEqual:` (page 2101) method to perform a deep comparison: two `NSXMLNode` objects are not considered equal unless they have the same name, same child nodes, same attributes, and so on. The comparison looks at the node and its children, but does not include the node's parent. If you want a different standard of comparison, override `isEqual:`.

Special Considerations

Because of the architecture and data model of NSXML, when it parses and processes a source of XML it cannot know about your subclass unless you override the `NSXMLDocument` class method `replacementClassForClass:` (page 1903) to return your custom class in place of an NSXML class. If your custom class has no direct NSXML counterpart—for example, it is a subclass of `NSXMLNode` that represents CDATA sections—then you can walk the tree after it has been created and insert the new node where appropriate.

Adopted Protocols

NSCopying
- `copyWithZone:` (page 2042)

Tasks

Creating and Initializing Node Objects

- `initWithKind:` (page 1979)
  Returns an `NSXMLNode` instance initialized with the constant indicating node kind.
- `initWithKind:options:` (page 1979)
  Returns an `NSXMLNode` instance initialized with the constant indicating node kind and one or more initialization options.
+ `document` (page 1969)
  Returns an empty document node.
+ `documentWithRootElement:` (page 1969)
  Returns an `NSXMLDocument` object initialized with a given root element.
+ `elementWithName:` (page 1970)
  Returns an `NSXMLElement` object with a given tag identifier, or name
+ `elementWithName:children:attributes:` (page 1971)
  Returns an `NSXMLElement` object with the given tag (name), attributes, and children.
+ `elementWithName:stringValue:` (page 1971)
  Returns an `NSXMLElement` object with a single text-node child containing the specified text.
+ `elementWithName:URI:` (page 1972)
  Returns an element whose fully qualified name is specified.
+ `attributeWithName:stringValue:` (page 1968)
  Returns an `NSXMLNode` object representing an attribute node with a given name and string.
+ `attributeWithName:URI:stringValue:` (page 1968)
  Returns an `NSXMLNode` object representing an attribute node with a given qualified name and string.
+ `textWithStringValue:` (page 1975)
  Returns an `NSXMLNode` object representing a text node with specified content.
+ `commentWithStringValue:` (page 1969)
  Returns an `NSXMLNode` object representing an comment node containing given text.
+ `namespaceWithName:stringValue:` (page 1973)
  Returns an `NSXMLNode` object representing a namespace with a specified name and URI.
+ `DTDNodeWithXMLString:` (page 1970)
  Returns a `NSXMLDTDNode` object representing the DTD declaration for an element, attribute, entity, or notation based on a given string.
+ `predefinedNamespaceForPrefix:` (page 1973)
  Returns an `NSXMLNode` object representing one of the predefined namespaces with the specified prefix.
+ `processingInstructionWithName:stringValue:` (page 1974)
  Returns an `NSXMLNode` object representing a processing instruction with a specified name and value.

Managing XML Node Objects

- `index` (page 1978)
  Returns the index of the receiver identifying its position relative to its sibling nodes.
- `kind` (page 1980)
  Returns the kind of node the receiver is as a constant of type `NSXMLNodeKind` (page 1992).
- `level` (page 1980)
  Returns the nesting level of the receiver within the tree hierarchy.
- `setName:` (page 1987)
  Sets the name of the receiver.
- `name` (page 1981)
  Returns the name of the receiver.
- `setObjectValue:` (page 1987)
  Sets the content of the receiver to an object value.
- `objectValue` (page 1984)
  Returns the object value of the receiver.
- `setStringValue:` (page 1988)
  Sets the content of the receiver as a string value.
- `setStringValue:resolvingEntities:` (page 1988)
  Sets the content of the receiver as a string value and, optionally, resolves character references, predefined entities, and user-defined entities as declared in the associated DTD.
- `stringValue` (page 1989)
  Returns the content of the receiver as a string value.
- `setURI:` (page 1989)
  Sets the URI of the receiver.
- `URI` (page 1990)
  Returns the URI associated with the receiver.

Navigating the Tree of Nodes

- `rootDocument` (page 1986)
  Returns the `NSXMLDocument` object containing the root element and representing the XML document as a whole.
- `parent` (page 1985)
  Returns the parent node of the receiver.
- `childAtIndex:` (page 1976)
  Returns the child node of the receiver at the specified location.
- `childCount` (page 1977)
  Returns the number of child nodes the receiver has.
- `children` (page 1977)
  Returns an immutable array containing the child nodes of the receiver (as `NSXMLNode` objects).
- `nextNode` (page 1982)
  Returns the next `NSXMLNode` object in document order.

- `nextSibling` (page 1982)
  Returns the next `NSXMLNode` object that is a sibling node to the receiver.
- `previousNode` (page 1986)
  Returns the previous `NSXMLNode` object in document order.
- `previousSibling` (page 1986)
  Returns the previous `NSXMLNode` object that is a sibling node to the receiver.
- `detach` (page 1978)
  Detaches the receiver from its parent node.

Emitting Node Content

- `XMLString` (page 1990)
  Returns the string representation of the receiver as it would appear in an XML document.
- `XMLStringWithOptions:` (page 1991)
  Returns the string representation of the receiver as it would appear in an XML document, with one or more output options specified.
- `canonicalXMLStringPreservingComments:` (page 1975)
  Returns a string object encapsulating the receiver's XML in canonical form.
- `description` (page 1977)
  Returns a description of the receiver.

Executing Queries

- `nodesForXPath:error:` (page 1982)
  Returns the nodes resulting from executing an XPath query upon the receiver.
- `objectsForXQuery:error:` (page 1984)
  Returns the objects resulting from executing an XQuery query upon the receiver.
- `objectsForXQuery:constants:error:` (page 1983)
  Returns the objects resulting from executing an XQuery query upon the receiver.
- `XPath` (page 1991)
  Returns the XPath expression identifying the receiver's location in the document tree.

Managing Namespaces

- `localName` (page 1981)
  Returns the local name of the receiver.
+ `localNameForName:` (page 1972)
  Returns the local name from the specified qualified name.
- `prefix` (page 1985)
  Returns the prefix of the receiver's name.
+ `prefixForName:` (page 1974)
  Returns the prefix from the specified qualified name.

Class Methods

attributeWithName:stringValue:

Returns an `NSXMLNode` object representing an attribute node with a given name and string.

```
+ (id)attributeWithName:(NSString *)name stringValue:(NSString *)value
```

Parameters

name

A string that is the name of an attribute.

value

A string that is the value of an attribute.

Return Value

An `NSXMLNode` object of kind `NSXMLAttributeKind` or `nil` if the object couldn't be created.

Discussion

For example, in the attribute "id=`12345'", "id" is the attribute name and "12345" is the attribute value.

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

AlbumToSlideshow

Core Data HTML Store

Declared In

`NSXMLNode.h`

attributeWithName:URI:stringValue:

Returns an `NSXMLNode` object representing an attribute node with a given qualified name and string.

```
+ (id)attributeWithName:(NSString *)name URI:(NSString *)URI stringValue:(NSString
    *)value
```

Parameters

name

A string that is the name of an attribute.

URI

A URI (Universal Resource Identifier) that qualifies *name*.

value

A string that is the value of the attribute.

Return Value

An `NSXMLNode` object of kind `NSXMLAttributeKind` or `nil` if the object couldn't be created.

Discussion

For example, in the attribute "bst:id=`12345'", "bst" is the name qualifier (derived from the URI), "id" is the attribute name, and "12345" is the attribute value.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSXMLNode.h`

commentWithStringValue:

Returns an `NSXMLNode` object representing an comment node containing given text.

```
+ (id)commentWithStringValue:(NSString *)stringValue
```

Parameters

`stringValue`

A string specifying the text of the comment. You may specify `nil` or an empty string (see Return Value).

Return Value
An `NSXMLNode` object representing an comment node (`NSXMLCommentKind`) containing the text *`stringValue`* or `nil` if the object couldn't be created. If *`stringValue`* is `nil` or an empty string, a content-less comment node is returned (`<!-->`).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSXMLNode.h`

document

Returns an empty document node.

```
+ (id)document
```

Return Value
An empty document node—that is, an `NSXMLDocument` instance without a root element or XML-declaration information (version, encoding, standalone flag). Returns `nil` if the object couldn't be created.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSXMLNode.h`

documentWithRootElement:

Returns an `NSXMLDocument` object initialized with a given root element.

```
+ (id)documentWithRootElement:(NSXMLElement *)element
```

Parameters

element

An NSXMLElement object representing an element.

Return Value

An NSXMLDocument object initialized with the root element *element* or nil if the object couldn't be created.

Availability

Available in Mac OS X v10.4 and later.

Declared In

NSXMLNode.h

DTDNodeWithXMLString:

Returns a NSXMLDTDNode object representing the DTD declaration for an element, attribute, entity, or notation based on a given string.

```
+ (id)DTDNodeWithXMLString:(NSString *)string
```

Parameters

string

A string that is a DTD declaration. The receiver parses this string to determine the kind of DTD node to create.

Return Value

An NSXMLDTDNode object representing the DTD declaration or nil if the object couldn't be created.

Discussion

For example, if *string* is the following:

```
<!ENTITY name (#PCDATA)>
```

NSXMLNode is able to assign the created node object a kind of NSXMLEntityDeclarationKind by parsing "ENTITY".

Note that if an attribute-list declaration (<!ATTLIST...>)has multiple attributes NSXMLNode only creates an NSXMLDTDNode object for the last attribute in the declaration.

Availability

Available in Mac OS X v10.4 and later.

Declared In

NSXMLNode.h

elementWithName:

Returns an NSXMLElement object with a given tag identifier, or name

```
+ (id)elementWithName:(NSString *)name
```

Parameters

name

A string that is the name (or tag identifier) of an element.

Return Value
An `NSXMLElement` object or `nil` if the object couldn't be created.

Discussion
The equivalent XML markup is `<name></name>`.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
AlbumToSlideshow
Core Data HTML Store

Declared In
`NSXMLNode.h`

elementWithName:children:attributes:

Returns an `NSXMLElement` object with the given tag (name), attributes, and children.

```
+ (id)elementWithName:(NSString *)name children:(NSArray *)children
    attributes:(NSArray *)attributes
```

Parameters

name
A string that is the name (tag identifier) of the element.

children
An array of `NSXMLElement` objects or `NSXMLNode` objects of kinds `NSXMLElementKind`, `NSXMLProcessingInstructionKind`, `NSXMLCommentKind`, and `NSXMLTextKind`. Specify `nil` if there are no children to add to this node object.

attributes
An array of `NSXMLNode` objects of kind `NSXMLAttributeKind`. Specify `nil` if there are no attributes to add to this node object.

Return Value
An `NSXMLElement` object or `nil` if the object couldn't be created.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSXMLNode.h`

elementWithName:stringValue:

Returns an `NSXMLElement` object with a single text-node child containing the specified text.

```
+ (id)elementWithName:(NSString *)name stringValue:(NSString *)string
```

Parameters

name
A string that is the name (tag identifier) of the element.

string
A string that is the content of the receiver's text node.

Return Value
An `NSXMLElement` object with a single text-node child—an `NSXMLNode` object of kind `NSXMLTextKind`—containing the text specified in *string*. Returns `nil` if the object couldn't be created.

Discussion
The equivalent XML markup is <*name*>*string*</*name*>.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
AlbumToSlideshow
Core Data HTML Store

Declared In
`NSXMLNode.h`

elementWithName:URI:

Returns an element whose fully qualified name is specified.

```
+ (id)elementWithName:(NSString *)name URI:(NSString *)URI
```

Parameters

name
A string that is the name (or tag identifier) of an element.

URI
A URI (Universal Resource Identifier) that qualifies *name*.

Return Value
An `NSXMLElement` object or `nil` if the object cannot be created.

Discussion

The equivalent XML markup is <*URI*:*name*></*URI*:*name*>.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSXMLNode.h`

localNameForName:

Returns the local name from the specified qualified name.

```
+ (NSString *)localNameForName:(NSString *)qName
```

Parameters

qName
A string that is a qualified name.

Discussion
For example, if the qualified name is “bst:title”, this method returns “title”.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `- localName` (page 1981)
- `+ predefinedNamespaceForPrefix:` (page 1973)
- `+ prefixForName:` (page 1974)

Declared In
`NSXMLNode.h`

namespaceWithName:stringValue:

Returns an `NSXMLNode` object representing a namespace with a specified name and URI.

```
+ (id)namespaceWithName:(NSString *)name stringValue:(NSString *)value
```

Parameters

name
: A string that is the name of the namespace. Specify `nil` or an empty string for *name* if this object represents the default namespace.

value
: A string that identifies the URI associated with the namespace.

Return Value
An `NSXMLNode` object of kind `NSXMLNamespaceKind` or `nil` if the object couldn't be created.

Discussion
The equivalent namespace declaration in XML markup is `xmlns:`*name*`="`*value*`"`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSXMLNode.h`

predefinedNamespaceForPrefix:

Returns an `NSXMLNode` object representing one of the predefined namespaces with the specified prefix.

```
+ (NSXMLNode *)predefinedNamespaceForPrefix:(NSString *)name
```

Parameters

name
: A string specifying a prefix for a predefined namespace, for example “xml”, “xs”, or “xsi”.

Return Value
An `NSXMLNode` object of kind `NSXMLNamespaceKind` or `nil` if the object couldn't be created. If something other than a predefined-namespace prefix is specified, the method returns `nil`.

Availability
Available in Mac OS X v10.4 and later.

See Also
+ localNameForName: (page 1972)
+ prefixForName: (page 1974)

Declared In
NSXMLNode.h

prefixForName:

Returns the prefix from the specified qualified name.

```
+ (NSString *)prefixForName:(NSString *)qName
```

Parameters

qName
A string that is a qualified name.

Discussion
For example, if the qualified name is "bst:title", this method returns "bst".

Availability
Available in Mac OS X v10.4 and later.

See Also
+ localNameForName: (page 1972)
- prefix (page 1985)
+ predefinedNamespaceForPrefix: (page 1973)

Declared In
NSXMLNode.h

processingInstructionWithName:stringValue:

Returns an NSXMLNode object representing a processing instruction with a specified name and value.

```
+ (id)processingInstructionWithName:(NSString *)name stringValue:(NSString *)value
```

Parameters

name
A string that is the name of the processing instruction.

value
A string that is the value of the processing instruction.

Return Value
An NSXMLNode object of kind NSXMLProcessingInstructionKind or nil if the object couldn't be created.

Discussion
The equivalent XML markup is <?*name value*?>.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSXMLNode.h`

textWithStringValue:

Returns an `NSXMLNode` object representing a text node with specified content.

```
+ (id)textWithStringValue:(NSString *)value
```

Parameters

value

A string that is the textual content of the node.

Return Value
An `NSXMLNode` object of kind `NSXMLTextKind` initialized with the textual *value* or `nil` if the object couldn't be created.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSXMLNode.h`

Instance Methods

canonicalXMLStringPreservingComments:

Returns a string object encapsulating the receiver's XML in canonical form.

```
- (NSString *)canonicalXMLStringPreservingComments:(BOOL)comments
```

Parameters

comments

`YES` to preserve comments, `NO` otherwise.

Discussion
Be sure to set the input option `NSXMLNodePreserveWhitespace` for true canonical form. The canonical form of an XML document is defined by the World Wide Web Consortium at `http://www.w3.org/TR/xml-c14n`. Generally, if two documents with varying physical representations have the same canonical form, then they are considered logically equivalent within the given application context. The following list summarizes most key aspects of canonical form as defined by the W3C recommendation:

- Encodes the document in UTF-8.
- Normalizes line breaks to "#xA" on input before parsing.
- Normalizes attribute values in the manner of a validating processor.
- Replaces character and parsed entity references with their character content.

- Replaces CDATA sections with their character content.
- Removes the XML declaration and the document type declaration (DTD).
- Converts empty elements to start-end tag pairs.
- Normalizes whitespace outside of the document element and within start and end tags.
- Retains all whitespace characters in content (excluding characters removed during line-feed normalization).
- Sets attribute value delimiters to quotation marks (double quotes).
- Replaces special characters in attribute values and character content with character references.
- Removes superfluous namespace declarations from each element.
- Adds default attributes to each element.
- Imposes lexicographic order on the namespace declarations and attributes of each element.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `XMLString` (page 1990)
- `XMLStringWithOptions:` (page 1991)

Declared In
`NSXMLNode.h`

childAtIndex:

Returns the child node of the receiver at the specified location.

```
- (NSXMLNode *)childAtIndex:(NSUInteger)index
```

Parameters

index

An integer specifying a node position in the receiver's array of children. If *index* is out of bounds, an exception is raised.

Return Value
An NSXMLNode object or `nil` f the receiver has no children.

Discussion
The receiver should be an `NSXMLNode` object representing a document, element, or document type declaration. The returned node object can represent an element, comment, text, or processing instruction.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `childCount` (page 1977)

Related Sample Code
Core Data HTML Store

Declared In
NSXMLNode.h

childCount

Returns the number of child nodes the receiver has.

```
- (NSUInteger)childCount
```

Discussion
This receiver should be an NSXMLNode object representing a document, element, or document type declaration. For performance reasons, use this method instead of getting the count from the array returned by children (page 1977) (for example, [[thisNode children] count]).

Availability
Available in Mac OS X v10.4 and later.

See Also
- childAtIndex: (page 1976)
- children (page 1977)
- parent (page 1985)

Related Sample Code
Core Data HTML Store

Declared In
NSXMLNode.h

children

Returns an immutable array containing the child nodes of the receiver (as NSXMLNode objects).

```
- (NSArray *)children
```

Availability
Available in Mac OS X v10.4 and later.

See Also
- childAtIndex: (page 1976)
- childCount (page 1977)
- parent (page 1985)

Declared In
NSXMLNode.h

description

Returns a description of the receiver.

```
- (NSString *)description
```

Discussion
Use this method for debugging rather than for generating XML output. It could yield more information than `XMLString` (page 1990) and `XMLStringWithOptions:` (page 1991).

Availability
Available in Mac OS X v10.4 and later.

See Also
- `XMLString` (page 1990)
- `XMLStringWithOptions:` (page 1991)

Declared In
`NSXMLNode.h`

detach

Detaches the receiver from its parent node.

```
- (void)detach
```

Discussion
This method is applicable to `NSXMLNode` objects representing elements, text, comments, processing instructions, attributes, and namespaces. Once the node object is detached, you can add it as a child node of another parent.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSXMLNode.h`

index

Returns the index of the receiver identifying its position relative to its sibling nodes.

```
- (NSUInteger)index
```

Return Value
An integer that is the index of the receiver relative to its sibling nodes.

Discussion
The first child node of a parent has an index of zero.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `childAtIndex:` (page 1976)

Declared In
`NSXMLNode.h`

initWithKind:

Returns an NSXMLNode instance initialized with the constant indicating node kind.

```
- (id)initWithKind:(NSXMLNodeKind)kind
```

Parameters

kind

An enum constant of type NSXMLNodeKind (page 1992) that indicates the type of node. See "Constants" (page 1992) for a list of valid NSXMLNodeKind constants.

Return Value

An NSXMLNode object initialized with kind or nil if the object couldn't be created. If *kind* is not a valid NSXMLNodeKind constant, the method returns an NSXMLNode object of kind NSXMLInvalidKind.

Discussion

This method invokes initWithKind:options: (page 1979) with the *options* parameter set to NSXMLNodeOptionsNone.

Do not use this initializer for creating instances of NSXMLDTDNode for attribute-list declarations. Instead, use the DTDNodeWithXMLString: (page 1970) class method of this class or the initWithXMLString: (page 1937) method of the NSXMLDTDNode class.

Availability

Available in Mac OS X v10.4 and later.

Declared In

NSXMLNode.h

initWithKind:options:

Returns an NSXMLNode instance initialized with the constant indicating node kind and one or more initialization options.

```
- (id)initWithKind:(NSXMLNodeKind)kind options:(NSUInteger)options
```

Parameters

kind

An enum constant of type NSXMLNodeKind (page 1992) that indicates the type of node. See "Constants" (page 1992) for a list of valid NSXMLNodeKind constants.

options

One or more constants that specify initialization options; if there are multiple constants, bit-OR them together. These options request operations on the represented XML related to fidelity (for example, preserving entities), quoting style, handling of empty elements, and other things. See "Constants" (page 1992) for a list of valid node-initialization constants.

Return Value

An NSXMLNode object initialized with the given kind and options, or nil if the object couldn't be created. If *kind* is not a valid NSXMLNodeKind constant, the method returns an NSXMLNode object of kind NSXMLInvalidKind.

Discussion

Do not use this initializer for creating instances of NSXMLDTDNode for attribute-list declarations. Instead, use the DTDNodeWithXMLString: (page 1970) class method of this class or the initWithXMLString: (page 1937) method of the NSXMLDTDNode class.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `initWithKind:` (page 1979)

Related Sample Code
Core Data HTML Store

Declared In
`NSXMLNode.h`

kind

Returns the kind of node the receiver is as a constant of type `NSXMLNodeKind` (page 1992).

```
- (NSXMLNodeKind)kind
```

Discussion
`NSXMLNode` objects can represent documents, elements, attributes, namespaces, text, processing instructions, comments, document type declarations, and specific declarations within DTDs. See "Constants" (page 1992) for a list of valid `NSXMLNodeKind` constants

Availability
Available in Mac OS X v10.4 and later.

See Also
- `initWithKind:` (page 1979)

Declared In
`NSXMLNode.h`

level

Returns the nesting level of the receiver within the tree hierarchy.

```
- (NSUInteger)level
```

Return Value
An integer indicating a nesting level.

Discussion
The root element of a document has a nesting level of one.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSXMLNode.h`

localName

Returns the local name of the receiver.

```
- (NSString *)localName
```

Discussion
The local name is the part of a node name that follows a namespace-qualifying colon or the full name if there is no colon. For example, “chapter” is the local name in the qualified name “acme:chapter”.

Availability
Available in Mac OS X v10.4 and later.

See Also
`+ localNameForName:` (page 1972)

Declared In
`NSXMLNode.h`

name

Returns the name of the receiver.

```
- (NSString *)name
```

Return Value
Returns a string specifying the name of the receiver. May return `nil` if the receiver is not a valid kind of node (see discussion).

Discussion
This method is applicable only to `NSXMLNode` objects representing elements, attributes, namespaces, processing instructions, and DTD-declaration nodes. If the receiver is not an object of one of these kinds, this method returns `nil`. For example, in the following construction:

```
<title>Chapter One</title>
```

The returned name for the element is “title”. If the name is associated with a namespace, the qualified name is returned. For example, if you create an element with local name “foo” and URI “http://bar.com” and the namespace “xmlns:baz=‘http://bar.com’” is applied to this node, when you invoke this method on the node you get “baz:foo”.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setName:` (page 1987)

Related Sample Code
Core Data HTML Store

Declared In
`NSXMLNode.h`

nextNode

Returns the next `NSXMLNode` object in document order.

```
- (NSXMLNode *)nextNode
```

Discussion
You use this method to "walk" forward through the tree structure representing an XML document or document section. (Use `previousNode` (page 1986) to traverse the tree in the opposite direction.) Document order is the natural order that XML constructs appear in markup text. If you send this message to the last node in the tree, `nil` is returned. `NSXMLNode` bypasses namespace and attribute nodes when it traverses a tree in document order.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `nextSibling` (page 1982)
- `previousSibling` (page 1986)

Declared In
`NSXMLNode.h`

nextSibling

Returns the next `NSXMLNode` object that is a sibling node to the receiver.

```
- (NSXMLNode *)nextSibling
```

Discussion
This object will have an `index` (page 1978) value that is one more than the receiver's. If there are no more subsequent siblings (that is, other child nodes of the receiver's parent) the method returns `nil`.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `nextNode` (page 1982)
- `previousNode` (page 1986)
- `previousSibling` (page 1986)

Declared In
`NSXMLNode.h`

nodesForXPath:error:

Returns the nodes resulting from executing an XPath query upon the receiver.

```
- (NSArray *)nodesForXPath:(NSString *)xpath error:(NSError **)error
```

Parameters
xpath
A string that expresses an XPath query.

error

If query errors occur, indirectly returns an `NSError` object describing the errors.

Return Value
An array of `NSXMLNode` objects that match the query, or an empty array if there are no matches.

Discussion
The receiver acts as the context item for the query (“.”). If you have explicitly added adjacent text nodes as children of an element, you should invoke the `NSXMLElement` method `normalizeAdjacentTextNodesPreservingCDATA:` (page 1957) (with an argument of `NO`) on the element before applying any XPath queries to it; this method coalesces these text nodes. The same precaution applies if you have processed a document preserving CDATA sections and these sections are adjacent to text nodes.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- XPath` (page 1991)

Related Sample Code
CocoaSOAP
Core Data HTML Store

Declared In
`NSXMLNode.h`

objectsForXQuery:constants:error:

Returns the objects resulting from executing an XQuery query upon the receiver.

```
- (NSArray *)objectsForXQuery:(NSString *)xquery constants:(NSDictionary *)constants
    error:(NSError **)error
```

Parameters

xquery

A string that expresses an XQuery query.

constants

A dictionary containing externally declared constants where the name of each constant variable is a key.

error

If query errors occur, indirectly returns an `NSError` object describing the errors.

Discussion
The receiver acts as the context item for the query (“.”). If the receiver has been changed after parsing to have multiple adjacent text nodes, you should invoke the `NSXMLElement` method `normalizeAdjacentTextNodesPreservingCDATA:` (page 1957) (with an argument of `NO`) to coalesce the text nodes before querying.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- XPath` (page 1991)

Declared In
NSXMLNode.h

objectsForXQuery:error:

Returns the objects resulting from executing an XQuery query upon the receiver.

```
- (NSArray *)objectsForXQuery:(NSString *)xquery error:(NSError **)error
```

Parameters

xquery
A string that expresses an XQuery query.

error
If query errors occur, indirectly returns an NSError object describing the errors.

Discussion
The receiver acts as the context item for the query (“.”). If the receiver has been changed after parsing to have multiple adjacent text nodes, you should invoke the NSXMLElement method normalizeAdjacentTextNodesPreservingCDATA: (page 1957) (with an argument of NO) to coalesce the text nodes before querying .This convenience method invokes objectsForXQuery:constants:error: (page 1983) with nil for the *constants* dictionary.

Availability
Available in Mac OS X v10.4 and later.

See Also
- XPath (page 1991)

Related Sample Code
TimelineToTC

Declared In
NSXMLNode.h

objectValue

Returns the object value of the receiver.

```
- (id)objectValue
```

Return Value
The object value of the receiver, which may be the same as the value returned by stringValue (page 1989). For nodes without content (for example, document nodes), this method returns the string value, or an empty string if there is no string value.

Availability
Available in Mac OS X v10.4 and later.

See Also
- setObjectValue: (page 1987)
- setStringValue: (page 1988)

Related Sample Code
CocoaSOAP
Core Data HTML Store

Declared In
NSXMLNode.h

parent

Returns the parent node of the receiver.

```
- (NSXMLNode *)parent
```

Discussion
Document nodes and standalone nodes (that is, the root of a detached branch of a tree) have no parent, and sending this message to them returns `nil`. A one-to-one relationship does not always exists between a parent and its children; although a namespace or attribute node cannot be a child, it still has a parent element.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `childCount` (page 1977)
- `children` (page 1977)

Related Sample Code
Core Data HTML Store

Declared In
NSXMLNode.h

prefix

Returns the prefix of the receiver's name.

```
- (NSString *)prefix
```

Discussion
The prefix is the part of a namespace-qualified name that precedes the colon. For example, "acme" is the local name in the qualified name "acme:chapter". This method returns an empty string if the receiver's name is not qualified by a namespace.

Availability
Available in Mac OS X v10.4 and later.

See Also
+ `prefixForName:` (page 1974)

Declared In
NSXMLNode.h

previousNode

Returns the previous `NSXMLNode` object in document order.

```
- (NSXMLNode *)previousNode
```

Discussion
You use this method to "walk" backward through the tree structure representing an XML document or document section. (Use `nextNode` (page 1982) to traverse the tree in the opposite direction.) Document order is the natural order that XML constructs appear in markup text. If you send this message to the first node in the tree (that is, the root element), `nil` is returned. `NSXMLNode` bypasses namespace and attribute nodes when it traverses a tree in document order.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `nextSibling` (page 1982)
- `previousSibling` (page 1986)

Declared In
`NSXMLNode.h`

previousSibling

Returns the previous `NSXMLNode` object that is a sibling node to the receiver.

```
- (NSXMLNode *)previousSibling
```

Discussion
This object will have an `index` (page 1978) value that is one less than the receiver's. If there are no more previous siblings (that is, other child nodes of the receiver's parent) the method returns `nil`

Availability
Available in Mac OS X v10.4 and later.

See Also
- `nextNode` (page 1982)
- `nextSibling` (page 1982)
- `previousNode` (page 1986)

Declared In
`NSXMLNode.h`

rootDocument

Returns the `NSXMLDocument` object containing the root element and representing the XML document as a whole.

```
- (NSXMLDocument *)rootDocument
```

Discussion
If the receiver is a standalone node (that is, a node at the head of a detached branch of the tree), this method returns `nil`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSXMLNode.h`

setName:

Sets the name of the receiver.

```
- (void)setName:(NSString *)name
```

Parameters

name

A string that is the name to assign to the receiver.

Discussion
This method is effective for the following node kinds: element, attribute, namespace, processing-instruction, document type declaration, element declaration, attribute declaration, entity declaration, and notation declaration. If an `NSXMLNode` object that requires a name doesn't have one, it cannot be written out as an XML string.

Availability
Available in Mac OS X v10.4 and later.

See Also
– `name` (page 1981)

Declared In
`NSXMLNode.h`

setObjectValue:

Sets the content of the receiver to an object value.

```
- (void)setObjectValue:(id)value
```

Parameters

value

An object to assign as the value to the receiver.

Discussion
This method can only be invoked on `NSXMLNode` objects that may have content, specifically elements, attributes, namespaces, processing instructions, text, and DTD-declaration nodes. The given object is usually a Foundation equivalent to one of the atomic types in the XQuery data model: `NSNumber` (integer, decimal, float, double, Boolean), `NSString` (string), `NSCalendarDate` (date), `NSData` (base64Binary and hexBinary), `NSURL` (URI), and `NSArray` (NMTOKENS, IDREFS, ENTITIES). However, you can also set the object value to be a custom value and register a value transformer (that is, an instance of `NSValueTransformer`) to convert the object value to an XML string representation when the node is asked for its string value. Setting a node's

object value removes all existing children, including processing instructions and comments. Setting an element node's object value creates a text node as the sole child. When an `NSXMLNode` object emits its object-value contents as a string, and it can determine the type of the value, it ensures that it the string is in a canonical form as defined by the W3C XML Schema Data Types specification.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `objectValue` (page 1984)
- `setStringValue:resolvingEntities:` (page 1988)

Related Sample Code
Core Data HTML Store

Declared In
`NSXMLNode.h`

setStringValue:

Sets the content of the receiver as a string value.

```
- (void)setStringValue:(NSString *)string
```

Parameters
string
A string to assign as the value of the receiver.

Discussion
This method invokes `setStringValue:resolvingEntities:` (page 1988), passing in an argument of `NO` for the second parameter. This method can only be invoked on `NSXMLNode` objects that may have content, specifically elements, attributes, namespaces, processing instructions, text, and DTD-declaration nodes. Setting the string value of a node object removes all existing children, including processing instructions and comments. Setting the string value of an element-node object creates a text node as the sole child.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setObjectValue:` (page 1987)
- `stringValue` (page 1989)

Related Sample Code
Core Data HTML Store

Declared In
`NSXMLNode.h`

setStringValue:resolvingEntities:

Sets the content of the receiver as a string value and, optionally, resolves character references, predefined entities, and user-defined entities as declared in the associated DTD.

```
- (void)setStringValue:(NSString *)string resolvingEntities:(BOOL)resolve
```

Parameters

string

A string to assign as the value of the receiver.

resolve

`YES` to resolve character references, predefined entities, and user-defined entities as declared in the associated DTD; `NO` otherwise. Namespace and processing-instruction nodes have their entities resolved even if *resolve* is `NO`.

Discussion

User-defined entities not declared in the DTD remain in their unresolved form. This method can only be invoked on `NSXMLNode` objects that may have content, specifically elements, attributes, namespaces, processing instructions, text, and DTD-declaration nodes. Setting the string value of a node object removes all existing children, including processing instructions and comments. Setting the string value of an element -node object creates a text node as the sole child.

Availability

Available in Mac OS X v10.4 and later.

See Also

- `setObjectValue:` (page 1987)
- `setStringValue:` (page 1988)
- `stringValue` (page 1989)

Declared In

`NSXMLNode.h`

setURI:

Sets the URI of the receiver.

```
- (void)setURI:(NSString *)URI
```

Parameters

URI

The URI to associate with the receiver. A URI (Universal Resource Identifier) is a scheme such as “http” or “ftp” followed by a colon character, and then a scheme-specific part.

Discussion

The receiver must be an `NSXMLElement` or `NSXMLDocument` document, or an attribute (that is, an`NSXMLNode` object of type `NSXMLAttributeKind`). For documents it is the URI of document origin.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`NSXMLNode.h`

stringValue

Returns the content of the receiver as a string value.

```
- (NSString *)stringValue
```

Discussion
If the receiver is a node object of element kind, the content is that of any text-node children. This method recursively visits elements nodes and concatenates their text nodes in document order with no intervening spaces. If the receiver's content is set as an object value, this method returns the string value representing the object. If the object value is one of the standard, built-in ones (`NSNumber`, `NSCalendarDate`, and so on), the string value is in canonical format as defined by the W3C XML Schema Data Types specification. If the object value is not represented by one of these classes (or if the default value transformer for a class has been overridden), the string value is generated by the `NSValueTransformer` registered for that object type.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `objectValue` (page 1984)
- `setStringValue:` (page 1988)
- `setStringValue:resolvingEntities:` (page 1988)

Related Sample Code
Core Data HTML Store
TimelineToTC

Declared In
`NSXMLNode.h`

URI

Returns the URI associated with the receiver.

```
- (NSString *)URI
```

Discussion
A node's URI is derived from its namespace or a document's URI; for documents, the URI comes either from the parsed XML or is explicitly set. You cannot change the URI for a particular node other for than a namespace or document node.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `setURI:` (page 1916) (NSXMLDocument)

Declared In
`NSXMLNode.h`

XMLString

Returns the string representation of the receiver as it would appear in an XML document.

```
- (NSString *)XMLString
```

Discussion
The returned string includes the string representations of all children. This method invokes `XMLStringWithOptions:` (page 1991) with an *options* argument of `NSXMLNodeOptionsNone`.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `canonicalXMLStringPreservingComments:` (page 1975)
- `description` (page 1977)

Declared In
`NSXMLNode.h`

XMLStringWithOptions:

Returns the string representation of the receiver as it would appear in an XML document, with one or more output options specified.

```
- (NSString *)XMLStringWithOptions:(NSUInteger)options
```

Parameters
options
One or more `enum` constants identifying an output option; bit-OR multiple constants together. See "Constants" (page 1992) for a list of valid constants for specifying output options.

Discussion
The returned string includes the string representations of all children.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSXMLNode.h`

XPath

Returns the XPath expression identifying the receiver's location in the document tree.

```
- (NSString *)XPath
```

Discussion
For example, this method might return a string such as "foo/bar[2]/baz". The result of this method can be used directly in the `nodesForXPath:error:` (page 1982) and `objectsForXQuery:constants:error:` (page 1983) methods.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSXMLNode.h`

Constants

NSXMLNodeKind

A type defined for the node-kind constants described in "Node Kind Constants" (page 1992).

```
typedef NSUInteger NSXMLNodeKind;
```

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSXMLNode.h

Node Kind Constants

NSXMLNode declares the following constants of type NSXMLNodeKind for specifying a node's kind in the initializer methods initWithKind: (page 1979) and initWithKind:options: (page 1979):

```
enum {
        NSXMLInvalidKind = 0,
        NSXMLDocumentKind,
        NSXMLElementKind,
        NSXMLAttributeKind,
        NSXMLNamespaceKind,
        NSXMLProcessingInstructionKind,
        NSXMLCommentKind,
        NSXMLTextKind,
        NSXMLDTDKind,
        NSXMLEntityDeclarationKind,
        NSXMLAttributeDeclarationKind,
        NSXMLElementDeclarationKind,
        NSXMLNotationDeclarationKind
};
```

Constants

NSXMLInvalidKind
: Indicates a node object created without a valid kind being specified (as returned by the kind (page 1980) method).

Available in Mac OS X v10.4 and later.

Declared in NSXMLNode.h.

NSXMLDocumentKind
: Specifies a document node.

Available in Mac OS X v10.4 and later.

Declared in NSXMLNode.h.

NSXMLElementKind
: Specifies an element node.

Available in Mac OS X v10.4 and later.

Declared in NSXMLNode.h.

`NSXMLAttributeKind`
Specifies an attribute node

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNode.h`.

`NSXMLNamespaceKind`
Specifies a namespace node.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNode.h`.

`NSXMLProcessingInstructionKind`
Specifies a processing-instruction node.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNode.h`.

`NSXMLCommentKind`
Specifies a comment node.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNode.h`.

`NSXMLTextKind`
Specifies a text node.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNode.h`.

`NSXMLDTDKind`
Specifies a document-type declaration (DTD) node.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNode.h`.

`NSXMLEntityDeclarationKind`
Specifies an entity-declaration node.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNode.h`.

`NSXMLAttributeDeclarationKind`
Specifies an attribute-list declaration node.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNode.h`.

`NSXMLElementDeclarationKind`
Specifies an element declaration node.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNode.h`.

`NSXMLNotationDeclarationKind`
Specifies a notation declaration node.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNode.h`.

Declared In
`NSXMLNode.h`

Input and Output Options

These constants are input and output options for all `NSXMLNode` objects (unless otherwise indicated), including `NSXMLDocument` objects. You can specify these options (OR'ing multiple options) in the `NSXMLNode` methods `initWithKind:options:` (page 1979) and `XMLStringWithOptions:` (page 1991).

```
enum {
    NSXMLNodeOptionsNone = 0,
    NSXMLNodeIsCDATA = 1 << 0,
    NSXMLNodeExpandEmptyElement = 1 << 1, // <a></a>
    NSXMLNodeCompactEmptyElement =  1 << 2, // <a/>
    NSXMLNodeUseSingleQuotes = 1 << 3,
    NSXMLNodeUseDoubleQuotes = 1 << 4,
    NSXMLDocumentTidyHTML = 1 << 9,
    NSXMLDocumentTidyXML = 1 << 10,
    NSXMLDocumentValidate = 1 << 13,
    NSXMLDocumentXInclude = 1 << 16,
    NSXMLNodePrettyPrint = 1 << 17,
    NSXMLDocumentIncludeContentTypeDeclaration = 1 << 18,
    NSXMLNodePreserveNamespaceOrder = 1 << 20,
    NSXMLNodePreserveAttributeOrder = 1 << 21,
    NSXMLNodePreserveEntities = 1 << 22,
    NSXMLNodePreservePrefixes = 1 << 23,
    NSXMLNodePreserveCDATA = 1 << 24,
    NSXMLNodePreserveWhitespace = 1 << 25,
    NSXMLNodePreserveDTD = 1 << 26,
    NSXMLNodePreserveCharacterReferences = 1 << 27,
    NSXMLNodePreserveEmptyElements =
            (NSXMLNodeExpandEmptyElement | NSXMLNodeCompactEmptyElement),
    NSXMLNodePreserveQuotes =
            (NSXMLNodeUseSingleQuotes | NSXMLNodeUseDoubleQuotes),
    NSXMLNodePreserveAll = (
            NSXMLNodePreserveNamespaceOrder |
            NSXMLNodePreserveAttributeOrder |
            NSXMLNodePreserveEntities |
            NSXMLNodePreservePrefixes |
            NSXMLNodePreserveCDATA |
            NSXMLNodePreserveEmptyElements |
            NSXMLNodePreserveQuotes |
            NSXMLNodePreserveWhitespace |
            NSXMLNodePreserveDTD |
            NSXMLNodePreserveCharacterReferences |
            0xFFF00000) // high 12 bits
};
```

Constants

`NSXMLNodeOptionsNone`

No options are requested for this input or output action.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLNodeIsCDATA`

Specifies that a text node contains and is written out as a CDATA section.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLNodeExpandEmptyElement`
Requests that an element should be expanded when empty; for example, `<flag></flag>`. This is the default.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLNodeCompactEmptyElement`
Requests that an element should be contracted when empty; for example, `<flag/>`.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLNodeUseSingleQuotes`
Requests that NSXML use single quotes for the value of an attribute or namespace node.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLNodeUseDoubleQuotes`
Requests that NSXML use double quotes for the value of an attribute or namespace node. This is the default.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLNodePrettyPrint`
Print this node with extra space for readability. (Output)

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLNodePreserveNamespaceOrder`
Requests NSXML to preserve the order of namespace URI definitions as in the source XML.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLNodePreserveAttributeOrder`
Requests that NSXMLNode preserve the order of attributes as in the source XML.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLNodePreserveEntities`
Specifies that entities (`&`*xyz*`;`) should not be resolved for XML output of this node.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLNodePreserveCharacterReferences`
Specifies that character references (`&#`*nnn*`;`) should not be resolved for XML output of this node.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLNodePreservePrefixes`
Requests NSXMLNode not to choose prefixes based on the closest namespace URI definition.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLNodePreserveCDATA`
Requests that NSXMLNode preserve CDATA blocks where defined in the input XML.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLNodePreserveWhitespace`
Requests NSXMLNode to preserve whitespace characters (such as tabs and carriage returns) in the XML source that are not part of node content.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLNodePreserveEmptyElements`
Specifies that empty elements in the input XML be preserved in their contracted or expanded form.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLNodePreserveQuotes`
Specifies that the quoting style used in the input XML (single or double quotes) be preserved.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLNodePreserveDTD`
Specifies that declarations in a DTD should be preserved until it the DTD is modified. For example, parameter entities are by default expanded; with this option, they are written out as they originally occur in the DTD.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

`NSXMLNodePreserveAll`
Turns on all preservation options: attribute and namespace order, entities, prefixes, CDATA, whitespace, quotes, and empty elements. You should try to turn on preservation options selectively because turning on all preservation options significantly affects performance.

Available in Mac OS X v10.4 and later.

Declared in `NSXMLNodeOptions.h`.

Discussion
The options with "Preserve" in their names are applicable only when external sources of XML are parsed; they have no effect on node objects that are programmatically created. Other options are used in initialization and output methods of `NSXMLDocument`; see the `NSXMLDocument` reference documentation for details.

Declared In
`NSXMLNodeOptions.hj`

NSXMLParser Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.3 and later.
Declared in	Foundation/NSXMLParser.h
Companion guide	Event-Driven XML Programming Guide for Cocoa
Related sample code	ImageMapExample

Overview

Instances of this class parse XML documents (including DTD declarations) in an event-driven manner. An `NSXMLParser` notifies its delegate about the items (elements, attributes, CDATA blocks, comments, and so on) that it encounters as it processes an XML document. It does not itself do anything with those parsed items except report them. It also reports parsing errors. For convenience, an `NSXMLParser` object in the following descriptions is sometimes referred to as a parser object.

Note: Namespace support was implemented in `NSXMLParser` for Mac OS X v10.4. Namespace-related methods of `NSXMLParser` prior to this version have no effect.

Tasks

Initializing a Parser Object

`- initWithContentsOfURL:` (page 2001)
Initializes the receiver with the XML content referenced by the given URL.

`- initWithData:` (page 2002)
Initializes the receiver with the XML contents encapsulated in a given data object.

Managing Delegates

- `setDelegate:` (page 2003)
  Sets the receiver's delegate.
- `delegate` (page 2001)
  Returns the receiver's delegate.

Managing Parser Behavior

- `setShouldProcessNamespaces:` (page 2004)
  Specifies whether the receiver reports the namespace and the qualified name of an element in related delegation methods .
- `shouldProcessNamespaces` (page 2005)
  Indicates whether the receiver reports the namespace and the qualified name of an element in related delegation methods.
- `setShouldReportNamespacePrefixes:` (page 2004)
  Specifies whether the receiver reports the scope of namespace declarations using related delegation methods.
- `shouldReportNamespacePrefixes` (page 2006)
  Indicates whether the receiver reports the prefixes indicating the scope of namespace declarations using related delegation methods.
- `setShouldResolveExternalEntities:` (page 2005)
  Specifies whether the receiver reports declarations of external entities using the delegate method `parser:foundExternalEntityDeclarationWithName:publicID:systemID:` (page 2011).
- `shouldResolveExternalEntities` (page 2006)
  Indicates whether the receiver reports declarations of external entities using the delegate method `parser:foundExternalEntityDeclarationWithName:publicID:systemID:` (page 2011).

Parsing

- `parse` (page 2002)
  Starts the event-driven parsing operation.
- `abortParsing` (page 2000)
  Stops the parser object.
- `parserError` (page 2003)
  Returns an `NSError` object from which you can obtain information about a parsing error.

Handling XML

- `parserDidStartDocument:` (page 2017) *delegate method*
  Sent by the parser object to the delegate when it begins parsing a document.
- `parserDidEndDocument:` (page 2016) *delegate method*
  Sent by the parser object to the delegate when it has successfully completed parsing.

- parser:didStartElement:namespaceURI:qualifiedName:attributes: (page 2008) *delegate method*

  Sent by a parser object to its delegate when it encounters a start tag for a given element.

- parser:didEndElement:namespaceURI:qualifiedName: (page 2007) *delegate method*

  Sent by a parser object to its delegate when it encounters an end tag for a specific element.

- parser:didStartMappingPrefix:toURI: (page 2009) *delegate method*

  Sent by a parser object to its delegate the first time it encounters a given namespace prefix, which is mapped to a URI.

- parser:didEndMappingPrefix: (page 2008) *delegate method*

  Sent by a parser object to its delegate when a given namespace prefix goes out of scope.

- parser:resolveExternalEntityName:systemID: (page 2015) *delegate method*

  Sent by a parser object to its delegate when it encounters a given external entity with a specific system ID.

- parser:parseErrorOccurred: (page 2015) *delegate method*

  Sent by a parser object to its delegate when it encounters a fatal error.

- parser:validationErrorOccurred: (page 2016) *delegate method*

  Sent by a parser object to its delegate when it encounters a fatal validation error. NSXMLParser currently does not invoke this method and does not perform validation.

- parser:foundCharacters: (page 2010) *delegate method*

  Sent by a parser object to provide its delegate with a string representing all or part of the characters of the current element.

- parser:foundIgnorableWhitespace: (page 2012) *delegate method*

  Reported by a parser object to provide its delegate with a string representing all or part of the ignorable whitespace characters of the current element.

- parser:foundProcessingInstructionWithTarget:data: (page 2014) *delegate method*

  Sent by a parser object to its delegate when it encounters a processing instruction.

- parser:foundComment: (page 2011) *delegate method*

  Sent by a parser object to its delegate when it encounters a comment in the XML.

- parser:foundCDATA: (page 2010) *delegate method*

  Sent by a parser object to its delegate when it encounters a CDATA block.

Handling the DTD

- parser:foundAttributeDeclarationWithName:forElement:type:defaultValue: (page 2009) *delegate method*

  Sent by a parser object to its delegate when it encounters a declaration of an attribute that is associated with a specific element.

- parser:foundElementDeclarationWithName:model: (page 2011) *delegate method*

  Sent by a parser object to its delegate when it encounters a declaration of an element with a given model.

- parser:foundExternalEntityDeclarationWithName:publicID:systemID: (page 2011) *delegate method*

  Sent by a parser object to its delegate when it encounters an external entity declaration.

- parser:foundInternalEntityDeclarationWithName:value: (page 2013) *delegate method*

  Sent by a parser object to the delegate when it encounters an internal entity declaration.

- parser:foundUnparsedEntityDeclarationWithName:publicID:systemID:notationName: (page 2014) *delegate method*
  Sent by a parser object to its delegate when it encounters an unparsed entity declaration.
- parser:foundNotationDeclarationWithName:publicID:systemID: (page 2013) *delegate method*
  Sent by a parser object to its delegate when it encounters a notation declaration.

Obtaining Parser State

- columnNumber (page 2000)
  Returns the column number of the XML document being processed by the receiver.
- lineNumber (page 2002)
  Returns the line number of the XML document being processed by the receiver.
- publicID (page 2003)
  Returns the public identifier of the external entity referenced in the XML document.
- systemID (page 2007)
  Returns the system identifier of the external entity referenced in the XML document.

Instance Methods

abortParsing

Stops the parser object.

```
- (void)abortParsing
```

Discussion
If you invoke this method, the delegate, if it implements parser:parseErrorOccurred: (page 2015), is informed of the cancelled parsing operation.

Availability
Available in Mac OS X v10.3 and later.

See Also
- parse (page 2002)
- parserError (page 2003)

Related Sample Code
ImageMapExample

Declared In
NSXMLParser.h

columnNumber

Returns the column number of the XML document being processed by the receiver.

```
- (NSInteger)columnNumber
```

Discussion
The column refers to the nesting level of the XML elements in the document. You may invoke this method once a parsing operation has begun or after an error occurs.

Availability
Available in Mac OS X v10.3 and later.

See Also
- `lineNumber` (page 2002)

Declared In
`NSXMLParser.h`

delegate

Returns the receiver's delegate.

```
- (id)delegate
```

Availability
Available in Mac OS X v10.3 and later.

See Also
- `setDelegate:` (page 2003)

Declared In
`NSXMLParser.h`

initWithContentsOfURL:

Initializes the receiver with the XML content referenced by the given URL.

```
- (id)initWithContentsOfURL:(NSURL *)url
```

Parameters
url

An `NSURL` object specifying a URL. The URL must be fully qualified and refer to a scheme that is supported by the `NSURL` class.

Return Value
An initialized `NSXMLParser` object or `nil` if an error occurs.

Availability
Available in Mac OS X v10.3 and later.

See Also
- `initWithData:` (page 2002)

Declared In
`NSXMLParser.h`

initWithData:

Initializes the receiver with the XML contents encapsulated in a given data object.

```
- (id)initWithData:(NSData *)data
```

Parameters

data

An `NSData` object containing XML markup.

Return Value

An initialized `NSXMLParser` object or `nil` if an error occurs.

Discussion

This method is the designated initializer.

Availability

Available in Mac OS X v10.3 and later.

See Also

`- initWithContentsOfURL:` (page 2001)

Declared In

`NSXMLParser.h`

lineNumber

Returns the line number of the XML document being processed by the receiver.

```
- (NSInteger)lineNumber
```

Discussion

You may invoke this method once a parsing operation has begun or after an error occurs.

Availability

Available in Mac OS X v10.3 and later.

See Also

`- columnNumber` (page 2000)

Declared In

`NSXMLParser.h`

parse

Starts the event-driven parsing operation.

```
- (BOOL)parse
```

Return Value

`YES` if parsing is successful and `NO` in there is an error or if the parsing operation is aborted.

Availability

Available in Mac OS X v10.3 and later.

See Also
- `abortParsing` (page 2000)
- `parserError` (page 2003)

Related Sample Code
ImageMapExample

Declared In
`NSXMLParser.h`

parserError

Returns an `NSError` object from which you can obtain information about a parsing error.

```
- (NSError *)parserError
```

Discussion
You may invoke this method after a parsing operation abnormally terminates to determine the cause of error.

Availability
Available in Mac OS X v10.3 and later.

See Also
- `abortParsing` (page 2000)
- `parse` (page 2002)

Declared In
`NSXMLParser.h`

publicID

Returns the public identifier of the external entity referenced in the XML document.

```
- (NSString *)publicID
```

Discussion
You may invoke this method once a parsing operation has begun or after an error occurs.

Availability
Available in Mac OS X v10.3 and later.

See Also
- `systemID` (page 2007)

Declared In
`NSXMLParser.h`

setDelegate:

Sets the receiver's delegate.

```
- (void)setDelegate:(id)delegate
```

Parameters

delegate

An object that is the new delegate. It is not retained.

Availability
Available in Mac OS X v10.3 and later.

See Also
- delegate (page 2001)

Related Sample Code
ImageMapExample

Declared In
NSXMLParser.h

setShouldProcessNamespaces:

Specifies whether the receiver reports the namespace and the qualified name of an element in related delegation methods .

```
- (void)setShouldProcessNamespaces:(BOOL)shouldProcessNamespaces
```

Parameters

shouldProcessNamespaces

YES if the receiver should report the namespace and qualified name of each element, NO otherwise. The default value is NO.

Discussion
The invoked delegation methods are
parser:didStartElement:namespaceURI:qualifiedName:attributes: (page 2008) and
parser:didEndElement:namespaceURI:qualifiedName: (page 2007).

Availability
Available in Mac OS X v10.3 and later.

See Also
- shouldProcessNamespaces (page 2005)

Declared In
NSXMLParser.h

setShouldReportNamespacePrefixes:

Specifies whether the receiver reports the scope of namespace declarations using related delegation methods.

```
- (void)setShouldReportNamespacePrefixes:(BOOL)shouldReportNamespacePrefixes
```

Parameters

shouldReportNamespacePrefixes

YES if the receiver should report the scope of namespace declarations, NO otherwise. The default value is NO.

Discussion
The invoked delegation methods are `parser:didStartMappingPrefix:toURI:` (page 2009) and `parser:didEndMappingPrefix:` (page 2008).

Availability
Available in Mac OS X v10.3 and later.

See Also
`- shouldReportNamespacePrefixes` (page 2006)

Declared In
`NSXMLParser.h`

setShouldResolveExternalEntities:

Specifies whether the receiver reports declarations of external entities using the delegate method `parser:foundExternalEntityDeclarationWithName:publicID:systemID:` (page 2011).

```
- (void)setShouldResolveExternalEntities:(BOOL)shouldResolveExternalEntities
```

Parameters
shouldResolveExternalEntities
`YES` if the receiver should report declarations of external entities, `NO` otherwise. The default value is `NO`.

Discussion
If you pass in `YES`, you may cause other I/O operations, either network-based or disk-based, to load the external DTD.

Availability
Available in Mac OS X v10.3 and later.

See Also
`- shouldResolveExternalEntities` (page 2006)

Declared In
`NSXMLParser.h`

shouldProcessNamespaces

Indicates whether the receiver reports the namespace and the qualified name of an element in related delegation methods.

```
- (BOOL)shouldProcessNamespaces
```

Return Value
`YES` if the receiver reports namespace and qualified name, `NO` otherwise.

Discussion
The invoked delegation methods are `parser:didStartElement:namespaceURI:qualifiedName:attributes:` (page 2008) and `parser:didEndElement:namespaceURI:qualifiedName:` (page 2007).

Availability
Available in Mac OS X v10.3 and later.

See Also
- `setShouldProcessNamespaces:` (page 2004)

Declared In
`NSXMLParser.h`

shouldReportNamespacePrefixes

Indicates whether the receiver reports the prefixes indicating the scope of namespace declarations using related delegation methods.

```
- (BOOL)shouldReportNamespacePrefixes
```

Return Value
`YES` if the receiver reports the scope of namespace declarations, `NO` otherwise. The default value is `NO`.

Discussion
The invoked delegation methods are `parser:didStartMappingPrefix:toURI:` (page 2009) and `parser:didEndMappingPrefix:` (page 2008).

Availability
Available in Mac OS X v10.3 and later.

See Also
- `setShouldReportNamespacePrefixes:` (page 2004)

Declared In
`NSXMLParser.h`

shouldResolveExternalEntities

Indicates whether the receiver reports declarations of external entities using the delegate method `parser:foundExternalEntityDeclarationWithName:publicID:systemID:` (page 2011).

```
- (BOOL)shouldResolveExternalEntities
```

Return Value
`YES` if the receiver reports declarations of external entities, `NO` otherwise. The default value is `NO`.

Availability
Available in Mac OS X v10.3 and later.

See Also
- `setShouldResolveExternalEntities:` (page 2005)

Declared In
`NSXMLParser.h`

systemID

Returns the system identifier of the external entity referenced in the XML document.

```
- (NSString *)systemID
```

Discussion
You may invoke this method once a parsing operation has begun or after an error occurs.

Availability
Available in Mac OS X v10.3 and later.

See Also
- publicID (page 2003)

Declared In
NSXMLParser.h

Delegate Methods

parser:didEndElement:namespaceURI:qualifiedName:

Sent by a parser object to its delegate when it encounters an end tag for a specific element.

```
- (void)parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName
    namespaceURI:(NSString *)namespaceURI qualifiedName:(NSString *)qName
```

Parameters

parser
A parser object.

elementName
A string that is the name of an element (in its end tag).

namespaceURI
If namespace processing is turned on, contains the URI for the current namespace as a string object.

qName
If namespace processing is turned on, contains the qualified name for the current namespace as a string object..

Availability
Available in Mac OS X v10.3 and later.

See Also
- parser:didStartElement:namespaceURI:qualifiedName:attributes: (page 2008)
- setShouldProcessNamespaces: (page 2004)

Declared In
NSXMLParser.h

parser:didEndMappingPrefix:

Sent by a parser object to its delegate when a given namespace prefix goes out of scope.

```
- (void)parser:(NSXMLParser *)parser didEndMappingPrefix:(NSString *)prefix
```

Parameters

parser

A parser object.

prefix

A string that is a namespace prefix.

Discussion

The parser sends this message only when namespace-prefix reporting is turned on through the `setShouldReportNamespacePrefixes:` (page 2004) method.

Availability

Available in Mac OS X v10.3 and later.

See Also

- `parser:didStartMappingPrefix:toURI:` (page 2009)

Declared In

`NSXMLParser.h`

parser:didStartElement:namespaceURI:qualifiedName:attributes:

Sent by a parser object to its delegate when it encounters a start tag for a given element.

```
- (void)parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName
    namespaceURI:(NSString *)namespaceURI qualifiedName:(NSString *)qualifiedName
     attributes:(NSDictionary *)attributeDict
```

Parameters

parser

A parser object.

elementName

A string that is the name of an element (in its start tag).

namespaceURI

If namespace processing is turned on, contains the URI for the current namespace as a string object.

qualifiedName

If namespace processing is turned on, contains the qualified name for the current namespace as a string object..

attributeDict

A dictionary that contains any attributes associated with the element. Keys are the names of attributes, and values are attribute values.

Availability

Available in Mac OS X v10.3 and later.

See Also

- `parser:didEndElement:namespaceURI:qualifiedName:` (page 2007)
- `setShouldProcessNamespaces:` (page 2004)

Declared In
NSXMLParser.h

parser:didStartMappingPrefix:toURI:

Sent by a parser object to its delegate the first time it encounters a given namespace prefix, which is mapped to a URI.

```
- (void)parser:(NSXMLParser *)parser didStartMappingPrefix:(NSString *)prefix
    toURI:(NSString *)namespaceURI
```

Parameters

parser
A parser object.

prefix
A string that is a namespace prefix.

namespaceURI
A string that specifies a namespace URI.

Discussion
The parser object sends this message only when namespace-prefix reporting is turned on through the `setShouldReportNamespacePrefixes:` (page 2004) method.

Availability
Available in Mac OS X v10.3 and later.

See Also
`- parser:didEndMappingPrefix:` (page 2008)

Declared In
NSXMLParser.h

parser:foundAttributeDeclarationWithName:forElement:type:defaultValue:

Sent by a parser object to its delegate when it encounters a declaration of an attribute that is associated with a specific element.

```
- (void)parser:(NSXMLParser *)parser foundAttributeDeclarationWithName:(NSString
    *)attributeName forElement:(NSString *)elementName type:(NSString *)type
    defaultValue:(NSString *)defaultValue
```

Parameters

parser
An `NSXMLParser` object parsing XML.

attributeName
A string that is the name of an attribute.

elementName
A string that is the name of an element that has the attribute *attributeName*.

type
A string, such as "ENTITY", "NOTATION", or "ID", that indicates the type of the attribute.

defaultValue
A string that specifies the default value of the attribute.

Availability
Available in Mac OS X v10.3 and later.

See Also
- `parser:didStartElement:namespaceURI:qualifiedName:attributes:` (page 2008)

Declared In
`NSXMLParser.h`

parser:foundCDATA:

Sent by a parser object to its delegate when it encounters a CDATA block.

```
- (void)parser:(NSXMLParser *)parser foundCDATA:(NSData *)CDATABlock
```

Parameters
parser
An `NSXMLParser` object parsing XML.

CDATABlock
A data object containing a block of CDATA.

Discussion
Through this method the parser object passes the contents of the block to its delegate in an `NSData` object. The CDATA block is character data that is ignored by the parser. The encoding of the character data is UTF-8. To convert the data object to a string object, use the `NSString` method `initWithData:encoding:` (page 1572).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`NSXMLParser.h`

parser:foundCharacters:

Sent by a parser object to provide its delegate with a string representing all or part of the characters of the current element.

```
- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string
```

Parameters
parser
A parser object.

string
A string representing the complete or partial textual content of the current element.

Discussion
The parser object may send the delegate several `parser:foundCharacters:` messages to report the characters of an element. Because *string* may be only part of the total character content for the current element, you should append it to the current accumulation of characters until the element changes.

Availability
Available in Mac OS X v10.3 and later.

Declared In
NSXMLParser.h

parser:foundComment:

Sent by a parser object to its delegate when it encounters a comment in the XML.

```
- (void)parser:(NSXMLParser *)parser foundComment:(NSString *)comment
```

Parameters

parser
An NSXMLParser object parsing XML.

comment
A string that is a the content of a comment in the XML.

Availability
Available in Mac OS X v10.3 and later.

Declared In
NSXMLParser.h

parser:foundElementDeclarationWithName:model:

Sent by a parser object to its delegate when it encounters a declaration of an element with a given model.

```
- (void)parser:(NSXMLParser *)parser foundElementDeclarationWithName:(NSString
    *)elementName model:(NSString *)model
```

Parameters

parser
An NSXMLParser object parsing XML.

elementName
A string that is the name of an element.

model
A string that specifies a model for *elementName*.

Availability
Available in Mac OS X v10.3 and later.

See Also
- parser:didStartElement:namespaceURI:qualifiedName:attributes: (page 2008)

Declared In
NSXMLParser.h

parser:foundExternalEntityDeclarationWithName:publicID:systemID:

Sent by a parser object to its delegate when it encounters an external entity declaration.

```
- (void)parser:(NSXMLParser *)parser foundExternalEntityDeclarationWithName:(NSString
    *)entityName publicID:(NSString *)publicID systemID:(NSString *)systemID
```

Parameters

parser

An `NSXMLParser` object parsing XML.

entityName

A string that is the name of an entity.

publicID

A string that specifies the public ID associated with *entityName*.

systemID

A string that specifies the system ID associated with *entityName*.

Availability
Available in Mac OS X v10.3 and later.

See Also
- `parser:foundInternalEntityDeclarationWithName:value:` (page 2013)
- `parser:foundUnparsedEntityDeclarationWithName:publicID:systemID:notationName:` (page 2014)
- `parser:resolveExternalEntityName:systemID:` (page 2015)

Declared In
`NSXMLParser.h`

parser:foundIgnorableWhitespace:

Reported by a parser object to provide its delegate with a string representing all or part of the ignorable whitespace characters of the current element.

```
- (void)parser:(NSXMLParser *)parser foundIgnorableWhitespace:(NSString
    *)whitespaceString
```

Parameters

parser

A parser object.

whitespaceString

A string representing all or part of the ignorable whitespace characters of the current element.

Discussion
All the whitespace characters of the element (including carriage returns, tabs, and new-line characters) may not be provided through an individual invocation of this method. The parser may send the delegate several `parser:foundIgnorableWhitespace:` messages to report the whitespace characters of an element. You should append the characters in each invocation to the current accumulation of characters.

Availability
Available in Mac OS X v10.3 and later.

See Also
- `parser:foundCharacters:` (page 2010)

Declared In
NSXMLParser.h

parser:foundInternalEntityDeclarationWithName:value:

Sent by a parser object to the delegate when it encounters an internal entity declaration.

```
- (void)parser:(NSXMLParser *)parser foundInternalEntityDeclarationWithName:(NSString
    *)name value:(NSString *)value
```

Parameters

parser
An NSXMLParser object parsing XML.

name
A string that is the declared name of an internal entity.

value
A string that is the value of entity *name*.

Availability
Available in Mac OS X v10.3 and later.

See Also
- parser:foundExternalEntityDeclarationWithName:publicID:systemID: (page 2011)
- parser:foundUnparsedEntityDeclarationWithName:publicID:systemID:notationName: (page 2014)

Declared In
NSXMLParser.h

parser:foundNotationDeclarationWithName:publicID:systemID:

Sent by a parser object to its delegate when it encounters a notation declaration.

```
- (void)parser:(NSXMLParser *)parser foundNotationDeclarationWithName:(NSString
    *)name publicID:(NSString *)publicID systemID:(NSString *)systemID
```

Parameters

parser
An NSXMLParser object parsing XML.

name
A string that is the name of the notation.

publicID
A string specifying the public ID associated with the notation *name*.

systemID
A string specifying the system ID associated with the notation *name*.

Availability
Available in Mac OS X v10.3 and later.

Declared In
NSXMLParser.h

parser:foundProcessingInstructionWithTarget:data:

Sent by a parser object to its delegate when it encounters a processing instruction.

```
- (void)parser:(NSXMLParser *)parser foundProcessingInstructionWithTarget:(NSString
    *)target data:(NSString *)data
```

Parameters

parser

A parser object.

target

A string representing the target of a processing instruction.

data

A string representing the data for a processing instruction.

Availability
Available in Mac OS X v10.3 and later.

Declared In
NSXMLParser.h

parser:foundUnparsedEntityDeclarationWithName:publicID:systemID:notationName:

Sent by a parser object to its delegate when it encounters an unparsed entity declaration.

```
- (void)parser:(NSXMLParser *)parser foundUnparsedEntityDeclarationWithName:(NSString
    *)name publicID:(NSString *)publicID systemID:(NSString *)systemID
   notationName:(NSString *)notationName
```

Parameters

parser

An NSXMLParser object parsing XML.

name

A string that is the name of the unparsed entity in the declaration.

publicID

A string specifying the public ID associated with the entity *name*.

systemID

A string specifying the system ID associated with the entity *name*.

notationName

A string specifying a notation of the declaration of entity *name*.

Availability
Available in Mac OS X v10.3 and later.

See Also
- parser:foundExternalEntityDeclarationWithName:publicID:systemID: (page 2011)
- parser:foundInternalEntityDeclarationWithName:value: (page 2013)
- parser:resolveExternalEntityName:systemID: (page 2015)

Declared In
NSXMLParser.h

parser:parseErrorOccurred:

Sent by a parser object to its delegate when it encounters a fatal error.

```
- (void)parser:(NSXMLParser *)parser parseErrorOccurred:(NSError *)parseError
```

Parameters

parser

A parser object.

parseError

An `NSError` object describing the parsing error that occurred.

Discussion

When this method is invoked, parsing is stopped. For further information about the error, you can query *parseError* or you can send the receiver a `parserError` (page 2003) message. You can also send the parser `lineNumber` (page 2002) and `columnNumber` (page 2000) messages to further isolate where the error occurred. Typically you implement this method to display information about the error to the user.

Availability

Available in Mac OS X v10.3 and later.

See Also

`- parser:validationErrorOccurred:` (page 2016)

Declared In

`NSXMLParser.h`

parser:resolveExternalEntityName:systemID:

Sent by a parser object to its delegate when it encounters a given external entity with a specific system ID.

```
- (NSData *)parser:(NSXMLParser *)parser resolveExternalEntityName:(NSString
    *)entityName systemID:(NSString *)systemID
```

Parameters

parser

A parser object.

entityName

A string that specifies the external name of an entity.

systemID

A string that specifies the system ID for the external entity.

Return Value

An `NSData` object that contains the resolution of the given external entity.

Discussion

The delegate can resolve the external entity (for example, locating and reading an externally declared DTD) and provide the result to the parser object as an `NSData` object.

Availability

Available in Mac OS X v10.3 and later.

See Also

`- parser:foundExternalEntityDeclarationWithName:publicID:systemID:` (page 2011)

- parser:foundUnparsedEntityDeclarationWithName:publicID:systemID:notationName: (page 2014)

Declared In
NSXMLParser.h

parser:validationErrorOccurred:

Sent by a parser object to its delegate when it encounters a fatal validation error. NSXMLParser currently does not invoke this method and does not perform validation.

```
- (void)parser:(NSXMLParser *)parser validationErrorOccurred:(NSError *)validError
```

Parameters

parser
A parser object.

validError
An NSError object describing the validation error that occurred.

Discussion
If you want to validate an XML document, use the validation features of the NSXMLDocument class.

Availability
Available in Mac OS X v10.3 and later.

See Also
- parser:parseErrorOccurred: (page 2015)

Declared In
NSXMLParser.h

parserDidEndDocument:

Sent by the parser object to the delegate when it has successfully completed parsing.

```
- (void)parserDidEndDocument:(NSXMLParser *)parser
```

Parameters

parser
A parser object.

Availability
Available in Mac OS X v10.3 and later.

See Also
- parserDidStartDocument: (page 2017)

Declared In
NSXMLParser.h

parserDidStartDocument:

Sent by the parser object to the delegate when it begins parsing a document.

```
- (void)parserDidStartDocument:(NSXMLParser *)parser
```

Parameters

parser

A parser object.

Availability
Available in Mac OS X v10.3 and later.

See Also
- parserDidEndDocument: (page 2016)

Declared In
NSXMLParser.h

Constants

NSXMLParserErrorDomain

This constant defines the NSXMLParser error domain.

```
NSString * const NSXMLParserErrorDomain
```

Constants
NSXMLParserErrorDomain

Indicates an error in XML parsing.

Used by NSError.

Available in Mac OS X v10.3 and later.

Declared in NSXMLParser.h.

Declared In
NSXMLParser.h

NSXMLParserError

A type defined for the contants listed in "Parser Error Constants" (page 2018).

```
typedef NSInteger NSXMLParserError;
```

Availability
Available in Mac OS X v10.3 and later.

Declared In
NSXMLParser.h

Parser Error Constants

The following error types are defined by `NSXMLParser`.

```
typedef enum {
    NSXMLParserInternalError = 1,
    NSXMLParserOutOfMemoryError = 2,
    NSXMLParserDocumentStartError = 3,
    NSXMLParserEmptyDocumentError = 4,
    NSXMLParserPrematureDocumentEndError = 5,
    NSXMLParserInvalidHexCharacterRefError = 6,
    NSXMLParserInvalidDecimalCharacterRefError = 7,
    NSXMLParserInvalidCharacterRefError = 8,
    NSXMLParserInvalidCharacterError = 9,
    NSXMLParserCharacterRefAtEOFError = 10,
    NSXMLParserCharacterRefInPrologError = 11,
    NSXMLParserCharacterRefInEpilogError = 12,
    NSXMLParserCharacterRefInDTDError = 13,
    NSXMLParserEntityRefAtEOFError = 14,
    NSXMLParserEntityRefInPrologError = 15,
    NSXMLParserEntityRefInEpilogError = 16,
    NSXMLParserEntityRefInDTDError = 17,
    NSXMLParserParsedEntityRefAtEOFError = 18,
    NSXMLParserParsedEntityRefInPrologError = 19,
    NSXMLParserParsedEntityRefInEpilogError = 20,
    NSXMLParserParsedEntityRefInInternalSubsetError = 21,
    NSXMLParserEntityReferenceWithoutNameError = 22,
    NSXMLParserEntityReferenceMissingSemiError = 23,
    NSXMLParserParsedEntityRefNoNameError = 24,
    NSXMLParserParsedEntityRefMissingSemiError = 25,
    NSXMLParserUndeclaredEntityError = 26,
    NSXMLParserUnparsedEntityError = 28,
    NSXMLParserEntityIsExternalError = 29,
    NSXMLParserEntityIsParameterError = 30,
    NSXMLParserUnknownEncodingError = 31,
    NSXMLParserEncodingNotSupportedError = 32,
    NSXMLParserStringNotStartedError = 33,
    NSXMLParserStringNotClosedError = 34,
    NSXMLParserNamespaceDeclarationError = 35,
    NSXMLParserEntityNotStartedError = 36,
    NSXMLParserEntityNotFinishedError = 37,
    NSXMLParserLessThanSymbolInAttributeError = 38,
    NSXMLParserAttributeNotStartedError = 39,
    NSXMLParserAttributeNotFinishedError = 40,
    NSXMLParserAttributeHasNoValueError = 41,
    NSXMLParserAttributeRedefinedError = 42,
    NSXMLParserLiteralNotStartedError = 43,
    NSXMLParserLiteralNotFinishedError = 44,
    NSXMLParserCommentNotFinishedError = 45,
    NSXMLParserProcessingInstructionNotStartedError = 46,
    NSXMLParserProcessingInstructionNotFinishedError = 47,
    NSXMLParserNotationNotStartedError = 48,
    NSXMLParserNotationNotFinishedError = 49,
    NSXMLParserAttributeListNotStartedError = 50,
    NSXMLParserAttributeListNotFinishedError = 51,
    NSXMLParserMixedContentDeclNotStartedError = 52,
    NSXMLParserMixedContentDeclNotFinishedError = 53,
    NSXMLParserElementContentDeclNotStartedError = 54,
    NSXMLParserElementContentDeclNotFinishedError = 55,
    NSXMLParserXMLDeclNotStartedError = 56,
    NSXMLParserXMLDeclNotFinishedError = 57,
    NSXMLParserConditionalSectionNotStartedError = 58,
```

```
    NSXMLParserConditionalSectionNotFinishedError = 59,
    NSXMLParserExternalSubsetNotFinishedError = 60,
    NSXMLParserDOCTYPEDeclNotFinishedError = 61,
    NSXMLParserMisplacedCDATAEndStringError = 62,
    NSXMLParserCDATANotFinishedError = 63,
    NSXMLParserMisplacedXMLDeclarationError = 64,
    NSXMLParserSpaceRequiredError = 65,
    NSXMLParserSeparatorRequiredError = 66,
    NSXMLParserNMTOKENRequiredError = 67,
    NSXMLParserNAMERequiredError = 68,
    NSXMLParserPCDATARequiredError = 69,
    NSXMLParserURIRequiredError = 70,
    NSXMLParserPublicIdentifierRequiredError = 71,
    NSXMLParserLTRequiredError = 72,
    NSXMLParserGTRequiredError = 73,
    NSXMLParserLTSlashRequiredError = 74,
    NSXMLParserEqualExpectedError = 75,
    NSXMLParserTagNameMismatchError = 76,
    NSXMLParserUnfinishedTagError = 77,
    NSXMLParserStandaloneValueError = 78,
    NSXMLParserInvalidEncodingNameError = 79,
    NSXMLParserCommentContainsDoubleHyphenError = 80,
    NSXMLParserInvalidEncodingError = 81,
    NSXMLParserExternalStandaloneEntityError = 82,
    NSXMLParserInvalidConditionalSectionError = 83,
    NSXMLParserEntityValueRequiredError = 84,
    NSXMLParserNotWellBalancedError = 85,
    NSXMLParserExtraContentError = 86,
    NSXMLParserInvalidCharacterInEntityError = 87,
    NSXMLParserParsedEntityRefInInternalError = 88,
    NSXMLParserEntityRefLoopError = 89,
    NSXMLParserEntityBoundaryError = 90,
    NSXMLParserInvalidURIError = 91,
    NSXMLParserURIFragmentError = 92,
    NSXMLParserNoDTDError = 94,
    NSXMLParserDelegateAbortedParseError = 512
} NSXMLParserError;
```

Constants

`NSXMLParserInternalError`

The parser object encountered an internal error.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserOutOfMemoryError`

The parser object ran out of memory.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserDocumentStartError`

The parser object is unable to start parsing.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserEmptyDocumentError`
The document is empty.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserPrematureDocumentEndError`
The document ended unexpectedly.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserInvalidHexCharacterRefError`
Invalid hexadecimal character reference encountered.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserInvalidDecimalCharacterRefError`
Invalid decimal character reference encountered.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserInvalidCharacterRefError`
Invalid character reference encountered.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserInvalidCharacterError`
Invalid character encountered.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserCharacterRefAtEOFError`
Target of character reference cannot be found.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserCharacterRefInPrologError`
Invalid character found in the prolog.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserCharacterRefInEpilogError`
Invalid character found in the epilog.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserCharacterRefInDTDError`
Invalid character encountered in the DTD.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEntityRefAtEOFError`
Target of entity reference is not found.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEntityRefInPrologError`
Invalid entity reference found in the prolog.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEntityRefInEpilogError`
Invalid entity reference found in the epilog.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEntityRefInDTDError`
Invalid entity reference found in the DTD.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserParsedEntityRefAtEOFError`
Target of parsed entity reference is not found.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserParsedEntityRefInPrologError`
Target of parsed entity reference is not found in prolog.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserParsedEntityRefInEpilogError`
Target of parsed entity reference is not found in epilog.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserParsedEntityRefInInternalSubsetError`
Target of parsed entity reference is not found in internal subset.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEntityReferenceWithoutNameError`
Entity reference is without name.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEntityReferenceMissingSemiError`
Entity reference is missing semicolon.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserParsedEntityRefNoNameError`
Parsed entity reference is without an entity name.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserParsedEntityRefMissingSemiError`
Parsed entity reference is missing semicolon.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserUndeclaredEntityError`
Entity is not declared.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserUnparsedEntityError`
Cannot parse entity.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserEntityIsExternalError`
Cannot parse external entity.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserEntityIsParameterError`
Entity is a parameter.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserUnknownEncodingError`
Document encoding is unknown.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserEncodingNotSupportedError`
Document encoding is not supported.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserStringNotStartedError`
String is not started.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserStringNotClosedError`
String is not closed.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserNamespaceDeclarationError`
Invalid namespace declaration encountered.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEntityNotStartedError`
Entity is not started.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEntityNotFinishedError`
Entity is not finished.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserLessThanSymbolInAttributeError`
Angle bracket is used in attribute.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserAttributeNotStartedError`
Attribute is not started.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserAttributeNotFinishedError`
Attribute is not finished.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserAttributeHasNoValueError`
Attribute doesn't contain a value.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserAttributeRedefinedError`
Attribute is redefined.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserLiteralNotStartedError`
Literal is not started.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserLiteralNotFinishedError`
Literal is not finished.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserCommentNotFinishedError`
Comment is not finished.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserProcessingInstructionNotStartedError`
Processing instruction is not started.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserProcessingInstructionNotFinishedError`
Processing instruction is not finished.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserNotationNotStartedError`
Notation is not started.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserNotationNotFinishedError`
Notation is not finished.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserAttributeListNotStartedError`
Attribute list is not started.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserAttributeListNotFinishedError`
Attribute list is not finished.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserMixedContentDeclNotStartedError`
Mixed content declaration is not started.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserMixedContentDeclNotFinishedError`
Mixed content declaration is not finished.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserElementContentDeclNotStartedError`
Element content declaration is not started.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserElementContentDeclNotFinishedError`
Element content declaration is not finished.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserXMLDeclNotStartedError`
XML declaration is not started.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserXMLDeclNotFinishedError`
XML declaration is not finished.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserConditionalSectionNotStartedError`
Conditional section is not started.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserConditionalSectionNotFinishedError`
Conditional section is not finished.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserExternalSubsetNotFinishedError`
External subset is not finished.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserDOCTYPEDeclNotFinishedError`
Document type declaration is not finished.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserMisplacedCDATAEndStringError`
Misplaced CDATA end string.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserCDATANotFinishedError`
CDATA block is not finished.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserMisplacedXMLDeclarationError`
Misplaced XML declaration.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserSpaceRequiredError`
Space is required.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserSeparatorRequiredError`
Separator is required.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserNMTOKENRequiredError`
Name token is required.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserNAMERequiredError`
Name is required.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserPCDATARequiredError`
CDATA is required.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserURIRequiredError`
URI is required.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserPublicIdentifierRequiredError`
Public identifier is required.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserLTRequiredError`
Left angle bracket is required.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserGTRequiredError`
Right angle bracket is required.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserLTSlashRequiredError`
Left angle bracket slash is required.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEqualExpectedError`
Equal sign expected.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserTagNameMismatchError`
Tag name mismatch.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserUnfinishedTagError`
Unfinished tag found.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserStandaloneValueError`
Standalone value found.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserInvalidEncodingNameError`
Invalid encoding name found.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserCommentContainsDoubleHyphenError`
Comment contains double hyphen.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserInvalidEncodingError`
Invalid encoding.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserExternalStandaloneEntityError`
External standalone entity.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserInvalidConditionalSectionError`
Invalid conditional section.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEntityValueRequiredError`
Entity value is required.
Available in Mac OS X v10.3 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserNotWellBalancedError`
Document is not well balanced.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserExtraContentError`
Error in content found.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserInvalidCharacterInEntityError`
Invalid character in entity found.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserParsedEntityRefInInternalError`
Internal error in parsed entity reference found.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserEntityRefLoopError`
Entity reference loop encountered.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserEntityBoundaryError`
Entity boundary error.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserInvalidURIError`
Invalid URI specified.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserURIFragmentError`
URI fragment.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserNoDTDError`
Missing DTD.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserDelegateAbortedParseError`
Delegate aborted parse.

Available in Mac OS X v10.3 and later.

Declared in `NSXMLParser.h`.

Declared In
`NSXMLParser.h`

Protocols

NSCoding Protocol Reference

Adopted by	Various Cocoa classes
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSObject.h
Companion guide	Archives and Serializations Programming Guide for Cocoa
Related sample code	Squiggles

Overview

The `NSCoding` protocol declares the two methods that a class must implement so that instances of that class can be encoded and decoded. This capability provides the basis for archiving (where objects and other structures are stored on disk) and distribution (where objects are copied to different address spaces).

In keeping with object-oriented design principles, an object being encoded or decoded is responsible for encoding and decoding its instance variables. A coder instructs the object to do so by invoking `encodeWithCoder:` (page 2034) or `initWithCoder:` (page 2034). `encodeWithCoder:` (page 2034) instructs the object to encode its instance variables to the coder provided; an object can receive this method any number of times. `initWithCoder:` (page 2034) instructs the object to initialize itself from data in the coder provided; as such, it replaces any other initialization method and is sent only once per object. Any object class that should be codable must adopt the `NSCoding` protocol and implement its methods.

It is important to consider the possible types of archiving that a coder supports. On Mac OS X version 10.2 and later, keyed archiving is preferred. You may, however, need to support classic archiving. For details, see *Archives and Serializations Programming Guide for Cocoa*.

Tasks

Initializing with a Coder

`- initWithCoder:` (page 2034)
Returns an object initialized from data in a given unarchiver.

Encoding with a Coder

Instance Methods

encodeWithCoder:

Encodes the receiver using a given archiver.

```
- (void)encodeWithCoder:(NSCoder *)encoder
```

Parameters

encoder

An archiver object.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSObject.h

initWithCoder:

Returns an object initialized from data in a given unarchiver.

```
- (id)initWithCoder:(NSCoder *)decoder
```

Parameters

decoder

An unarchiver object.

Return Value
self, initialized using the data in *decoder*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSObject.h

NSComparisonMethods Protocol Reference

(informal protocol)

Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSScriptWhoseTests.h
Companion guide	Cocoa Scripting Guide

Overview

This informal protocol defines a set of default comparison methods useful for the comparisons in `NSSpecifierTest`.

If you have scriptable objects that need to perform comparisons for scripting purposes, you may need to implement some of the methods declared in `NSScriptingComparisonMethods`. The default implementation provided for many of these methods by `NSObject` is appropriate for objects that implement a single comparison method whose selector, signature, and description match the following:

```
- (NSComparisonResult)compare:(id)object;
```

This method should return `NSOrderedAscending` if the receiver is less than *`object`*, `NSOrderedDescending` if the receiver is greater than *`object`*, and `NSOrderedSame` if the receiver and *`object`* are equal. For example, `NSString` does not implement most of the methods declared in this informal protocol, but `NSString` objects still handle messages conforming to this protocol properly because `NSString` implements a `compare:` method that meets the necessary requirements. Cocoa also includes appropriate `compare:` method implementations for the `NSDate`, `NSDecimalNumber`, and `NSValue` classes.

Tasks

Performing Comparisons

`- doesContain:` (page 2036)
Returns a Boolean value that indicates whether the receiver contains a given object.

`- isCaseInsensitiveLike:` (page 2036)
Returns a Boolean value that indicates whether receiver is considered to be “like” a given string when the case of characters in the receiver is ignored.

`- isEqualTo:` (page 2037)
Returns a Boolean value that indicates whether the receiver is equal to another given object.

`- isGreaterThan:` (page 2037)
Returns a Boolean value that indicates whether the receiver is greater than another given object.

- `isGreaterThanOrEqualTo:` (page 2038)
  Returns a Boolean value that indicates whether the receiver is greater than or equal to another given object.
- `isLessThan:` (page 2039)
  Returns a Boolean value that indicates whether the receiver is less than another given object.
- `isLessThanOrEqualTo:` (page 2039)
  Returns a Boolean value that indicates whether the receiver is less than or equal to another given object.
- `isLike:` (page 2040)
  Returns a Boolean value that indicates whether the receiver is "like" another given object.
- `isNotEqualTo:` (page 2040)
  Returns a Boolean value that indicates whether the receiver is not equal to another given object.

Instance Methods

doesContain:

Returns a Boolean value that indicates whether the receiver contains a given object.

```
- (BOOL)doesContain:(id)object
```

Parameters

object
  The object to search for in the receiver.

Return Value
`YES` if the receiver contains *object*, otherwise `NO`.

Discussion
Currently, `doesContain:` messages are never sent to any object from within Cocoa itself.

The default implementation for this method provided by `NSObject` returns `YES` if the receiver is actually an `NSArray` object and an `indexOfObjectIdenticalTo:` (page 124) message sent to the same object would return something other than `NSNotFound`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptWhoseTests.h`

isCaseInsensitiveLike:

Returns a Boolean value that indicates whether receiver is considered to be “like” a given string when the case of characters in the receiver is ignored.

```
- (BOOL)isCaseInsensitiveLike:(NSString *)aString
```

Parameters

aString

The string with which to compare the receiver.

Return Value

YES if the receiver is considered to be "like" *aString* when the case of characters in the receiver is ignored, otherwise NO.

Discussion

Currently, isCaseInsensitiveLike: messages are never sent to any object from within Cocoa itself.

The default implementation for this method provided by NSObject returns NO. NSString also provides an implementation of this method, which returns YES if the receiver matches a pattern described by *aString*, ignoring the case of the characters in the receiver.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSScriptWhoseTests.h

isEqualTo:

Returns a Boolean value that indicates whether the receiver is equal to another given object.

```
- (BOOL)isEqualTo:(id)object
```

Parameters

object

The object with which to compare the receiver.

Return Value

YES if the receiver is equal to *object*, otherwise NO. In effect returns NO if receiver is nil.

Discussion

During the evaluation of an NSWhoseSpecifier object that contains a test whose operator is NSEqualToComparison, an isEqualTo: message may be sent to each potentially specified object, if neither the potentially specified object nor the object being tested against implements a scriptingIsEqualTo: (page 2115) method.

The default implementation for this method provided by NSObject returns YES if an isEqualTo: message sent to the same object would return YES.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSScriptWhoseTests.h

isGreaterThan:

Returns a Boolean value that indicates whether the receiver is greater than another given object.

```
- (BOOL)isGreaterThan:(id)object
```

Parameters

object

The object with which to compare the receiver.

Return Value

YES if the receiver is greater than *object*, otherwise NO.

Discussion

During the evaluation of an NSWhoseSpecifier object that contains a test whose operator is NSGreaterThanComparison, an isGreaterThan: message may be sent to each potentially specified object, if the potentially specified object does not implement a scriptingIsGreaterThan: (page 2115) method and the object being tested against does not implement a scriptingIsLessThanOrEqualTo: (page 2116) method.

The default implementation for this method provided by NSObject returns YES if a compare: message sent to the same object would return NSOrderedDescending.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSScriptWhoseTests.h

isGreaterThanOrEqualTo:

Returns a Boolean value that indicates whether the receiver is greater than or equal to another given object.

```
- (BOOL)isGreaterThanOrEqualTo:(id)object
```

Parameters

object

The object with which to compare the receiver.

Return Value

YES if the receiver is greater than or equal to *object*, otherwise NO.

Discussion

During the evaluation of an NSWhoseSpecifier object that contains a test whose operator is NSGreaterThanOrEqualToComparison, an isGreaterThanOrEqualTo: message may be sent to each potentially specified object, if the potentially specified object does not implement a scriptingIsGreaterThanOrEqualTo: (page 2115) method and the object being tested against does not implement a scriptingIsLessThan: (page 2116) method.

The default implementation for this method provided by NSObject returns YES if a compare: message sent to the same object would return NSOrderedSame or NSOrderedDescending.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSScriptWhoseTests.h

isLessThan:

Returns a Boolean value that indicates whether the receiver is less than another given object.

```
- (BOOL)isLessThan:(id)object
```

Parameters

object

The object with which to compare the receiver.

Return Value

`YES` if the receiver is less than *object*, otherwise `NO`.

Discussion

During the evaluation of an `NSWhoseSpecifier` object that contains a test whose operator is `NSLessThanComparison`, an `isLessThan:` message may be sent to each potentially specified object, if the potentially specified object does not implement a `scriptingIsLessThan:` (page 2116) method and the object being tested against does not implement a `scriptingIsGreaterThanOrEqualTo:` (page 2115) method.

The default implementation for this method provided by `NSObject` method returns `YES` if a `compare:` message sent to the same object would return `NSOrderedAscending`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSScriptWhoseTests.h`

isLessThanOrEqualTo:

Returns a Boolean value that indicates whether the receiver is less than or equal to another given object.

```
- (BOOL)isLessThanOrEqualTo:(id)object
```

Parameters

object

The object with which to compare the receiver.

Return Value

`YES` if the receiver is less than or equal to *object*, otherwise `NO`.

Discussion

During the evaluation of an `NSWhoseSpecifier` object that contains a test whose operator is `NSLessThanOrEqualToComparison`, an `isLessThanOrEqualTo:` message may be sent to each potentially specified object, if the potentially specified object does not implement a `scriptingIsLessThanOrEqualTo:` (page 2116) method and the object being tested against does not implement a `scriptingIsGreaterThan:` (page 2115) method.

The default implementation for this method provided by `NSObject` method returns `YES` if a `compare:` message sent to the same object would return `NSOrderedAscending` or `NSOrderedSame`.

Availability

Available in Mac OS X v10.0 and later.

Declared In
NSScriptWhoseTests.h

isLike:

Returns a Boolean value that indicates whether the receiver is "like" another given object.

```
- (BOOL)isLike:(NSString *)object
```

Parameters

object
The object with which to compare the receiver.

Return Value
YES if the receiver is considered to be “like” *object*, otherwise NO.

Discussion
Currently, isLike: messages are never sent to any object from within Cocoa itself.

The default implementation for this method provided by NSObject method returns NO. NSString also provides an implementation of this method, which returns YES if the receiver matches a pattern described by *object*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptWhoseTests.h

isNotEqualTo:

Returns a Boolean value that indicates whether the receiver is not equal to another given object.

```
- (BOOL)isNotEqualTo:(id)object
```

Parameters

object
The object with which to compare the receiver.

Return Value
YES if the receiver is not equal to *object*, otherwise NO.

Discussion
Currently, isNotEqualTo: messages are never sent to any object from within Cocoa itself.

The default implementation for this method provided by NSObject method returns YES if an isEqual: message sent to the same object would return NO.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptWhoseTests.h

NSCopying Protocol Reference

Adopted by	Various Cocoa classes
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSObject.h
Companion guide	Memory Management Programming Guide for Cocoa

Overview

The `NSCopying` protocol declares a method for providing functional copies of an object. The exact meaning of "copy" can vary from class to class, but a copy must be a functionally independent object with values identical to the original at the time the copy was made. A copy produced with `NSCopying` is implicitly retained by the sender, who is responsible for releasing it.

`NSCopying` declares one method, `copyWithZone:` (page 2042), but copying is commonly invoked with the convenience method `copy`. The `copy` method is defined for all objects inheriting from `NSObject` and simply invokes `copyWithZone:` (page 2042) with the default zone.

Your options for implementing this protocol are as follows:

- Implement `NSCopying` using `alloc` (page 1152) and `init...` in classes that don't inherit `copyWithZone:` (page 2042).
- Implement `NSCopying` by invoking the superclass's `copyWithZone:` (page 2042) when `NSCopying` behavior is inherited. If the superclass implementation might use the `NSCopyObject` (page 2175) function, make explicit assignments to pointer instance variables for retained objects.
- Implement `NSCopying` by retaining the original instead of creating a new copy when the class and its contents are immutable.

If a subclass inherits NSCopying from its superclass and declares additional instance variables, the subclass has to override `copyWithZone:` (page 2042) to properly handle its own instance variables, invoking the superclass's implementation first.

Tasks

Copying

- copyWithZone: (page 2042)
  Returns a new instance that's a copy of the receiver.

Instance Methods

copyWithZone:

Returns a new instance that's a copy of the receiver.

```
- (id)copyWithZone:(NSZone *)zone
```

Parameters

zone

The zone identifies an area of memory from which to allocate for the new instance. If *zone* is NULL, the new instance is allocated from the default zone, which is returned from the function NSDefaultMallocZone.

Discussion

The returned object is implicitly retained by the sender, who is responsible for releasing it. The copy returned is immutable if the consideration "immutable vs. mutable" applies to the receiving object; otherwise the exact nature of the copy is determined by the class.

Availability

Available in Mac OS X v10.0 and later.

See Also

- mutableCopyWithZone: (page 2094) (NSMutableCopying protocol)
- copy (page 1172) (NSObject class)

Declared In

NSObject.h

NSDecimalNumberBehaviors Protocol Reference

Adopted by	NSDecimalNumberHandler
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSDecimalNumber.h
Companion guide	Number and Value Programming Topics for Cocoa

Overview

The `NSDecimalBehaviors` protocol declares three methods that control the discretionary aspects of working with `NSDecimalNumber` objects.

The `scale` (page 2045) and `roundingMode` (page 2044) methods determine the precision of `NSDecimalNumber`'s return values and the way in which those values should be rounded to fit that precision. The `exceptionDuringOperation:error:leftOperand:rightOperand:` (page 2044) method determines the way in which an `NSDecimalNumber` object should handle different calculation errors.

For an example of a class that adopts the `NSDecimalBehaviors` protocol, see the specification for NSDecimalNumberHandler.

Tasks

Rounding

- `roundingMode` (page 2044)
  Returns the way that `NSDecimalNumber`'s `decimalNumberBy...` methods round their return values.
- `scale` (page 2045)
  Returns the number of digits allowed after the decimal separator.

Handling Errors

- `exceptionDuringOperation:error:leftOperand:rightOperand:` (page 2044)
  Specifies what an `NSDecimalNumber` object will do when it encounters an error.

Instance Methods

exceptionDuringOperation:error:leftOperand:rightOperand:

Specifies what an `NSDecimalNumber` object will do when it encounters an error.

```
- (NSDecimalNumber *)exceptionDuringOperation:(SEL)method
    error:(NSCalculationError)error leftOperand:(NSDecimalNumber *)leftOperand
    rightOperand:(NSDecimalNumber *)rightOperand
```

Parameters

method
The method that was being executed when the error occurred.

error
The type of error that was generated.

leftOperand
The left operand.

rightOperand
The right operand.

Discussion
There are four possible values for *error*, described in NSCalculationError (page 2047). The first three have to do with limits on the ability of `NSDecimalNumber` to represent decimal numbers. An `NSDecimalNumber` object can represent any number that can be expressed as mantissa x 10^exponent, where mantissa is a decimal integer up to 38 digits long, and exponent is between –256 and 256. The fourth results from the caller trying to divide by `0`.

In implementing `exceptionDuringOperation:error:leftOperand:rightOperand:`, you can handle each of these errors in several ways:

- Raise an exception. For an explanation of exceptions, see *Exception Programming Topics for Cocoa*.
- Return `nil`. The calling method will return its value as though no error had occurred. If *error* is `NSCalculationLossOfPrecision`, *method* will return an imprecise value—that is, one constrained to 38 significant digits. If *error* is `NSCalculationUnderflow` or `NSCalculationOverflow`, *method* will return `NSDecimalNumber`'s `notANumber`. You shouldn't return `nil` if *error* is `NSDivideByZero`.
- Correct the error and return a valid `NSDecimalNumber` object. The calling method will use this as its own return value.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSDecimalNumber.h`

roundingMode

Returns the way that `NSDecimalNumber`'s `decimalNumberBy...` methods round their return values.

```
- (NSRoundingMode)roundingMode
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSDecimalNumber.h`

scale

Returns the number of digits allowed after the decimal separator.

```
- (short)scale
```

Return Value
The number of digits allowed after the decimal separator.

Discussion
This method limits the precision of the values returned by `NSDecimalNumber`'s `decimalNumberBy...` methods. If `scale` returns a negative value, it affects the digits before the decimal separator as well. If `scale` returns `NSDecimalNoScale`, the number of digits is unlimited.

Assuming that `roundingMode` (page 2044) returns `NSRoundPlain`, different values of `scale` have the following effects on the number 123.456:

Scale	Return Value
`NSDecimalNoScale`	123.456
2	123.45
0	123
–2	100

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSDecimalNumber.h`

Constants

NSRoundingMode

These constants specify rounding behaviors.

```
typedef enum {
    NSRoundPlain,
    NSRoundDown,
    NSRoundUp,
    NSRoundBankers
} NSRoundingMode;
```

Constants

`NSRoundPlain`

Round to the closest possible return value; when caught halfway between two positive numbers, round up; when caught between two negative numbers, round down.

Available in Mac OS X v10.0 and later.

Declared in `NSDecimal.h`.

`NSRoundDown`

Round return values down.

Available in Mac OS X v10.0 and later.

Declared in `NSDecimal.h`.

`NSRoundUp`

Round return values up.

Available in Mac OS X v10.0 and later.

Declared in `NSDecimal.h`.

`NSRoundBankers`

Round to the closest possible return value; when halfway between two possibilities, return the possibility whose last digit is even.

In practice, this means that, over the long run, numbers will be rounded up as often as they are rounded down; there will be no systematic bias.

Available in Mac OS X v10.0 and later.

Declared in `NSDecimal.h`.

Discussion

The rounding mode matters only if the `scale` (page 2045) method sets a limit on the precision of `NSDecimalNumber` return values. It has no effect if `scale` returns `NSDecimalNoScale`. Assuming that `scale` (page 2045) returns 1, the rounding mode has the following effects on various original values:

Original Value	NSRoundPlain	NSRoundDown	NSRoundUp	NSRoundBankers
1.24	1.2	1.2	1.3	1.2
1.26	1.3	1.2	1.3	1.3
1.25	1.3	1.2	1.3	1.2
1.35	1.4	1.3	1.4	1.4
–1.35	–1.4	–1.4	–1.3	–1.4

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`NSDecimal.h`

NSCalculationError

Calculation error constants used to describe an error in `exceptionDuringOperation:error:leftOperand:rightOperand:` (page 2044).

```
typedef enum {
    NSCalculationNoError = 0,
    NSCalculationLossOfPrecision,
    NSCalculationUnderflow,
    NSCalculationOverflow,
    NSCalculationDivideByZero
} NSCalculationError;
```

Constants

`NSCalculationNoError`

No error occurred.

Available in Mac OS X v10.0 and later.

Declared in `NSDecimal.h`.

`NSCalculationLossOfPrecision`

The number can't be represented in 38 significant digits.

Available in Mac OS X v10.0 and later.

Declared in `NSDecimal.h`.

`NSCalculationOverflow`

The number is too large to represent.

Available in Mac OS X v10.0 and later.

Declared in `NSDecimal.h`.

`NSCalculationUnderflow`

The number is too small to represent.

Available in Mac OS X v10.0 and later.

Declared in `NSDecimal.h`.

`NSCalculationDivideByZero`

The caller tried to divide by `0`.

Available in Mac OS X v10.0 and later.

Declared in `NSDecimal.h`.

Availability

Available in Mac OS X version 10.0 and later.

Declared In

`NSDecimal.h`

NSErrorRecoveryAttempting Protocol Reference

(informal protocol)

Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSErrorRecoveryAttempting.h

Overview

The `NSErrorRecoveryAttempting` informal protocol provides methods that allow your application to attempt to recover from an error. These methods are invoked when an `NSError` object is returned that specifies the implementing object as the error `recoveryAttempter` and the user has selected one of the error's localized recovery options.

Which method is invoked is dependent on how the error is presented to the user. If the error is presented in a document-modal sheet, `attemptRecoveryFromError:optionIndex:delegate:didRecoverSelector:contextInfo:` (page 2050) is invoked. If the error is presented in an application-modal dialog, `attemptRecoveryFromError:optionIndex:` (page 2049) is invoked.

Tasks

Attempting Recovery From Errors

`- attemptRecoveryFromError:optionIndex:delegate:didRecoverSelector:contextInfo:` (page 2050)
Implemented to attempt a recovery from an error noted in an document-modal sheet.

`- attemptRecoveryFromError:optionIndex:` (page 2049)
Implemented to attempt a recovery from an error noted in an application-modal dialog.

Instance Methods

attemptRecoveryFromError:optionIndex:

Implemented to attempt a recovery from an error noted in an application-modal dialog.

```
- (BOOL)attemptRecoveryFromError:(NSError *)error
    optionIndex:(NSUInteger)recoveryOptionIndex
```

Parameters

error

An NSError object that describes the error, including error recovery options.

recoveryOptionIndex

The index of the user selected recovery option in *error*'s localized recovery array.

Return Value

YES if the error recovery was completed successfully, NO otherwise.

Discussion

Invoked when an error alert is been presented to the user in an application-modal dialog, and the user has selected an error recovery option specified by *error*.

Availability

Available in Mac OS X v10.4 and later.

Declared In

NSError.h

attemptRecoveryFromError:optionIndex:delegate:didRecoverSelector:contextInfo:

Implemented to attempt a recovery from an error noted in an document-modal sheet.

```
- (void)attemptRecoveryFromError:(NSError *)error
    optionIndex:(NSUInteger)recoveryOptionIndex delegate:(id)delegate
    didRecoverSelector:(SEL)didRecoverSelector contextInfo:(void *)contextInfo
```

Parameters

error

An NSError object that describes the error, including error recovery options.

recoveryOptionIndex

The index of the user selected recovery option in *error*'s localized recovery array.

delegate

An object that is the modal delegate.

didRecoverSelector

A selector identifying the method implemented by the modal delegate.

contextInfo

Arbitrary data associated with the attempt at error recovery, to be passed to *delegate* in *didRecoverSelector*.

Discussion

Invoked when an error alert is presented to the user in a document-modal sheet, and the user has selected an error recovery option specified by *error*. After recovery is attempted, your implementation should send *delegate* the message specified in *didRecoverSelector*, passing the provided *contextInfo*.

The *didRecoverSelector* should have the following signature:

```
- (void)didPresentErrorWithRecovery:(BOOL)didRecover contextInfo:(void
*)contextInfo;
```

where *didRecover* is YES if the error recovery attempt was successful; otherwise it is NO.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSError.h`

NSFastEnumeration Protocol Reference

Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	Foundation/NSEnumerator.h
Companion guide	The Objective-C 2.0 Programming Language

Overview

The fast enumeration protocol `NSFastEnumeration` must be adopted and implemented by objects used in conjunction with the for language construct used in conjunction with Cocoa objects.

The abstract class `NSEnumerator` provides a convenience implementation that uses `nextObject` (page 558) to return items one at a time. For more details, see Fast Enumeration.

Tasks

Enumeration

`- countByEnumeratingWithState:objects:count:` (page 2053)
Returns by reference a C array of objects over which the sender should iterate, and as the return value the number of objects in the array.

Instance Methods

countByEnumeratingWithState:objects:count:

Returns by reference a C array of objects over which the sender should iterate, and as the return value the number of objects in the array.

```
- (NSUInteger)countByEnumeratingWithState:(NSFastEnumerationState *)state
    objects:(id *)stackbuf
    count:(NSUInteger)len
```

Parameters

state

Context information that is used in the enumeration to, in addition to other possibilities, ensure that the collection has not been mutated.

stackbuf

A C array of objects over which the sender is to iterate.

len

The maximum number of objects to return in *stackbuf*.

Return Value

The number of objects returned in *stackbuf*. Returns 0 when the iteration is finished.

Discussion

The state structure is assumed to be of stack local memory and, from a garbage collection perspective, does not require write-barriers on stores, so you can recast the passed in state structure to one more suitable for your iteration.

Availability

Available in Mac OS X v10.5 and later.

Declared In

NSEnumerator.h

Constants

NSFastEnumerationState

This defines the structure used as contextual information in the NSFastEnumeration protocol.

```
typedef struct {
    unsigned long state;
    id *itemsPtr;
    unsigned long *mutationsPtr;
    unsigned long extra[5];
} NSFastEnumerationState;
```

Fields

state

Arbitrary state information used by the iterator. Typically this is set to 0 at the beginning of the iteration.

itemsPtr

A C array of objects.

mutationsPtr

Arbitrary state information used to detect whether the collection has been mutated.

extra

A C array that you can use to hold returned values.

Discussion

For more information, see countByEnumeratingWithState:objects:count: (page 2053).

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSEnumerator.h`

NSKeyValueCoding Protocol Reference

(informal protocol)

Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSKeyValueCoding.h
Companion guide	Key-Value Coding Programming Guide

Overview

The `NSKeyValueCoding` informal protocol defines a mechanism by which you can access the properties of an object indirectly by name (or key), rather than directly through invocation of an accessor method or as instance variables. Thus, all of an object's properties can be accessed in a consistent manner.

The basic methods for accessing an object's values are `setValue:forKey:` (page 2064), which sets the value for the property identified by the specified key, and `valueForKey:` (page 2070), which returns the value for the property identified by the specified key. The default implementation uses the accessor methods normally implemented by objects (or to access instance variables directly if need be).

Tasks

Getting Values

- `valueForKey:` (page 2070)
  Returns the value for the property identified by a given key.
- `valueForKeyPath:` (page 2071)
  Returns the value for the derived property identified by a given key path.
- `dictionaryWithValuesForKeys:` (page 2060)
  Returns a dictionary containing the property values identified by each of the keys in a given array.
- `valueForUndefinedKey:` (page 2071)
  Invoked by `valueForKey:` (page 2070) when it finds no property corresponding to a given key.
- `mutableArrayValueForKey:` (page 2061)
  Returns a mutable array proxy that provides read-write access to an ordered to-many relationship specified by a given key.
- `mutableArrayValueForKeyPath:` (page 2062)
  Returns a mutable array that provides read-write access to the ordered to-many relationship specified by a given key path.

- `mutableSetValueForKey:` (page 2062)
  Returns a mutable set proxy that provides read-write access to the unordered to-many relationship specified by a given key.
- `mutableSetValueForKeyPath:` (page 2063)
  Returns a mutable set that provides read-write access to the unordered to-many relationship specified by a given key path.

Setting Values

- `setValue:forKeyPath:` (page 2065)
  Sets the value for the property identified by a given key path to a given value.
- `setValuesForKeysWithDictionary:` (page 2066)
  Sets properties of the receiver with values from a given dictionary, using its keys to identify the properties.
- `setNilValueForKey:` (page 2064)
  Invoked by `setValue:forKey:` (page 2064) when it's given a `nil` value for a scalar value (such as an `int` or `float`).
- `setValue:forKey:` (page 2064)
  Sets the property of the receiver specified by a given key to a given value.
- `setValue:forUndefinedKey:` (page 2065)
  Invoked by `setValue:forKey:` (page 2064) when it finds no property for a given key.

Changing Default Behavior

+ `accessInstanceVariablesDirectly` (page 2059)
  Returns a Boolean value that indicates whether the key-value coding methods should access the corresponding instance variable directly on finding no accessor method for a property.

Validation

- `validateValue:forKey:error:` (page 2069)
  Returns a Boolean value that indicates whether the value specified by a given pointer is valid for the property identified by a given key.
- `validateValue:forKeyPath:error:` (page 2069)
  Returns a Boolean value that indicates whether the value specified by a given pointer is valid for a given key path relative to the receiver.

Deprecated Methods

- `handleQueryWithUnboundKey:` (page 2061) Deprecated in Mac OS X v10.3
  Invoked by `valueForKey:` (page 2070) when it finds no property corresponding to *`key`*. (Deprecated. Use `valueForUndefinedKey:` (page 2071) instead.)

- `handleTakeValue:forUnboundKey:` (page 2061) Deprecated in Mac OS X v10.3
  Invoked by `takeValue:forKey:` (page 2068) when it finds no property binding for *key*. (Deprecated. Use `setValue:forUndefinedKey:` (page 2065) instead.)
- `takeValue:forKey:` (page 2068) Deprecated in Mac OS X v10.3
  Sets the value for the property identified by *key* to *value*. (Deprecated. Use `setValue:forKey:` (page 2064) instead.)
- `takeValue:forKeyPath:` (page 2068) Deprecated in Mac OS X v10.3
  Sets the value for the property identified by *keyPath* to *value*. (Deprecated. Use `setValue:forKeyPath:` (page 2065) instead.)
- `takeValuesFromDictionary:` (page 2068) Deprecated in Mac OS X v10.3
  Sets properties of the receiver with values from a given dictionary, using its keys to identify the properties (Deprecated. Use `setValuesForKeysWithDictionary:` (page 2066) instead.)
- `unableToSetNilForKey:` (page 2068) Deprecated in Mac OS X v10.3
  Invoked if *key* is represented by a scalar attribute. (Deprecated. Use `setNilValueForKey:` (page 2064) instead.)
- `valuesForKeys:` (page 2072) Deprecated in Mac OS X v10.3
  Returns a dictionary containing as keys the property names in *keys*, with corresponding values being the corresponding property values. (Deprecated. Use `dictionaryWithValuesForKeys:` (page 2060) instead.)
- `+ useStoredAccessor` (page 2060) Deprecated in Mac OS X v10.4
  Returns `YES` if the stored value methods `storedValueForKey:` (page 2066) and `takeStoredValue:forKey:` (page 2067) should use private accessor methods in preference to public accessors. (Deprecated. This method has no direct replacement, although see accessInstanceVariablesDirectly (page 2059).)
- `storedValueForKey:` (page 2066) Deprecated in Mac OS X v10.4
  Returns the property identified by a given key. (Deprecated. If you are using the `NSManagedObject` class, use `primitiveValueForKey:` instead.)
- `takeStoredValue:forKey:` (page 2067) Deprecated in Mac OS X v10.4
  Sets the value of the property identified by a given key. (Deprecated. If you are using the `NSManagedObject` class, use `setPrimitiveValue:forKey:` instead.)

Class Methods

accessInstanceVariablesDirectly

Returns a Boolean value that indicates whether the key-value coding methods should access the corresponding instance variable directly on finding no accessor method for a property.

```
+ (BOOL)accessInstanceVariablesDirectly
```

Return Value

`YES` if the key-value coding methods should access the corresponding instance variable directly on finding no accessor method for a property, otherwise `NO`.

Discussion

The default returns `YES`. Subclasses can override it to return `NO`, in which case the key-value coding methods won't access instance variables.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSKeyValueCoding.h

useStoredAccessor

Returns YES if the stored value methods storedValueForKey: (page 2066) and takeStoredValue:forKey: (page 2067) should use private accessor methods in preference to public accessors. (Deprecated in Mac OS X v10.4. This method has no direct replacement, although see accessInstanceVariablesDirectly (page 2059).)

```
+ (BOOL)useStoredAccessor
```

Discussion
Returning NO causes the stored value methods to use the same accessor method or instance variable search order as the corresponding basic key-value coding methods (valueForKey: (page 2070) and takeValue:forKey: (page 2068)). The default implementation returns YES.

Applications should use the valueForKey: and setValue:forKey: methods instead of storedValueForKey: and takeStoredValue:forKey:.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
NSKeyValueCoding.h

Instance Methods

dictionaryWithValuesForKeys:

Returns a dictionary containing the property values identified by each of the keys in a given array.

```
- (NSDictionary *)dictionaryWithValuesForKeys:(NSArray *)keys
```

Parameters
keys
An array containing NSString objects that identify properties of the receiver.

Return Value
A dictionary containing as keys the property names in *keys*, with corresponding values being the corresponding property values.

Discussion
The default implementation invokes valueForKey: (page 2070) for each key in *keys* and substitutes NSNull values in the dictionary for returned nil values.

Availability
Available in Mac OS X v10.3 and later.

See Also
- `setValuesForKeysWithDictionary:` (page 2066)

Declared In
`NSKeyValueCoding.h`

handleQueryWithUnboundKey:

Invoked by `valueForKey:` (page 2070) when it finds no property corresponding to *key*. (Deprecated in Mac OS X v10.3. Use `valueForUndefinedKey:` (page 2071) instead.)

```
- (id)handleQueryWithUnboundKey:(NSString *)key
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.

Declared In
`NSKeyValueCoding.h`

handleTakeValue:forUnboundKey:

Invoked by `takeValue:forKey:` (page 2068) when it finds no property binding for *key*. (Deprecated in Mac OS X v10.3. Use `setValue:forUndefinedKey:` (page 2065) instead.)

```
- (void)handleTakeValue:(id)value forUnboundKey:(NSString *)key
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.

Declared In
`NSKeyValueCoding.h`

mutableArrayValueForKey:

Returns a mutable array proxy that provides read-write access to an ordered to-many relationship specified by a given key.

```
- (NSMutableArray *)mutableArrayValueForKey:(NSString *)key
```

Parameters
key
The name of an ordered to-many relationship.

Return Value
A mutable array proxy that provides read-write access to the ordered to-many relationship specified by *key*.

Discussion
Objects added to the mutable array become related to the receiver, and objects removed from the mutable array become unrelated. The default implementation recognizes the same simple accessor methods and array accessor methods as `valueForKey:` (page 2070), and follows the same direct instance variable access policies, but always returns a mutable collection proxy object instead of the immutable collection that `valueForKey:` would return.

The search pattern that `mutableArrayValueForKey:` uses is described in Accessor Search Implementation Details in *Key-Value Coding Programming Guide*.

Availability
Available in Mac OS X v10.3 and later.

See Also
`- mutableArrayValueForKeyPath:` (page 2062)

Declared In
`NSKeyValueCoding.h`

mutableArrayValueForKeyPath:

Returns a mutable array that provides read-write access to the ordered to-many relationship specified by a given key path.

```
- (NSMutableArray *)mutableArrayValueForKeyPath:(NSString *)keyPath
```

Parameters
keyPath
A key path, relative to the receiver, to an ordered to-many relationship.

Return Value
A mutable array that provides read-write access to the ordered to-many relationship specified by *keyPath*.

Discussion
See `mutableArrayValueForKey:` (page 2061) for additional details.

Availability
Available in Mac OS X v10.3 and later.

See Also
`- mutableArrayValueForKey:` (page 2061)

Declared In
`NSKeyValueCoding.h`

mutableSetValueForKey:

Returns a mutable set proxy that provides read-write access to the unordered to-many relationship specified by a given key.

```
- (NSMutableSet *)mutableSetValueForKey:(NSString *)key
```

Parameters

key

The name of an unordered to-many relationship.

Return Value

A mutable set that provides read-write access to the unordered to-many relationship specified by *key*.

Discussion

Objects added to the mutable set proxy become related to the receiver, and objects removed from the mutable set become unrelated. The default implementation recognizes the same simple accessor methods and set accessor methods as `valueForKey:` (page 2070), and follows the same direct instance variable access policies, but always returns a mutable collection proxy object instead of the immutable collection that `valueForKey:` would return.

The search pattern that `mutableSetValueForKey:` uses is described in Accessor Search Implementation Details in *Key-Value Coding Programming Guide*.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- mutableArrayValueForKeyPath:` (page 2062)

Related Sample Code

CoreRecipes

QTMetadataEditor

Declared In

`NSKeyValueCoding.h`

mutableSetValueForKeyPath:

Returns a mutable set that provides read-write access to the unordered to-many relationship specified by a given key path.

```
- (NSMutableSet *)mutableSetValueForKeyPath:(NSString *)keyPath
```

Parameters

keyPath

A key path, relative to the receiver, to an unordered to-many relationship.

Return Value

A mutable set that provides read-write access to the unordered to-many relationship specified by *keyPath*.

Discussion

See `mutableSetValueForKey:` (page 2062) for additional details.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- mutableArrayValueForKey:` (page 2061)

Declared In

`NSKeyValueCoding.h`

setNilValueForKey:

Invoked by `setValue:forKey:` (page 2064) when it's given a `nil` value for a scalar value (such as an `int` or `float`).

```
- (void)setNilValueForKey:(NSString *)key
```

Parameters

key

The name of one of the receiver's properties.

Discussion

Subclasses can override this method to handle the request in some other way, such as by substituting `0` or a sentinel value for `nil` and invoking `setValue:forKey:` again or setting the variable directly. The default implementation raises an `NSInvalidArgumentException`.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`NSKeyValueCoding.h`

setValue:forKey:

Sets the property of the receiver specified by a given key to a given value.

```
- (void)setValue:(id)value forKey:(NSString *)key
```

Parameters

value

The value for the property identified by *key*.

key

The name of one of the receiver's properties.

Discussion

If *key* identifies a to-one relationship, relate the object specified by *value* to the receiver, unrelating the previously related object if there was one. Given a collection object and a *key* that identifies a to-many relationship, relate the objects contained in the collection to the receiver, unrelating previously related objects if there were any.

The search pattern that `setValue:forKey:` uses is described in Accessor Search Implementation Details in *Key-Value Coding Programming Guide*.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CarbonCocoaCoreImageTab

CIAnnotation

CITransitionSelectorSample2

Reducer

StickiesExample

Declared In
NSKeyValueCoding.h

setValue:forKeyPath:

Sets the value for the property identified by a given key path to a given value.

```
- (void)setValue:(id)value forKeyPath:(NSString *)keyPath
```

Parameters

value

The value for the property identified by *keyPath*.

keyPath

A key path of the form *relationship.property* (with one or more relationships): for example "department.name" or "department.manager.lastName."

Discussion
The default implementation of this method gets the destination object for each relationship using `valueForKey:` (page 2070), and sends the final object a `setValue:forKey:` message.

Availability
Available in Mac OS X v10.3 and later.

See Also
`- valueForKeyPath:` (page 2071)

Declared In
NSKeyValueCoding.h

setValue:forUndefinedKey:

Invoked by `setValue:forKey:` (page 2064) when it finds no property for a given key.

```
- (void)setValue:(id)value forUndefinedKey:(NSString *)key
```

Parameters

value

The value for the key identified by *key*.

key

A string that is not equal to the name of any of the receiver's properties.

Discussion
Subclasses can override this method to handle the request in some other way. The default implementation raises an `NSUndefinedKeyException`.

Availability
Available in Mac OS X v10.3 and later.

See Also
`- valueForUndefinedKey:` (page 2071)

Declared In
NSKeyValueCoding.h

setValuesForKeysWithDictionary:

Sets properties of the receiver with values from a given dictionary, using its keys to identify the properties.

```
- (void)setValuesForKeysWithDictionary:(NSDictionary *)keyedValues
```

Parameters

keyedValues

A dictionary whose keys identify properties in the receiver. The values of the properties in the receiver are set to the corresponding values in the dictionary.

Discussion

The default implementation invokes `setValue:forKey:` (page 2064) for each key-value pair, substituting `nil` for `NSNull` values in *keyedValues*.

Availability

Available in Mac OS X v10.3 and later.

See Also

`- dictionaryWithValuesForKeys:` (page 2060)

Related Sample Code

Core Data HTML Store

Departments and Employees

Declared In

`NSKeyValueCoding.h`

storedValueForKey:

Returns the property identified by a given key. (Deprecated in Mac OS X v10.4. If you are using the `NSManagedObject` class, use `primitiveValueForKey:` instead.)

```
- (id)storedValueForKey:(NSString *)key
```

Discussion

This method is used when the value is retrieved for storage in an object store (generally, this storage is ultimately in a database) or for inclusion in a snapshot. The default implementation is similar to the implementation of `valueForKey:` (page 2070), but it resolves *key* with a different method/instance variable search order:

1. Searches for a private accessor method based on *key* (a method preceded by an underbar). For example, with a *key* of "lastName", `storedValueForKey:` looks for a method named `_getLastName` or `_lastName`.

2. If a private accessor is not found, searches for an instance variable based on *key* and returns its value directly. For example, with a `key` of "lastName", `storedValueForKey:` looks for an instance variable named `_lastName` or `lastName`.

3. If neither a private accessor nor an instance variable is found, `storedValueForKey:` searches for a public accessor method based on *key*. For the *key* "lastName", this would be `getLastName` or `lastName`.

4. If *key* is unknown, `storedValueForKey:` calls `handleTakeValue:forUnboundKey:` (page 2061).

This different search order allows an object to bypass processing that is performed before returning a value through a public API. However, if you always want to use the search order in `valueForKey:` (page 2070), you can implement the class method `useStoredAccessor` (page 2060) to return `NO`. And as with `valueForKey:` (page 2070), you can prevent direct access of an instance variable with the class method `accessInstanceVariablesDirectly` (page 2059).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`NSKeyValueCoding.h`

takeStoredValue:forKey:

Sets the value of the property identified by a given key. (Deprecated in Mac OS X v10.4. If you are using the `NSManagedObject` class, use `setPrimitiveValue:forKey:` instead.)

```
- (void)takeStoredValue:(id)value forKey:(NSString *)key
```

Discussion
This method is used to initialize the receiver with values from an object store (generally, this storage is ultimately from a database) or to restore a value from a snapshot. The default implementation is similar to the implementation of `takeValue:forKey:` (page 2068), but it resolves *key* with a different method/instance variable search order:

1. Searches for a private accessor method based on *key* (a method preceded by an underbar). For example, with a *key* of "lastName", `takeStoredValue:forKey:` looks for a method named `_setLastName:`.
2. If a private accessor is not found, searches for an instance variable based on *key* and sets its *value* directly. For example, with a *key* of "lastName", `takeStoredValue:forKey:` looks for an instance variable named `_lastName` or `lastName`.
3. If neither a private accessor nor an instance variable is found, `takeStoredValue:forKey:` searches for a public accessor method based on *key*. For the *key* "lastName", this would be `setLastName:`.
4. If *key* is unknown, `takeStoredValue:forKey:` calls `handleTakeValue:forUnboundKey:` (page 2061).

This different search order allows an object to bypass processing that is performed before setting a value through a public API. However, if you always want to use the search order in `takeValue:forKey:` (page 2068), you can implement the class method `useStoredAccessor` (page 2060) to return `NO`. And as with `valueForKey:` (page 2070), you can prevent direct access of an instance variable with the class method `accessInstanceVariablesDirectly` (page 2059).

Availability
Deprecated in Mac OS X v10.4.

Related Sample Code
StickiesExample

Declared In
`NSKeyValueCoding.h`

takeValue:forKey:

Sets the value for the property identified by *key* to *value*. (Deprecated in Mac OS X v10.3. Use `setValue:forKey:` (page 2064) instead.)

```
- (void)takeValue:(id)value forKey:(NSString *)key
```

Availability
Deprecated in Mac OS X v10.3.

Declared In
NSKeyValueCoding.h

takeValue:forKeyPath:

Sets the value for the property identified by *keyPath* to *value*. (Deprecated in Mac OS X v10.3. Use `setValue:forKeyPath:` (page 2065) instead.)

```
- (void)takeValue:(id)value forKeyPath:(NSString *)keyPath
```

Availability
Deprecated in Mac OS X v10.3.

Declared In
NSKeyValueCoding.h

takeValuesFromDictionary:

Sets properties of the receiver with values from a given dictionary, using its keys to identify the properties (Deprecated in Mac OS X v10.3. Use `setValuesForKeysWithDictionary:` (page 2066) instead.)

```
- (void)takeValuesFromDictionary:(NSDictionary *)aDictionary
```

Availability
Deprecated in Mac OS X v10.3.

Declared In
NSKeyValueCoding.h

unableToSetNilForKey:

Invoked if *key* is represented by a scalar attribute. (Deprecated in Mac OS X v10.3. Use `setNilValueForKey:` (page 2064) instead.)

```
- (void)unableToSetNilForKey:(NSString *)key
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.

Declared In
NSKeyValueCoding.h

validateValue:forKey:error:

Returns a Boolean value that indicates whether the value specified by a given pointer is valid for the property identified by a given key.

```
- (BOOL)validateValue:(id *)ioValue forKey:(NSString *)key error:(NSError **)outError
```

Parameters

ioValue

A pointer to a new value for the property identified by *key*. This method may modify or replace the value in order to make it valid.

key

The name of one of the receiver's properties. The key must specify an attribute or a to-one relationship.

outError

If validation is necessary and *ioValue* is not transformed into a valid value, upon return contains an `NSError` object that describes the reason that *ioValue* is not a valid value.

Return Value

YES if **ioValue* is a valid value for the property identified by *key*, or of the method is able to modify the value to **ioValue* to make it valid; otherwise `NO`.

Discussion

The default implementation of this method searches the class of the receiver for a validation method whose name matches the pattern `validate<Key>:error:`. If such a method is found it is invoked and the result is returned. If no such method is found, `YES` is returned.

The sender of the message is never given responsibility for releasing *ioValue* or *outError*.

See "Key-Value Validation" for more information.

Availability

Available in Mac OS X v10.3 and later.

See Also

- validateValue:forKeyPath:error: (page 2069)

Declared In

`NSKeyValueCoding.h`

validateValue:forKeyPath:error:

Returns a Boolean value that indicates whether the value specified by a given pointer is valid for a given key path relative to the receiver.

```
- (BOOL)validateValue:(id *)ioValue forKeyPath:(NSString *)inKeyPath error:(NSError
    **)outError
```

Parameters

ioValue

A pointer to a new value for the property identified by *keyPath*. This method may modify or replace the value in order to make it valid.

key

The name of one of the receiver's properties. The key path must specify an attribute or a to-one relationship. The key path has the form *relationship.property* (with one or more relationships); for example “department.name” or “department.manager.lastName”.

outError

If validation is necessary and *ioValue* is not transformed into a valid value, upon return contains an `NSError` object that describes the reason that *ioValue* is not a valid value.

Discussion
The default implementation gets the destination object for each relationship using `valueForKey:` (page 2070) and returns the result of a `validateValue:forKey:error:` message to the final object.

Availability
Available in Mac OS X v10.3 and later.

See Also
`- validateValue:forKey:error:` (page 2069)

Declared In
`NSKeyValueCoding.h`

valueForKey:

Returns the value for the property identified by a given key.

```
- (id)valueForKey:(NSString *)key
```

Parameters

key

The name of one of the receiver's properties.

Return Value
The value for the property identified by *key*.

Discussion
The search pattern that `valueForKey:` uses to find the correct value to return is described in Accessor Search Implementation Details in *Key-Value Coding Programming Guide*.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- valueForKeyPath:` (page 2071)

Related Sample Code
CIAnnotation
CITransitionSelectorSample2
CustomAtomicStoreSubclass
DynamicProperties
QTCarbonCoreImage101

Declared In
`NSKeyValueCoding.h`

valueForKeyPath:

Returns the value for the derived property identified by a given key path.

```
- (id)valueForKeyPath:(NSString *)keyPath
```

Parameters

keyPath

A key path of the form *relationship.property* (with one or more relationships); for example "department.name" or "department.manager.lastName".

Return Value

The value for the derived property identified by *keyPath*.

Discussion

The default implementation gets the destination object for each relationship using `valueForKey:` (page 2070) and returns the result of a `valueForKey:` message to the final object.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- setValue:forKeyPath:` (page 2065)

Related Sample Code

Aperture Edit Plugin - Borders & Titles
Core Data HTML Store
CoreRecipes
Dicey
Spotlight

Declared In

`NSKeyValueCoding.h`

valueForUndefinedKey:

Invoked by `valueForKey:` (page 2070) when it finds no property corresponding to a given key.

```
- (id)valueForUndefinedKey:(NSString *)key
```

Parameters

key

A string that is not equal to the name of any of the receiver's properties.

Discussion

Subclasses can override this method to return an alternate value for undefined keys. The default implementation raises an `NSUndefinedKeyException`.

Availability

Available in Mac OS X v10.3 and later.

See Also

`- setValue:forUndefinedKey:` (page 2065)

Declared In
NSKeyValueCoding.h

valuesForKeys:

Returns a dictionary containing as keys the property names in *keys*, with corresponding values being the corresponding property values. (Deprecated in Mac OS X v10.3. Use `dictionaryWithValuesForKeys:` (page 2060) instead.)

```
- (NSDictionary *)valuesForKeys:(NSArray *)keys
```

Availability
Deprecated in Mac OS X v10.3.

Related Sample Code
Core Data HTML Store
Departments and Employees

Declared In
NSKeyValueCoding.h

Constants

Key Value Coding Exception Names

This constant defines the name of an exception raised when a key value coding operation fails.

```
extern NSString *NSUndefinedKeyException;
```

Constants
NSUndefinedKeyException
Raised when a key value coding operation fails. *userInfo* keys are described in "NSUndefinedKeyException userInfo Keys" (page 2072)

Available in Mac OS X v10.3 and later.

Declared in NSKeyValueCoding.h.

Declared In
NSKeyValueCoding.h

NSUndefinedKeyException userInfo Keys

These constants are keys into an NSUndefinedKeyException *userInfo* dictionary

```
extern NSString *NSTargetObjectUserInfoKey;
extern NSString *NSUnknownUserInfoKey;
```

Constants
NSTargetObjectUserInfoKey
The object on which the key value coding operation failed.

`NSUnknownUserInfoKey`
The key for which the key value coding operation failed.

Discussion
For additional information see "Key Value Coding Exception Names" (page 2072).

Declared In
`NSKeyValueCoding.h`

Array operators

These constants define the available array operators. See Set and Array Operators for more information.

```
NSString *const NSAverageKeyValueOperator;
NSString *const NSCountKeyValueOperator;
NSString *const NSDistinctUnionOfArraysKeyValueOperator;
NSString *const NSDistinctUnionOfObjectsKeyValueOperator;
NSString *const NSDistinctUnionOfSetsKeyValueOperator;
NSString *const NSMaximumKeyValueOperator;
NSString *const NSMinimumKeyValueOperator;
NSString *const NSSumKeyValueOperator;
NSString *const NSUnionOfArraysKeyValueOperator;
NSString *const NSUnionOfObjectsKeyValueOperator;
NSString *const NSUnionOfSetsKeyValueOperator;
```

Constants

`NSAverageKeyValueOperator`
The `@avg` array operator.

Available in Mac OS X v10.4 and later.

Declared in `NSKeyValueCoding.h`.

`NSCountKeyValueOperator`
The `@count` array operator.

Available in Mac OS X v10.4 and later.

Declared in `NSKeyValueCoding.h`.

`NSDistinctUnionOfArraysKeyValueOperator`
The `@distinctUnionOfArrays` array operator.

Available in Mac OS X v10.4 and later.

Declared in `NSKeyValueCoding.h`.

`NSDistinctUnionOfObjectsKeyValueOperator`
The `@distinctUnionOfObjects` array operator.

Available in Mac OS X v10.4 and later.

Declared in `NSKeyValueCoding.h`.

`NSDistinctUnionOfSetsKeyValueOperator`
The `@distinctUnionOfSets` array operator.

Available in Mac OS X v10.4 and later.

Declared in `NSKeyValueCoding.h`.

`NSMaximumKeyValueOperator`
The `@max` array operator.

Available in Mac OS X v10.4 and later.

Declared in `NSKeyValueCoding.h`.

`NSMinimumKeyValueOperator`
The `@min` array operator.

Available in Mac OS X v10.4 and later.

Declared in `NSKeyValueCoding.h`.

`NSSumKeyValueOperator`
The `@sum` array operator.

Available in Mac OS X v10.4 and later.

Declared in `NSKeyValueCoding.h`.

`NSUnionOfArraysKeyValueOperator`
The `@unionOfArrays` array operator.

Available in Mac OS X v10.4 and later.

Declared in `NSKeyValueCoding.h`.

`NSUnionOfObjectsKeyValueOperator`
The `@unionOfObjects` array operator.

Available in Mac OS X v10.4 and later.

Declared in `NSKeyValueCoding.h`.

`NSUnionOfSetsKeyValueOperator`
The `@unionOfSets` array operator.

Available in Mac OS X v10.4 and later.

Declared in `NSKeyValueCoding.h`.

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`NSKeyValueCoding.h`

NSKeyValueObserving Protocol Reference

(informal protocol)

Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSKeyValueObserving.h
Companion guide	Key-Value Observing Programming Guide

Overview

The `NSKeyValueObserving` (KVO) informal protocol defines a mechanism that allows objects to be notified of changes to the specified properties of other objects.

You can observe any object properties including simple attributes, to-one relationships, and to-many relationships. Observers of to-many relationships are informed of the type of change made — as well as which objects are involved in the change.

`NSObject` provides an implementation of the `NSKeyValueObserving` protocol that provides an automatic observing capability for all objects. You can further refine notifications by disabling automatic observer notifications and implementing manual notifications using the methods in this protocol.

Note: Key-value observing is not available for Java applications.

Tasks

Change Notification

- `observeValueForKeyPath:ofObject:change:context:` (page 2081)
  This message is sent to the receiver when the value at the specified key path relative to the given object has changed.

Registering for Observation

- `addObserver:forKeyPath:options:context:` (page 2079)
  Registers *anObserver* to receive KVO notifications for the specified key-path relative to the receiver.
- `removeObserver:forKeyPath:` (page 2082)
  Stops a given object from receiving change notifications for the property specified by a given key-path relative to the receiver.

Notifying Observers of Changes

`- willChangeValueForKey:` (page 2084)
Invoked to inform the receiver that the value of a given property is about to change.

`- didChangeValueForKey:` (page 2080)
Invoked to inform the receiver that the value of a given property has changed.

`- willChange:valuesAtIndexes:forKey:` (page 2083)
Invoked to inform the receiver that the specified change is about to be executed at given indexes for a specified ordered to-many relationship.

`- didChange:valuesAtIndexes:forKey:` (page 2079)
Invoked to inform the receiver that the specified change has occurred on the indexes for a specified ordered to-many relationship.

`- willChangeValueForKey:withSetMutation:usingObjects:` (page 2084)
Invoked to inform the receiver that the specified change is about to be made to a specified unordered to-many relationship.

`- didChangeValueForKey:withSetMutation:usingObjects:` (page 2080)
Invoked to inform the receiver that the specified change was made to a specified unordered to-many relationship.

Observing Customization

`+ automaticallyNotifiesObserversForKey:` (page 2076)
Returns a Boolean value that indicates whether the receiver supports automatic key-value observation for the given key.

`+ keyPathsForValuesAffectingValueForKey:` (page 2077)
Returns a set of key paths for properties whose values affect the value of the specified key.

`- setObservationInfo:` (page 2083)
Sets the observation info for the receiver.

`- observationInfo` (page 2081)
Returns a pointer that identifies information about all of the observers that are registered with the receiver.

`+ setKeys:triggerChangeNotificationsForDependentKey:` (page 2078) Deprecated in Mac OS X v10.5 and later
Configures the receiver to post change notifications for a given property if any of the properties specified in a given array changes. (Deprecated. You should use the method `keyPathsForValuesAffectingValueForKey:` (page 2077) instead.)

Class Methods

automaticallyNotifiesObserversForKey:

Returns a Boolean value that indicates whether the receiver supports automatic key-value observation for the given key.

```
+ (BOOL)automaticallyNotifiesObserversForKey:(NSString *)key
```

Return Value
`YES` if the key-value observing machinery should automatically invoke `willChangeValueForKey:` (page 2084)/`didChangeValueForKey:` (page 2080) and `willChange:valuesAtIndexes:forKey:` (page 2083)/`didChange:valuesAtIndexes:forKey:` (page 2079) whenever instances of the class receive key-value coding messages for the *key*, or mutating key-value-coding-compliant methods for the *key* are invoked; otherwise `NO`.

Discussion
The default implementation returns `YES`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`NSKeyValueObserving.h`

keyPathsForValuesAffectingValueForKey:

Returns a set of key paths for properties whose values affect the value of the specified key.

```
+ (NSSet *)keyPathsForValuesAffectingValueForKey:(NSString *)key
```

Parameters

key

The key whose value is affected by the key paths.

Return Value

Discussion
When an observer for the key is registered with an instance of the receiving class, key-value observing itself automatically observes all of the key paths for the same instance, and sends change notifications for the key to the observer when the value for any of those key paths changes.

The default implementation of this method searches the receiving class for a method whose name matches the pattern `+keyPathsForValuesAffecting<Key>`, and returns the result of invoking that method if it is found. Any such method must return an NSSet. If no such method is found, an NSSet that is computed from information provided by previous invocations of the now-deprecated `setKeys:triggerChangeNotificationsForDependentKey:` (page 2078) method is returned, for backward binary compatibility.

You can override this method when the getter method of one of your properties computes a value to return using the values of other properties, including those that are located by key paths. Your override should typically invoke super and return a set that includes any members in the set that result from doing that (so as not to interfere with overrides of this method in superclasses).

Note: You must not override this method when you add a computed property to an existing class using a category, overriding methods in categories is unsupported. In that case, implement a matching `+keyPathsForValuesAffecting<Key>` to take advantage of this mechanism.

Availability
Available in Mac OS X v10.5 and later.

Declared In
NSKeyValueObserving.h

setKeys:triggerChangeNotificationsForDependentKey:

Configures the receiver to post change notifications for a given property if any of the properties specified in a given array changes. (Deprecated in Mac OS X v10.5 and later. You should use the method `keyPathsForValuesAffectingValueForKey:` (page 2077) instead.)

```
+ (void)setKeys:(NSArray *)keys
    triggerChangeNotificationsForDependentKey:(NSString *)dependentKey
```

Parameters

keys

The names of the properties upon which the value of the property identified by *dependentKey* depends.

dependentKey

The name of a property whose value depends on the properties specified by *keys*.

Discussion
Invocations of will- and did-change KVO notification methods for any key in *keys* will automatically invoke the corresponding change notification methods for *dependentKey*. The receiver will not receive `willChange`/`didChange` messages to generate the notifications.

Dependencies should be registered before any instances of the receiving class are created, so you typically invoke this method in a class's `initialize` (page 1158) method, as illustrated in the following example.

```
+ (void)initialize
{
    [self setKeys:[NSArray arrayWithObjects:@"firstName", @"lastName", nil]
          triggerChangeNotificationsForDependentKey:@"fullName"];
}
```

Availability
Deprecated in Mac OS X v10.5 and later.

Related Sample Code
CoreRecipes
Dicey
iSpend
QTRecorder
Reducer

Declared In
NSKeyValueObserving.h

Instance Methods

addObserver:forKeyPath:options:context:

Registers *anObserver* to receive KVO notifications for the specified key-path relative to the receiver.

```
- (void)addObserver:(NSObject *)anObserver
    forKeyPath:(NSString *)keyPath
    options:(NSKeyValueObservingOptions)options
    context:(void *)context
```

Parameters

anObserver

The object to register for KVO notifications. The observer must implement the key-value observing method `observeValueForKeyPath:ofObject:change:context:` (page 2081).

keyPath

The key path, relative to the receiver, of the property to observe. This value must not be `nil`.

options

A combination of the `NSKeyValueObservingOptions` values that specifies what is included in observation notifications. For possible values, see NSKeyValueObservingOptions (page 2086).

context

Arbitrary data that is passed to *anObserver* in observeValueForKeyPath:ofObject:change:context: (page 2081).

Discussion

Neither the receiver, nor *anObserver*, are retained.

Availability

Available in Mac OS X v10.3 and later.

See Also

`- removeObserver:forKeyPath:` (page 2082)

Declared In

`NSKeyValueObserving.h`

didChange:valuesAtIndexes:forKey:

Invoked to inform the receiver that the specified change has occurred on the indexes for a specified ordered to-many relationship.

```
- (void)didChange:(NSKeyValueChange)change
    valuesAtIndexes:(NSIndexSet *)indexes
    forKey:(NSString *)key
```

Parameters

change

The type of change that was made.

indexes

The indexes of the to-many relationship that were affected by the change.

key
The name of a property that is an ordered to-many relationship.

Discussion
You should invoke this method when implementing key-value-observing compliance manually.

Availability
Available in Mac OS X v10.3 and later.

See Also
- `willChange:valuesAtIndexes:forKey:` (page 2083)
- `didChangeValueForKey:` (page 2080)

Declared In
`NSKeyValueObserving.h`

didChangeValueForKey:

Invoked to inform the receiver that the value of a given property has changed.

```
- (void)didChangeValueForKey:(NSString *)key
```

Parameters
key
The name of the property that changed.

Discussion
You should invoke this method when implementing key-value observer compliance manually.

Availability
Available in Mac OS X v10.3 and later.

See Also
- `willChangeValueForKey:` (page 2084)
- `didChange:valuesAtIndexes:forKey:` (page 2079)

Declared In
`NSKeyValueObserving.h`

didChangeValueForKey:withSetMutation:usingObjects:

Invoked to inform the receiver that the specified change was made to a specified unordered to-many relationship.

```
- (void)didChangeValueForKey:(NSString *)key
    withSetMutation:(NSKeyValueSetMutationKind)mutationKind
    usingObjects:(NSSet *)objects
```

Parameters
key
The name of a property that is an unordered to-many relationship

mutationKind
The type of change that was made.

objects
The objects that were involved in the change (see NSKeyValueSetMutationKind (page 2088)).

Discussion
You invoke this method when implementing key-value observer compliance manually.

Availability
Available in Mac OS X v10.4 and later.

See Also
- willChangeValueForKey:withSetMutation:usingObjects: (page 2084)

Declared In
NSKeyValueObserving.h

observationInfo

Returns a pointer that identifies information about all of the observers that are registered with the receiver.

```
- (void *)observationInfo
```

Return Value
A pointer that identifies information about all of the observers that are registered with the receiver, the options that were used at registration-time, and so on.

Discussion
The default implementation of this method retrieves the information from a global dictionary keyed by the receiver's pointers.

For improved performance, this method and `setObservationInfo:` can be overridden to store the opaque data pointer in an instance variable. Overrides of this method must not attempt to send Objective-C messages to the stored data, including `retain` and `release`.

Availability
Available in Mac OS X v10.3 and later.

See Also
- setObservationInfo: (page 2083)

Declared In
NSKeyValueObserving.h

observeValueForKeyPath:ofObject:change:context:

This message is sent to the receiver when the value at the specified key path relative to the given object has changed.

```
- (void)observeValueForKeyPath:(NSString *)keyPath
    ofObject:(id)object
    change:(NSDictionary *)change
    context:(void *)context
```

Parameters

keyPath

The key path, relative to *object*, to the value that has changed.

object

The source object of the key path *keyPath*.

change

A dictionary that describes the changes that have been made to the value of the property at the key path *keyPath* relative to *object*. Entries are described in "Keys used by the change dictionary" (page 2087).

context

The value that was provided when the receiver was registered to receive key-value observation notifications.

Discussion

The receiver must be registered as an observer for the specified *keyPath* and *object*.

Availability

Available in Mac OS X v10.3 and later.

Declared In

NSKeyValueObserving.h

removeObserver:forKeyPath:

Stops a given object from receiving change notifications for the property specified by a given key-path relative to the receiver.

```
- (void)removeObserver:(NSObject *)anObserver
    forKeyPath:(NSString *)keyPath
```

Parameters

anObserver

The object to remove as an observer.

keyPath

A key-path, relative to the receiver, for which *anObserver* is registered to receive KVO change notifications.

Availability

Available in Mac OS X v10.3 and later.

See Also

- addObserver:forKeyPath:options:context: (page 2079)

Related Sample Code

Departments and Employees

Declared In

NSKeyValueObserving.h

setObservationInfo:

Sets the observation info for the receiver.

```
- (void)setObservationInfo:(void *)observationInfo
```

Parameters

observationInfo

The observation info for the receiver.

Discussion

The *observationInfo* is a pointer that identifies information about all of the observers that are registered with the receiver. The default implementation of this method stores *observationInfo* in a global dictionary keyed by the receiver's pointers.

For improved performance, this method and `observationInfo` can be overridden to store the opaque data pointer in an instance variable. Classes that override this method must not attempt to send Objective-C messages to *observationInfo*, including `retain` and `release`.

Availability

Available in Mac OS X v10.3 and later.

See Also

`- observationInfo` (page 2081)

Declared In

`NSKeyValueObserving.h`

willChange:valuesAtIndexes:forKey:

Invoked to inform the receiver that the specified change is about to be executed at given indexes for a specified ordered to-many relationship.

```
- (void)willChange:(NSKeyValueChange)change
    valuesAtIndexes:(NSIndexSet *)indexes
    forKey:(NSString *)key
```

Parameters

change

The type of change that is about to be made.

indexes

The indexes of the to-many relationship that will be affected by the change.

key

The name of a property that is an ordered to-many relationship.

Discussion

You should invoke this method when implementing key-value-observing compliance manually.

Important: After the values have been changed, a corresponding `didChange:valuesAtIndexes:forKey:` (page 2079) must be invoked with the same parameters.

Availability

Available in Mac OS X v10.3 and later.

See Also
- `didChange:valuesAtIndexes:forKey:` (page 2079)
- `willChangeValueForKey:` (page 2084)

Declared In
`NSKeyValueObserving.h`

willChangeValueForKey:

Invoked to inform the receiver that the value of a given property is about to change.

```
- (void)willChangeValueForKey:(NSString *)key
```

Parameters

key

The name of the property that will change.

Discussion
You should invoke this method when implementing key-value observer compliance manually.

The change type of this method is `NSKeyValueChangeSetting`.

Important: After the values have been changed, a corresponding `didChangeValueForKey:` (page 2080) must be invoked with the same parameter.

Availability
Available in Mac OS X v10.3 and later.

See Also
- `didChangeValueForKey:` (page 2080)
- `willChange:valuesAtIndexes:forKey:` (page 2083)

Declared In
`NSKeyValueObserving.h`

willChangeValueForKey:withSetMutation:usingObjects:

Invoked to inform the receiver that the specified change is about to be made to a specified unordered to-many relationship.

```
- (void)willChangeValueForKey:(NSString *)key
    withSetMutation:(NSKeyValueSetMutationKind)mutationKind
    usingObjects:(NSSet *)objects
```

Parameters

key

The name of a property that is an unordered to-many relationship

mutationKind

The type of change that will be made.

objects
The objects that are involved in the change (see NSKeyValueSetMutationKind (page 2088)).

Discussion
You invoke this method when implementing key-value observer compliance manually.

Important: After the values have been changed, a corresponding `didChangeValueForKey:withSetMutation:usingObjects:` (page 2080) must be invoked with the same parameters.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- didChangeValueForKey:withSetMutation:usingObjects:` (page 2080)

Declared In
`NSKeyValueObserving.h`

Constants

NSKeyValueChange

These constants are returned as the value for a `NSKeyValueChangeKindKey` key in the change dictionary passed to `observeValueForKeyPath:ofObject:change:context:` (page 2081) indicating the type of change made:

```
enum {
    NSKeyValueChangeSetting = 1,
    NSKeyValueChangeInsertion = 2,
    NSKeyValueChangeRemoval = 3,
    NSKeyValueChangeReplacement = 4
};
typedef NSUInteger NSKeyValueChange;
```

Constants

`NSKeyValueChangeSetting`
Indicates that the value of the observed key path was set to a new value. This change can occur when observing an attribute of an object, as well as properties that specify to-one and to-many relationships.

Available in Mac OS X v10.3 and later.

Declared in `NSKeyValueObserving.h`.

`NSKeyValueChangeInsertion`
Indicates that an object has been inserted into the to-many relationship that is being observed.

Available in Mac OS X v10.3 and later.

Declared in `NSKeyValueObserving.h`.

`NSKeyValueChangeRemoval`
Indicates that an object has been removed from the to-many relationship that is being observed.

Available in Mac OS X v10.3 and later.

Declared in `NSKeyValueObserving.h`.

`NSKeyValueChangeReplacement`
Indicates that an object has been replaced in the to-many relationship that is being observed.

Available in Mac OS X v10.3 and later.

Declared in `NSKeyValueObserving.h`.

Declared In
`NSKeyValueObserving.h`

NSKeyValueObservingOptions

These constants are passed to `addObserver:forKeyPath:options:context:` (page 2079) and determine the values that are returned as part of the change dictionary passed to an `observeValueForKeyPath:ofObject:change:context:` (page 2081). You can pass `0` if you require no change dictionary values.

```
enum {
    NSKeyValueObservingOptionNew = 0x01,
    NSKeyValueObservingOptionOld = 0x02,
    NSKeyValueObservingOptionInitial = 0x04,
    NSKeyValueObservingOptionPrior = 0x08
};
typedef NSUInteger NSKeyValueObservingOptions;
```

Constants
`NSKeyValueObservingOptionNew`
Indicates that the change dictionary should provide the new attribute value, if applicable.

Available in Mac OS X v10.3 and later.

Declared in `NSKeyValueObserving.h`.

`NSKeyValueObservingOptionOld`
Indicates that the change dictionary should contain the old attribute value, if applicable.

Available in Mac OS X v10.3 and later.

Declared in `NSKeyValueObserving.h`.

`NSKeyValueObservingOptionInitial`
If specified, a notification should be sent to the observer immediately, before the observer registration method even returns. The change dictionary in the notification will always contain an `NSKeyValueChangeNewKey` entry if `NSKeyValueObservingOptionNew` is also specified but will never contain an `NSKeyValueChangeOldKey` entry. (In an initial notification the current value of the observed property may be old, but it's new to the observer.) You can use this option instead of explicitly invoking, at the same time, code that is also invoked by the observer's `observeValueForKeyPath:ofObject:change:context:` method. When this option is used with`addObserver:forKeyPath:options:context:` a notification will be sent for each indexed object to which the observer is being added.

Available in Mac OS X v10.5 and later.

Declared in `NSKeyValueObserving.h`.

`NSKeyValueObservingOptionPrior`

Whether separate notifications should be sent to the observer before and after each change, instead of a single notification after the change. The change dictionary in a notification sent before a change always contains an `NSKeyValueChangeNotificationIsPriorKey` entry whose value is `[NSNumber numberWithBool:YES]`, but never contains an `NSKeyValueChangeNewKey` entry. When this option is specified the change dictionary in a notification sent after a change contains the same entries that it would contain if this option were not specified. You can use this option when the observer's own key-value observing-compliance requires it to invoke one of the `-willChange...` methods for one of its own properties, and the value of that property depends on the value of the observed object's property. (In that situation it's too late to easily invoke `-willChange...` properly in response to receiving an `observeValueForKeyPath:ofObject:change:context:` message after the change.)

Available in Mac OS X v10.5 and later.

Declared in `NSKeyValueObserving.h`.

Declared In
`NSKeyValueObserving.h`

Keys used by the change dictionary

These constants are used as keys in the change dictionary passed to `observeValueForKeyPath:ofObject:change:context:` (page 2081).

```
NSString *const NSKeyValueChangeKindKey;
NSString *const NSKeyValueChangeNewKey;
NSString *const NSKeyValueChangeOldKey;
NSString *const NSKeyValueChangeIndexesKey;
```

Constants

`NSKeyValueChangeKindKey`

An `NSNumber` object that contains a value corresponding to one of the `NSKeyValueChangeKindKey` enumerations, indicating what sort of change has occurred.

A value of `NSKeyValueChangeSetting` indicates that the observed object has received a `setValue:forKey:` message, or that the key-value-coding-compliant set method for the key has been invoked, or that `willChangeValueForKey:` (page 2084)/`didChangeValueForKey:` (page 2080) has otherwise been invoked.

A value of `NSKeyValueChangeInsertion`, `NSKeyValueChangeRemoval`, or `NSKeyValueChangeReplacement` indicates that mutating messages have been sent to the array returned by a `mutableArrayValueForKey:` message sent to the object, or that one of the key-value-coding-compliant array mutation methods for the key has been invoked, or that `willChange:valuesAtIndexes:forKey:` (page 2083)/`didChange:valuesAtIndexes:forKey:` (page 2079) has otherwise been invoked.

You can use `NSNumber`'s `intValue` (page 1073) method to retrieve the integer value of the change kind.

Available in Mac OS X v10.3 and later.

Declared in `NSKeyValueObserving.h`.

`NSKeyValueChangeNewKey`

If the value of the `NSKeyValueChangeKindKey` entry is `NSKeyValueChangeSetting`, and `NSKeyValueObservingOptionNew` was specified when the observer was registered, the value of this key is the new value for the attribute.

For `NSKeyValueChangeInsertion` or `NSKeyValueChangeReplacement`, if `NSKeyValueObservingOptionNew` was specified when the observer was registered, the value for this key is an `NSArray` instance that contains the objects that have been inserted or replaced other objects, respectively.

Available in Mac OS X v10.3 and later.

Declared in `NSKeyValueObserving.h`.

`NSKeyValueChangeOldKey`

If the value of the `NSKeyValueChangeKindKey` entry is `NSKeyValueChangeSetting`, and `NSKeyValueObservingOptionOld` was specified when the observer was registered, the value of this key is the value before the attribute was changed.

For `NSKeyValueChangeRemoval` or `NSKeyValueChangeReplacement`, if `NSKeyValueObservingOptionOld` was specified when the observer was registered, the value is an `NSArray` instance that contains the objects that have been removed or have been replaced by other objects, respectively.

Available in Mac OS X v10.3 and later.

Declared in `NSKeyValueObserving.h`.

`NSKeyValueChangeIndexesKey`

If the value of the `NSKeyValueChangeKindKey` entry is `NSKeyValueChangeInsertion`, `NSKeyValueChangeRemoval`, or `NSKeyValueChangeReplacement`, the value of this key is an `NSIndexSet` object that contains the indexes of the inserted, removed, or replaced objects.

Available in Mac OS X v10.3 and later.

Declared in `NSKeyValueObserving.h`.

Declared In
`NSKeyValueObserving.h`

NSKeyValueSetMutationKind

These constants are specified as the parameter to the methods `willChangeValueForKey:withSetMutation:usingObjects:` (page 2084) and `didChangeValueForKey:withSetMutation:usingObjects:` (page 2080).

```
enum {
    NSKeyValueUnionSetMutation = 1,
    NSKeyValueMinusSetMutation = 2,
    NSKeyValueIntersectSetMutation = 3,
    NSKeyValueSetSetMutation = 4
```

```
};
typedef NSUInteger NSKeyValueSetMutationKind;
```

Constants

`NSKeyValueUnionSetMutation`

Indicates that objects in the specified set are being added to the receiver. This mutation kind results in a `NSkeyValueChangeKindKey` value of `NSKeyValueChangeInsertion`.

Available in Mac OS X v10.4 and later.

Declared in `NSKeyValueObserving.h`.

`NSKeyValueMinusSetMutation`

Indicates that the objects in the specified set are being removed from the receiver. This mutation kind results in a `NSkeyValueChangeKindKey` value of `NSKeyValueChangeRemoval`.

Available in Mac OS X v10.4 and later.

Declared in `NSKeyValueObserving.h`.

`NSKeyValueIntersectSetMutation`

Indicates that the objects not in the specified set are being removed from the receiver. This mutation kind results in a `NSkeyValueChangeKindKey` value of `NSKeyValueChangeRemoval`.

Available in Mac OS X v10.4 and later.

Declared in `NSKeyValueObserving.h`.

`NSKeyValueSetSetMutation`

Indicates that set of objects are replacing the existing objects in the receiver. This mutation kind results in a `NSkeyValueChangeKindKey` value of `NSKeyValueChangeReplacement`.

Available in Mac OS X v10.4 and later.

Declared in `NSKeyValueObserving.h`.

Declared In

`NSKeyValueObserving.h`

NSLocking Protocol Reference

Adopted by	NSConditionLock NSLock NSRecursiveLock
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSLock.h
Companion guide	Threading Programming Guide

Overview

The `NSLocking` protocol declares the elementary methods adopted by classes that define lock objects. A lock object is used to coordinate the actions of multiple threads of execution within a single application. By using a lock object, an application can protect critical sections of code from being executed simultaneously by separate threads, thus protecting shared data and other shared resources from corruption.

Tasks

Working with Locks

- `lock` (page 2091)
  Attempts to acquire a lock, blocking a thread's execution until the lock can be acquired.
- `unlock` (page 2092)
  Relinquishes a previously acquired lock.

Instance Methods

lock

Attempts to acquire a lock, blocking a thread's execution until the lock can be acquired.

```
- (void)lock
```

Discussion
An application protects a critical section of code by requiring a thread to acquire a lock before executing the code. Once the critical section is past, the thread relinquishes the lock by invoking `unlock` (page 2092).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
SimpleThreads

Declared In
`NSLock.h`

unlock

Relinquishes a previously acquired lock.

```
- (void)unlock
```

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
SimpleThreads

Declared In
`NSLock.h`

NSMutableCopying Protocol Reference

Adopted by	Various Cocoa classes
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSObject.h
Companion guide	Memory Management Programming Guide for Cocoa

Overview

The `NSMutableCopying` protocol declares a method for providing mutable copies of an object. Only classes that define an "immutable vs. mutable" distinction should adopt this protocol. Classes that don't define such a distinction should adopt `NSCopying` instead.

`NSMutableCopying` declares one method, `mutableCopyWithZone:` (page 2094), but mutable copying is commonly invoked with the convenience method `mutableCopy`. The `mutableCopy` method is defined for all NSObjects and simply invokes `mutableCopyWithZone:` (page 2094) with the default zone.

If a subclass inherits NSMutableCopying from its superclass and declares additional instance variables, the subclass has to override `mutableCopyWithZone:` (page 2094) to properly handle its own instance variables, invoking the superclass's implementation first.

Tasks

Copying

`- mutableCopyWithZone:` (page 2094)
Returns a new instance that's a mutable copy of the receiver.

Instance Methods

mutableCopyWithZone:

Returns a new instance that's a mutable copy of the receiver.

```
- (id)mutableCopyWithZone:(NSZone *)zone
```

Parameters

zone

The zone from which memory is allocated for the new instance. If *zone* is `NULL`, the new instance is allocated from the default zone, which is returned by `NSDefaultMallocZone` (page 2188).

Discussion

The returned object is implicitly retained by the sender, which is responsible for releasing it. The copy returned is mutable whether the original is mutable or not.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `copyWithZone:` (page 2042) (NSCopying protocol)
- `mutableCopy` (page 1182) (NSObject class)

Declared In

`NSObject.h`

NSObjCTypeSerializationCallBack Protocol Reference

Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSSerialization.h
Availability	Deprecated in Mac OS X v10.2.
Companion guide	Archives and Serializations Programming Guide for Cocoa

Overview

`NSObjCTypeSerializationCallBack` is obsolete and had been deprecated.

An object conforms to the `NSObjCTypeSerializationCallBack` protocol so that it can intervene in the serialization and deserialization process. The primary purpose of this protocol is to allow for the serialization of objects and other data types that are not directly supported by Cocoa's serialization facility.

Tasks

Serializing

`- serializeObjectAt:ofObjCType:intoData:` (page 2096) Deprecated in Mac OS X v10.2
Encodes the referenced object *object* (which should always be of type "@") in the *data* object.

Deserializing

`- deserializeObjectAt:ofObjCType:fromData:atCursor:` (page 2096) Deprecated in Mac OS X v10.2
Decodes the referenced object *object* (which should always be of type "@") located at the *cursor* position in the *data* object.

Instance Methods

deserializeObjectAt:ofObjCType:fromData:atCursor:

Decodes the referenced object *object* (which should always be of type "@") located at the *cursor* position in the *data* object. (Deprecated in Mac OS X v10.2.)

```
- (void)deserializeObjectAt:(id *)object ofObjCType:(const char *)type
    fromData:(NSData *)data atCursor:(unsigned *)cursor
```

Discussion
The decoded object is not autoreleased.

Availability
Deprecated in Mac OS X v10.2.

See Also
- deserializeDataAt:ofObjCType:atCursor:context: (NSData)

Declared In
NSSerialization.h

serializeObjectAt:ofObjCType:intoData:

Encodes the referenced object *object* (which should always be of type "@") in the *data* object. (Deprecated in Mac OS X v10.2.)

```
- (void)serializeObjectAt:(id *)object ofObjCType:(const char *)type
    intoData:(NSMutableData *)data
```

Availability
Deprecated in Mac OS X v10.2.

See Also
- serializeDataAt:ofObjCType:context: (NSMutableData)

Declared In
NSSerialization.h

NSObject Protocol Reference

Adopted by	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSObject.h
Companion guides	Cocoa Fundamentals Guide Memory Management Programming Guide for Cocoa

Overview

The `NSObject` protocol groups methods that are fundamental to all Objective-C objects.

If an object conforms to this protocol, it can be considered a first-class object. Such an object can be asked about its:

- Class, and the place of its class in the inheritance hierarchy
- Conformance to protocols
- Ability to respond to a particular message

In addition, objects that conform to this protocol—with its retain (page 2108), release (page 2106), and autorelease (page 2099) methods—can also integrate with the object management and deallocation scheme defined in Foundation (for more information see, for example, *Memory Management Programming Guide for Cocoa*). Thus, an object that conforms to the `NSObject` protocol can be managed by container objects like those defined by `NSArray` and `NSDictionary`.

The Cocoa root class, `NSObject`, adopts this protocol, so all objects inheriting from `NSObject` have the features described by this protocol.

Tasks

Identifying Classes

`- class` (page 2100)
Returns the class object for the receiver's class.

- `superclass` (page 2110)
  Returns the class object for the receiver's superclass.

Identifying and Comparing Objects

- `isEqual:` (page 2101)
  Returns a Boolean value that indicates whether the receiver and a given object are equal.
- `hash` (page 2101)
  Returns an integer that can be used as a table address in a hash table structure.
- `self` (page 2109)
  Returns the receiver.

Managing Reference Counts

- `retain` (page 2108)
  Increments the receiver's reference count.
- `release` (page 2106)
  Decrements the receiver's reference count.
- `autorelease` (page 2099)
  Adds the receiver to the current autorelease pool.
- `retainCount` (page 2109)
  Returns the receiver's reference count.

Testing Object Inheritance, Behavior, and Conformance

- `isKindOfClass:` (page 2102)
  Returns a Boolean value that indicates whether the receiver is an instance of given class or an instance of any class that inherits from that class.
- `isMemberOfClass:` (page 2103)
  Returns a Boolean value that indicates whether the receiver is an instance of a given class.
- `respondsToSelector:` (page 2107)
  Returns a Boolean value that indicates whether the receiver implements or inherits a method that can respond to a specified message.
- `conformsToProtocol:` (page 2100)
  Returns a Boolean value that indicates whether the receiver conforms to a given protocol.

Describing Objects

- `description` (page 2100)
  Returns a string that describes the contents of the receiver.

Sending Messages

- `performSelector:` (page 2104)
  Sends a specified message to the receiver and returns the result of the message.
- `performSelector:withObject:` (page 2105)
  Sends a message to the receiver with an object as the argument.
- `performSelector:withObject:withObject:` (page 2105)
  Sends a message to the receiver with two objects as as arguments.

Determining Allocation Zones

- `zone` (page 2110)
  Returns a pointer to the zone from which the receiver was allocated.

Identifying Proxies

- `isProxy` (page 2104)
  Returns a Boolean value that indicates whether the receiver does not descend from `NSObject`.

Instance Methods

autorelease

Adds the receiver to the current autorelease pool.

```
- (id)autorelease
```

Return Value
`self`.

Discussion
You add an object to an autorelease pool so it will receive a `release` message—and thus might be deallocated—when the pool is destroyed. For more information on the autorelease mechanism, see *Memory Management Programming Guide for Cocoa*.

Special Considerations

If garbage collection is enabled, this method is a no-op.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `retain` (page 2108)

Related Sample Code
CoreRecipes
GridCalendar

iSpend
Mountains
PDFKitLinker2

Declared In
NSObject.h

class

Returns the class object for the receiver's class.

```
- (Class)class
```

Return Value
The class object for the receiver's class.

Availability
Available in Mac OS X v10.0 and later.

See Also
+ class (page 1155) (NSObject class)

Declared In
NSObject.h

conformsToProtocol:

Returns a Boolean value that indicates whether the receiver conforms to a given protocol.

```
- (BOOL)conformsToProtocol:(Protocol *)aProtocol
```

Parameters
aProtocol
A protocol object that represents a particular protocol.

Return Value
YES if the receiver conforms to *aProtocol*, otherwise NO.

Discussion
This method works identically to the conformsToProtocol: (page 1156) class method declared in NSObject. It's provided as a convenience so that you don't need to get the class object to find out whether an instance can respond to a given set of messages.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSObject.h

description

Returns a string that describes the contents of the receiver.

```
- (NSString *)description
```

Return Value
A string that describes the contents of the receiver.

Discussion
The debugger's print-object command indirectly invokes this method to produce a textual description of an object.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSObject.h`

hash

Returns an integer that can be used as a table address in a hash table structure.

```
- (NSUInteger)hash
```

Return Value
An integer that can be used as a table address in a hash table structure.

Discussion
If two objects are equal (as determined by the `isEqual:` (page 2101) method), they must have the same hash value. This last point is particularly important if you define `hash` in a subclass and intend to put instances of that subclass into a collection.

If a mutable object is added to a collection that uses hash values to determine the object's position in the collection, the value returned by the `hash` method of the object must not change while the object is in the collection. Therefore, either the `hash` method must not rely on any of the object's internal state information or you must make sure the object's internal state information does not change while the object is in the collection. Thus, for example, a mutable dictionary can be put in a hash table but you must not change it while it is in there. (Note that it can be difficult to know whether or not a given object is in a collection.)

Availability
Available in Mac OS X v10.0 and later.

See Also
`- isEqual:` (page 2101)

Related Sample Code
ImageMapExample
TrackBall

Declared In
`NSObject.h`

isEqual:

Returns a Boolean value that indicates whether the receiver and a given object are equal.

```
- (BOOL)isEqual:(id)anObject
```

Parameters

anObject

The object to be compared to the receiver.

Return Value

YES if the receiver and *anObject* are equal, otherwise NO.

Discussion

This method defines what it means for instances to be equal. For example, a container object might define two containers as equal if their corresponding objects all respond YES to an isEqual: request. See the NSData, NSDictionary, NSArray, and NSString class specifications for examples of the use of this method.

If two objects are equal, they must have the same hash value. This last point is particularly important if you define isEqual: in a subclass and intend to put instances of that subclass into a collection. Make sure you also define hash (page 2101) in your subclass.

Availability

Available in Mac OS X v10.0 and later.

See Also

- hash (page 2101)

Related Sample Code

IdentitySample

Quartz Composer WWDC 2005 TextEdit

Sketch-112

TextEditPlus

Declared In

NSObject.h

isKindOfClass:

Returns a Boolean value that indicates whether the receiver is an instance of given class or an instance of any class that inherits from that class.

```
- (BOOL)isKindOfClass:(Class)aClass
```

Parameters

aClass

A class object representing the Objective-C class to be tested.

Return Value

YES if the receiver is an instance of *aClass* or an instance of any class that inherits from *aClass*, otherwise NO.

Discussion

For example, in this code, isKindOfClass: would return YES because, in Foundation, the NSArchiver class inherits from NSCoder:

```
NSMutableData *myData = [NSMutableData dataWithCapacity:30];
id anArchiver = [[NSArchiver alloc] initForWritingWithMutableData:myData];
if ( [anArchiver isKindOfClass:[NSCoder class]] )
```

```
    ...
```

Be careful when using this method on objects represented by a class cluster. Because of the nature of class clusters, the object you get back may not always be the type you expected. If you call a method that returns a class cluster, the exact type returned by the method is the best indicator of what you can do with that object. For example, if a method returns a pointer to an `NSArray` object, you should not use this method to see if the array is mutable, as shown in the following code:

```
// DO NOT DO THIS!
if ([myArray isKindOfClass:[NSMutableArray class]])
{
    // Modify the object
}
```

If you use such constructs in your code, you might think it is alright to modify an object that in reality should not be modified. Doing so might then create problems for other code that expected the object to remain unchanged.

If the receiver is a class object, this method returns `YES` if *`aClass`* is a Class object of the same type, `NO` otherwise.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- isMemberOfClass:` (page 2103)

Declared In
`NSObject.h`

isMemberOfClass:

Returns a Boolean value that indicates whether the receiver is an instance of a given class.

```
- (BOOL)isMemberOfClass:(Class)aClass
```

Parameters

`aClass`
A class object representing the Objective-C class to be tested.

Return Value
`YES` if the receiver is an instance of *`aClass`*, otherwise `NO`.

Discussion
For example, in this code, `isMemberOfClass:` would return `NO`:

```
NSMutableData *myData = [NSMutableData dataWithCapacity:30];
id anArchiver = [[NSArchiver alloc] initForWritingWithMutableData:myData];
if ([anArchiver isMemberOfClass:[NSCoder class]])
    ...
```

Class objects may be compiler-created objects but they still support the concept of membership. Thus, you can use this method to verify that the receiver is a specific Class object.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `isKindOfClass:` (page 2102)

Declared In
`NSObject.h`

isProxy

Returns a Boolean value that indicates whether the receiver does not descend from `NSObject`.

```
- (BOOL)isProxy
```

Return Value
`NO` if the receiver really descends from `NSObject`, otherwise `YES`.

Discussion
This method is necessary because sending `isKindOfClass:` (page 2102) or `isMemberOfClass:` (page 2103) to an `NSProxy` object will test the object the proxy stands in for, not the proxy itself. Use this method to test if the receiver is a proxy (or a member of some other root class).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSObject.h`

performSelector:

Sends a specified message to the receiver and returns the result of the message.

```
- (id)performSelector:(SEL)aSelector
```

Parameters
aSelector
A selector identifying the message to send. If *aSelector* is `NULL`, an `NSInvalidArgumentException` is raised.

Return Value
An object that is the result of the message.

Discussion
The `performSelector:` method is equivalent to sending an *aSelector* message directly to the receiver. For example, all three of the following messages do the same thing:

```
id myClone = [anObject copy];
id myClone = [anObject performSelector:@selector(copy)];
id myClone = [anObject performSelector:sel_getUid("copy")];
```

However, the `performSelector:` method allows you to send messages that aren't determined until runtime. A variable selector can be passed as the argument:

```
SEL myMethod = findTheAppropriateSelectorForTheCurrentSituation();
[anObject performSelector:myMethod];
```

The *aSelector* argument should identify a method that takes no arguments. For methods that return anything other than an object, use NSInvocation.

Availability
Available in Mac OS X v10.0 and later.

See Also
- performSelector:withObject: (page 2105)
- performSelector:withObject:withObject: (page 2105)

Declared In
NSObject.h

performSelector:withObject:

Sends a message to the receiver with an object as the argument.

```
- (id)performSelector:(SEL)aSelector withObject:(id)anObject
```

Parameters

aSelector
A selector identifying the message to send. If *aSelector* is NULL, an NSInvalidArgumentException is raised.

anObject
An object that is the sole argument of the message.

Return Value
An object that is the result of the message.

Discussion
This method is the same as performSelector: (page 2104) except that you can supply an argument for *aSelector*. *aSelector* should identify a method that takes a single argument of type id. For methods with other argument types and return values, use NSInvocation.

Availability
Available in Mac OS X v10.0 and later.

See Also
- performSelector:withObject:withObject: (page 2105)
- methodForSelector: (page 1181) (NSObject class)

Related Sample Code
SearchField
ToolbarSample

Declared In
NSObject.h

performSelector:withObject:withObject:

Sends a message to the receiver with two objects as as arguments.

```
- (id)performSelector:(SEL)aSelector withObject:(id)anObject
    withObject:(id)anotherObject
```

Parameters

aSelector

A selector identifying the message to send. If *aSelector* is NULL, an NSInvalidArgumentException is raised.

anObject

An object that is the first argument of the message.

anotherObject

An object that is the second argument of the message

Return Value

An object that is the result of the message.

Discussion

This method is the same as performSelector: (page 2104) except that you can supply two arguments for *aSelector*. *aSelector* should identify a method that can take two arguments of type id. For methods with other argument types and return values, use NSInvocation.

Availability

Available in Mac OS X v10.0 and later.

See Also

- performSelector:withObject: (page 2105)
- methodForSelector: (page 1181) (NSObject class)

Declared In

NSObject.h

release

Decrements the receiver's reference count.

```
- (oneway void)release
```

Discussion

The receiver is sent a dealloc (page 1174) message when its reference count reaches 0.

You would only implement this method to define your own reference-counting scheme. Such implementations should not invoke the inherited method; that is, they should not include a release message to super.

For more information on the reference counting mechanism, see *Memory Management Programming Guide for Cocoa*.

Special Considerations

If garbage collection is enabled, this method is a no-op.

You must complete the object initialization (using an init method) before invoking release. For example, the following code shows an error:

```
id anObject = [MyObject alloc];
[anObject release];
```

You may call `release` from within an init method if initialization fails for some reason provided that you have at least called superclass's designated initializer.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CoreRecipes
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus
WhackedTV

Declared In
`NSObject.h`

respondsToSelector:

Returns a Boolean value that indicates whether the receiver implements or inherits a method that can respond to a specified message.

```
- (BOOL)respondsToSelector:(SEL)aSelector
```

Parameters

aSelector
A selector that identifies a message.

Return Value
`YES` if the receiver implements or inherits a method that can respond to *aSelector*, otherwise `NO`.

Discussion
The application is responsible for determining whether a `NO` response should be considered an error.

You cannot test whether an object inherits a method from its superclass by sending `respondsToSelector:` to the object using the `super` keyword. This method will still be testing the object as a whole, not just the superclass's implementation. Therefore, sending `respondsToSelector:` to `super` is equivalent to sending it to `self`. Instead, you must invoke the `NSObject` class method `instancesRespondToSelector:` (page 1161) directly on the object's superclass, as illustrated in the following code fragment.

```
if( [MySuperclass instancesRespondToSelector:@selector(aMethod)] ) {
    // invoke the inherited method
    [super aMethod];
}
```

You cannot simply use `[[self superclass] instancesRespondToSelector:@selector(aMethod)]` since this may cause the method to fail if it is invoked by a subclass.

Note that if the receiver is able to forward *aSelector* messages to another object, it will be able to respond to the message, albeit indirectly, even though this method returns `NO`.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `forwardInvocation:` (page 1177) (`NSObject` class)
+ `instancesRespondToSelector:` (page 1161) (`NSObject` class)

Declared In
`NSObject.h`

retain

Increments the receiver's reference count.

```
- (id)retain
```

Return Value
`self`.

Discussion
You send an object a `retain` message when you want to prevent it from being deallocated without your express permission.

An object is deallocated automatically when its reference count reaches 0. `retain` messages increment the reference count, and `release` (page 2106) messages decrement it. For more information on this mechanism, see *Memory Management Programming Guide for Cocoa*.

As a convenience, `retain` returns `self` because it is often used in nested expressions:

```
NSString *systemApps = [[NSString
    stringWithCString:"/Applications"] retain];
```

You would implement this method only if you were defining your own reference-counting scheme. Such implementations must return `self` and should not invoke the inherited method by sending a `retain` message to `super`.

Special Considerations

If garbage collection is enabled, this method is a no-op.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `autorelease` (page 2099)
- `release` (page 2106)

Related Sample Code
CIAnnotation
CoreRecipes
GridCalendar
Reducer
Sketch-112

Declared In
`NSObject.h`

retainCount

Returns the receiver's reference count.

```
- (NSUInteger)retainCount
```

Return Value
The receiver's reference count.

Discussion
You might override this method in a class to implement your own reference-counting scheme. For objects that never get released (that is, their `release` (page 2106) method does nothing), this method should return `UINT_MAX`, as defined in `<limits.h>`.

The `retainCount` method does not account for any pending `autorelease` (page 2099) messages sent to the receiver.

Important: This method is typically of no value in debugging memory management issues. Because any number of framework objects may have retained an object in order to hold references to it, while at the same time autorelease pools may be holding any number of deferred releases on an object, it is very unlikely that you can get useful information from this method.

To understand the fundamental rules of memory management that you must abide by, read Memory Management Rules. To diagnose memory management problems, use a suitable tool:

- The LLVM/Clang Static analyzer can typically find memory management problems even before you run your program.
- The Object Alloc instrument in the Instruments application (see *Instruments User Guide*) can track object allocation and destruction.
- Shark (see *Shark User Guide*) also profiles memory allocations (amongst numerous other aspects of your program).

Special Considerations

If garbage collection is enabled, the return value is undefined.

Availability
Available in Mac OS X v10.0 and later.

See Also
- `autorelease` (page 2099)
- `retain` (page 2108)

Related Sample Code
SimpleThreads

Declared In
`NSObject.h`

self

Returns the receiver.

```
- (id)self
```

Return Value
The receiver.

Availability
Available in Mac OS X v10.0 and later.

See Also
`- class` (page 2100)

Related Sample Code
GridCalendar
iSpend
Quartz Composer WWDC 2005 TextEdit
StickiesExample
TextEditPlus

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 Mac OS X v10.0 and later.

See Also
`+ superclass` (page 1167) (`NSObject` class)

Declared In
`NSObject.h`

zone

Returns a pointer to the zone from which the receiver was allocated.

```
- (NSZone *)zone
```

Return Value
A pointer to the zone from which the receiver was allocated.

Discussion
Objects created without specifying a zone are allocated from the default zone.

Availability
Available in Mac OS X v10.0 and later.

See Also
`+ allocWithZone:` (page 1152) (`NSObject` class)

Related Sample Code
QTCoreVideo102
QTCoreVideo103
QTCoreVideo201
QTCoreVideo202
QTCoreVideo301

Declared In
`NSObject.h`

NSScriptingComparisonMethods Protocol Reference

~~(informal protocol)~~

Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSScriptWhoseTests.h
Companion guide	Cocoa Scripting Guide

Overview

This informal protocol defines a set of methods useful for comparing script objects.

Often the correct way to compare two objects for scripting is different from the correct way to compare objects programmatically. This informal protocol defines a set of methods that can be implemented to perform a comparison appropriate for scripting that is independent of other methods for doing comparisons.

Cocoa scripting uses these scripting comparison methods, if available, in the process of evaluating specifier tests. If the first object being tested implements the appropriate method for the comparison operation, it will be used. If the first object doesn't implement the appropriate method but the second object implements the inverse, the inverted comparison is performed. For example, instead of determining whether object one is less than object two, Cocoa determines whether object two is greater than object one (but only for the operations `is equal`, `is less than or equal`, `is less than`, `is greater than or equal`, or `is greater than`). If neither of the objects implements the appropriate method, Cocoa falls back on similar comparison operators in the protocol `NSComparisonMethods` (but again, only for the operations `is equal`, `is less than or equal`, `is less than`, `is greater than or equal`, or `is greater than`).

Cocoa provides default implementations of these scripting comparison methods for `NSString` and `NSAttributedString`. You should define implementations of these methods for any of your scriptable objects that need to perform comparisons for scripting purposes that are different than the comparisons provided by `NSComparisonMethods`. If none require different comparison methods, you can implement only the methods you need from `NSScriptingComparisonMethods`.

Tasks

Performing Comparisons

`- scriptingBeginsWith:` (page 2114)

Returns `YES` if, in a scripting comparison, the compared object matches the beginning of *`object`*. A default implementation is provided for `NSString` and `NSAttributedString`.

- `scriptingContains:` (page 2114)
  Returns `YES` if, in a scripting comparison, the compared object contains *`object`*. A default implementation is provided for `NSString` and `NSAttributedString`.
- `scriptingEndsWith:` (page 2115)
  Returns `YES` if, in a scripting comparison, the compared object matches the end of *`object`*. A default implementation is provided for `NSString` and `NSAttributedString`.
- `scriptingIsEqualTo:` (page 2115)
  Returns `YES` if, in a scripting comparison, the compared object is equal to *`object`*. A default implementation is provided for `NSString` and `NSAttributedString`.
- `scriptingIsGreaterThan:` (page 2115)
  Returns `YES` if, in a scripting comparison, the compared object is greater than *`object`*. A default implementation is provided for `NSString` and `NSAttributedString`.
- `scriptingIsGreaterThanOrEqualTo:` (page 2115)
  Returns `YES` if, in a scripting comparison, the compared object is greater than or equal to *`object`*. A default implementation is provided for `NSString` and `NSAttributedString`.
- `scriptingIsLessThan:` (page 2116)
  Returns `YES` if, in a scripting comparison, the compared object is less than *`object`*. A default implementation is provided for `NSString` and `NSAttributedString`.
- `scriptingIsLessThanOrEqualTo:` (page 2116)
  Returns `YES` if, in a scripting comparison, the compared object is less than or equal to *`object`*. A default implementation is provided for `NSString` and `NSAttributedString`.

Instance Methods

scriptingBeginsWith:

Returns `YES` if, in a scripting comparison, the compared object matches the beginning of *`object`*. A default implementation is provided for `NSString` and `NSAttributedString`.

```
- (BOOL)scriptingBeginsWith:(id)object
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptWhoseTests.h`

scriptingContains:

Returns `YES` if, in a scripting comparison, the compared object contains *`object`*. A default implementation is provided for `NSString` and `NSAttributedString`.

```
- (BOOL)scriptingContains:(id)object
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptWhoseTests.h

scriptingEndsWith:

Returns YES if, in a scripting comparison, the compared object matches the end of *object*. A default implementation is provided for NSString and NSAttributedString.

```
- (BOOL)scriptingEndsWith:(id)object
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptWhoseTests.h

scriptingIsEqualTo:

Returns YES if, in a scripting comparison, the compared object is equal to *object*. A default implementation is provided for NSString and NSAttributedString.

```
- (BOOL)scriptingIsEqualTo:(id)object
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptWhoseTests.h

scriptingIsGreaterThan:

Returns YES if, in a scripting comparison, the compared object is greater than *object*. A default implementation is provided for NSString and NSAttributedString.

```
- (BOOL)scriptingIsGreaterThan:(id)object
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptWhoseTests.h

scriptingIsGreaterThanOrEqualTo:

Returns YES if, in a scripting comparison, the compared object is greater than or equal to *object*. A default implementation is provided for NSString and NSAttributedString.

```
- (BOOL)scriptingIsGreaterThanOrEqualTo:(id)object
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptWhoseTests.h

scriptingIsLessThan:

Returns YES if, in a scripting comparison, the compared object is less than *object*. A default implementation is provided for NSString and NSAttributedString.

```
- (BOOL)scriptingIsLessThan:(id)object
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptWhoseTests.h

scriptingIsLessThanOrEqualTo:

Returns YES if, in a scripting comparison, the compared object is less than or equal to *object*. A default implementation is provided for NSString and NSAttributedString.

```
- (BOOL)scriptingIsLessThanOrEqualTo:(id)object
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSScriptWhoseTests.h

NSScriptKeyValueCoding Protocol Reference

(informal protocol)

Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSScriptKeyValueCoding.h
Companion guides	Cocoa Scripting Guide Key-Value Coding Programming Guide

Overview

Cocoa scripting takes advantage of key-value coding to get and set information in scriptable objects. The methods in this category provide additional capabilities for working with key-value coding, including getting and setting key values by index in multivalue keys and coercing (or converting) a key value. Additional methods allow the implementer of a scriptable container class to provide fast access to elements that are being referenced by name and unique ID.

Because Cocoa scripting invokes `setValue:forKey:` and `mutableArrayValueForKey:`, changes to model objects made by AppleScript scripts are observable using automatic key-value observing.

Note: In Mac OS X version 10.3 and earlier, Cocoa scripting did not invoke `setValue:forKey:` or `mutableArrayValueForKey:`, so automatic key-value observing notification was not always done for model object changes caused by scripts. Also, In Mac OS X version 10.4, for backward binary compatibility, Cocoa invokes the now-deprecated method `takeValue:forKey:` instead of `setValue:forKey:`, if `takeValue:forKey:` is overridden.

Tasks

Indexed Access

- `insertValue:atIndex:inPropertyWithKey:` (page 2118)
  Inserts an object at the specified index in the collection specified by the passed key.
- `removeValueAtIndex:fromPropertyWithKey:` (page 2119)
  Removes the object at the specified index from the collection specified by the passed key.
- `replaceValueAtIndex:inPropertyWithKey:withValue:` (page 2119)
  Replaces the object at the specified index in the collection specified by the passed key.
- `valueAtIndex:inPropertyWithKey:` (page 2120)
  Retrieves an indexed object from the collection specified by the passed key.

Access by Name, Key, or ID

- `insertValue:inPropertyWithKey:` (page 2119)
  Inserts an object in the collection specified by the passed key.
- `valueWithName:inPropertyWithKey:` (page 2120)
  Retrieves a named object from the collection specified by the passed key.
- `valueWithUniqueID:inPropertyWithKey:` (page 2120)
  Retrieves an object by ID from the collection specified by the passed key.

Coercion

- `coerceValue:forKey:` (page 2118)
  Uses type info from the class description and `NSScriptCoercionHandler` to attempt to convert *`value`* for *`key`* to the proper type, if necessary.

Instance Methods

coerceValue:forKey:

Uses type info from the class description and `NSScriptCoercionHandler` to attempt to convert *`value`* for *`key`* to the proper type, if necessary.

```
- (id)coerceValue:(id)value forKey:(NSString *)key
```

Discussion
The method `coerceValueFor<Key>:` is used if it exists.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptKeyValueCoding.h`

insertValue:atIndex:inPropertyWithKey:

Inserts an object at the specified index in the collection specified by the passed key.

```
- (void)insertValue:(id)value atIndex:(NSUInteger)index inPropertyWithKey:(NSString
    *)key
```

Discussion
The method `insertIn<Key>:atIndex:` is invoked if it exists. If no corresponding scripting-KVC-compliant method (`insertIn<Key>:atIndex:`) is found, this method invokes `mutableArrayValueForKey:` and mutates the result.

Note: Prior to Mac OS X version 10.4, this method did not invoke `-mutableArrayValueForKey:`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptKeyValueCoding.h`

insertValue:inPropertyWithKey:

Inserts an object in the collection specified by the passed key.

```
- (void)insertValue:(id)value inPropertyWithKey:(NSString *)key
```

Discussion
The method `insertIn<Key>:` is used if it exists. Otherwise, raises an `NSUndefinedKeyException`. This is part of Cocoa's scripting support for inserting newly-created objects into containers without explicitly specifying a location.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`NSScriptKeyValueCoding.h`

removeValueAtIndex:fromPropertyWithKey:

Removes the object at the specified index from the collection specified by the passed key.

```
- (void)removeValueAtIndex:(NSUInteger)index fromPropertyWithKey:(NSString *)key
```

Discussion
The method `removeFrom<Key>AtIndex:` is invoked if it exists. If no corresponding scripting-KVC-compliant method (`-removeFrom<Key>AtIndex:`) is found, this method invokes `-mutableArrayValueForKey:` and mutates the result.

Note: Prior to Mac OS X version 10.4, this method did not invoke `-mutableArrayValueForKey:`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptKeyValueCoding.h`

replaceValueAtIndex:inPropertyWithKey:withValue:

Replaces the object at the specified index in the collection specified by the passed key.

```
- (void)replaceValueAtIndex:(NSUInteger)index inPropertyWithKey:(NSString *)key
    withValue:(id)value
```

Discussion
The method `replaceIn<Key>:atIndex:` is invoked if it exists. If no corresponding scripting-KVC-compliant method (`-replaceIn<Key>atIndex:`) is found, this method invokes `-mutableArrayValueForKey:` and mutates the result.

> **Note:** Prior to Mac OS X version 10.4, this method did not invoke `-mutableArrayValueForKey:`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptKeyValueCoding.h`

valueAtIndex:inPropertyWithKey:

Retrieves an indexed object from the collection specified by the passed key.

```
- (id)valueAtIndex:(NSUInteger)index inPropertyWithKey:(NSString *)key
```

Discussion
This actually works with a single-value key as well if *index* is 0. The method `valueIn<Key>AtIndex:` is used if it exists.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptKeyValueCoding.h`

valueWithName:inPropertyWithKey:

Retrieves a named object from the collection specified by the passed key.

```
- (id)valueWithName:(NSString *)name inPropertyWithKey:(NSString *)key
```

Discussion
The method `valueIn<Key>WithName:` is used if it exists. Otherwise, raises an `NSUndefinedKeyException`.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`NSScriptKeyValueCoding.h`

valueWithUniqueID:inPropertyWithKey:

Retrieves an object by ID from the collection specified by the passed key.

```
- (id)valueWithUniqueID:(id)uniqueID inPropertyWithKey:(NSString *)key
```

Discussion
The method `valueIn<Key>WithUniqueID:` is invoked if it exists. Otherwise, raises an `NSUndefinedKeyException`. The declared type of *`uniqueID`* in the constructed method must be `id`, `NSNumber *`, `NSString *`, or one of the scalar types that can be encapsulated by `NSNumber`.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`NSScriptKeyValueCoding.h`

Constants

NSScriptKeyValueCoding Exception Names

`NSScriptKeyValueCoding` defines the following exception.

```
extern NSString *NSOperationNotSupportedForKeyException;
```

Constants
`NSOperationNotSupportedForKeyException`

Can be raised by key-value coding methods that want to explicitly disallow certain manipulations or accesses.

For instance, a `setKey:` method for a read-only key can raise this exception.

Available in Mac OS X v10.0 and later.

Declared in `NSScriptKeyValueCoding.h`.

Declared In
`NSScriptKeyValueCoding.h`

NSScriptObjectSpecifiers Protocol Reference

(informal protocol)

Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSScriptObjectSpecifiers.h
Companion guide	Cocoa Scripting Guide

Overview

Informal `protocol`. Allows scriptable objects that can provide a fully specified object specifier to themselves within an application to do so. It also enables containers of objects to perform their own specifier evaluation.

For a comprehensive treatment of object specifiers, including sample code, see Object Specifiers in *Cocoa Scripting Guide*.

Tasks

Working with Object Specifiers

- `objectSpecifier` (page 2124)
  Returns an object specifier for the receiver.
- `indicesOfObjectsByEvaluatingObjectSpecifier:` (page 2123)
  Returns the indices of the specified container objects.

Instance Methods

indicesOfObjectsByEvaluatingObjectSpecifier:

Returns the indices of the specified container objects.

```
- (NSArray *)indicesOfObjectsByEvaluatingObjectSpecifier:(NSScriptObjectSpecifier
    *)specifier
```

Parameters

specifier
  An object specifier for the container objects for which to obtain the indices.

Return Value
A zero-based array of `NSNumber` objects that identify the zero-based indices of the container objects that match *`specifier`*, or `nil` if no matching objects were found.

Discussion
Containers that want to evaluate some specifiers on their own should implement this method. If this method returns `nil`, the object specifier will go on to do its own evaluation, so you should only return `nil` if that's the behavior you want, or if an error occurs. If this method returns an array, the object specifier will use the `NSNumber` objects in it as the indices. So, if you evaluate the specifier and there are no objects that match, you should return an empty array, not `nil`. If you find only one object, you should still return its index in an array. Returning an array with a single index where the index is –1 is interpreted to mean all the objects.

For an example implementation, see "Implementing Object Specifiers" in Object Specifiers in *Cocoa Scripting Guide*

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSScriptObjectSpecifiers.h`

objectSpecifier

Returns an object specifier for the receiver.

```
- (NSScriptObjectSpecifier *)objectSpecifier
```

Return Value
A fully specified object specifier to the receiver within the application.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus

Declared In
`NSScriptObjectSpecifiers.h`

NSURLAuthenticationChallengeSender Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSURLAuthenticationChallenge.h
Availability	Available in Mac OS X v10.2 with Safari 1.0 installed. Available in Mac OS X v10.2.7 and later.
Companion guide	URL Loading System

Overview

The `NSURLAuthenticationChallengeSender` protocol represents the interface that the sender of an authentication challenge must implement.

The methods in the protocol are generally sent by a delegate in response to receiving a `connection:didReceiveAuthenticationChallenge:` (page 1761) or `download:didReceiveAuthenticationChallenge:` (page 1788). The different methods provide different ways of responding to authentication challenges.

Tasks

Protocol Methods

- `cancelAuthenticationChallenge:` (page 2126)
  Cancels a given authentication challenge.
- `continueWithoutCredentialForAuthenticationChallenge:` (page 2126)
  Attempt to continue downloading a request without providing a credential for a given challenge.
- `useCredential:forAuthenticationChallenge:` (page 2126)
  Attempt to use a given credential for a given authentication challenge.

Instance Methods

cancelAuthenticationChallenge:

Cancels a given authentication challenge.

```
- (void)cancelAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
```

Parameters

challenge

The authentication challenge to cancel.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

NSURLAuthenticationChallenge.h

continueWithoutCredentialForAuthenticationChallenge:

Attempt to continue downloading a request without providing a credential for a given challenge.

```
- (void)continueWithoutCredentialForAuthenticationChallenge:(NSURLAuthenticationChallenge
 *)challenge
```

Parameters

challenge

A challenge without authentication credentials.

Discussion

This method has no effect if it is called with an authentication challenge that has already been handled.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

NSURLAuthenticationChallenge.h

useCredential:forAuthenticationChallenge:

Attempt to use a given credential for a given authentication challenge.

```
- (void)useCredential:(NSURLCredential *)credential
   forAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
```

Parameters

credential

The credential to use for authentication.

challenge

The challenge for which to use *credential*.

Discussion

This method has no effect if it is called with an authentication challenge that has already been handled.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

NSURLAuthenticationChallenge.h

NSURLClient Protocol Reference (Not Recommended)

~~(informal protocol)~~

Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSURL.h
Availability	Deprecated in Mac OS X v10.4 and later.

Important: `NSURLClient` is deprecated in Mac OS X v10.4 and later. Applications that are intended for deployment on Mac OS X v10.3 or later should use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.

Overview

`NSURLClient` is deprecated in Mac OS X v10.4 and later. Applications that are intended for deployment on Mac OS X v10.3 or later should use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.

Tasks

Working with URL Clients

- `URL:resourceDataDidBecomeAvailable:` (page 2129) Deprecated in Mac OS X v10.4 and later
  Notifies the URL client that the URL has loaded new data.
- `URL:resourceDidFailLoadingWithReason:` (page 2130) Deprecated in Mac OS X v10.4 and later
  Notifies the URL client that the URL failed to load its resource data.
- `URLResourceDidCancelLoading:` (page 2130) Deprecated in Mac OS X v10.4 and later
  Notifies the URL client that the URL stopped loading its resource data because loading was canceled.
- `URLResourceDidFinishLoading:` (page 2131) Deprecated in Mac OS X v10.4 and later
  Notifies the URL client that the URL has finished loading its resource data.

Instance Methods

URL:resourceDataDidBecomeAvailable:

Notifies the URL client that the URL has loaded new data. (Deprecated in Mac OS X v10.4 and later.)

```
- (void)URL:(NSURL *)sender resourceDataDidBecomeAvailable:(NSData *)newBytes
```

Parameters

sender

The URL that has loaded new data.

newBytes

The newly loaded data.

Availability
Deprecated in Mac OS X v10.4 and later.

Declared In
NSURL.h

URL:resourceDidFailLoadingWithReason:

Notifies the URL client that the URL failed to load its resource data. (Deprecated in Mac OS X v10.4 and later.)

```
- (void)URL:(NSURL *)sender resourceDidFailLoadingWithReason:(NSString *)reason
```

Parameters

sender

The URL that failed to load its resource data.

reason

The reason the load failed.

Availability
Deprecated in Mac OS X v10.4 and later.

Declared In
NSURL.h

URLResourceDidCancelLoading:

Notifies the URL client that the URL stopped loading its resource data because loading was canceled. (Deprecated in Mac OS X v10.4 and later.)

```
- (void)URLResourceDidCancelLoading:(NSURL *)sender
```

Parameters

sender

The URL that stopped loading its resource data because loading was canceled.

Availability
Deprecated in Mac OS X v10.4 and later.

Declared In
NSURL.h

URLResourceDidFinishLoading:

Notifies the URL client that the URL has finished loading its resource data. (Deprecated in Mac OS X v10.4 and later.)

```
- (void)URLResourceDidFinishLoading:(NSURL *)sender
```

Parameters

sender

The URL that has finished loading its resource data.

Availability

Deprecated in Mac OS X v10.4 and later.

Declared In

NSURL.h

NSURLHandleClient Protocol Reference

Adopted by	NSURL
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	Foundation/NSURLHandle.h

`NSURLHandleClient` is deprecated in Mac OS X v10.4 and later. Applications that are intended for deployment on Mac OS X v10.3 or later should use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.

Overview

This protocol defines the interface for clients to `NSURLHandle`.

Tasks

Notification Methods

`- URLHandle:resourceDataDidBecomeAvailable:` (page 2134) Deprecated in Mac OS X v10.4 and later

Sent periodically by an URL handle when new resource data becomes available. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

`- URLHandle:resourceDidFailLoadingWithReason:` (page 2134) Deprecated in Mac OS X v10.4 and later

Sent when the URL handle failed to load resource data for some reason other than being canceled. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

`- URLHandleResourceDidBeginLoading:` (page 2134) Deprecated in Mac OS X v10.4 and later

Sent when an URL handle begins loading resource data. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

`- URLHandleResourceDidCancelLoading:` (page 2135) Deprecated in Mac OS X v10.4 and later

Sent when an URL handle has canceled loading resource data in response to a programmatic request. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

`- URLHandleResourceDidFinishLoading:` (page 2135) Deprecated in Mac OS X v10.4 and later

Sent when an URL handle finishes loading resource data. (Deprecated. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

Instance Methods

URLHandle:resourceDataDidBecomeAvailable:

Sent periodically by an URL handle when new resource data becomes available. (Deprecated in Mac OS X v10.4 and later. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

```
- (void)URLHandle:(NSURLHandle *)sender resourceDataDidBecomeAvailable:(NSData
    *)newBytes
```

Parameters

sender

The URL handle sending the message.

newBytes

The newly available data.

Availability
Deprecated in Mac OS X v10.4 and later.

Declared In
`NSURLHandle.h`

URLHandle:resourceDidFailLoadingWithReason:

Sent when the URL handle failed to load resource data for some reason other than being canceled. (Deprecated in Mac OS X v10.4 and later. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

```
- (void)URLHandle:(NSURLHandle *)sender resourceDidFailLoadingWithReason:(NSString
    *)reason
```

Parameters

sender

The URL handle sending the message.

reason

A human-readable, localized string describing why the load failed.

Availability
Deprecated in Mac OS X v10.4 and later.

Declared In
`NSURLHandle.h`

URLHandleResourceDidBeginLoading:

Sent when an URL handle begins loading resource data. (Deprecated in Mac OS X v10.4 and later. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

```
- (void)URLHandleResourceDidBeginLoading:(NSURLHandle *)sender
```

Parameters

sender

The URL handle sending the message.

Availability

Deprecated in Mac OS X v10.4 and later.

Declared In

NSURLHandle.h

URLHandleResourceDidCancelLoading:

Sent when an URL handle has canceled loading resource data in response to a programmatic request. (Deprecated in Mac OS X v10.4 and later. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

```
- (void)URLHandleResourceDidCancelLoading:(NSURLHandle *)sender
```

Parameters

sender

The URL handle sending the message.

Availability

Deprecated in Mac OS X v10.4 and later.

Declared In

NSURLHandle.h

URLHandleResourceDidFinishLoading:

Sent when an URL handle finishes loading resource data. (Deprecated in Mac OS X v10.4 and later. Use `NSURLConnection` or `NSURLDownload` instead; see *URL Loading System*.)

```
- (void)URLHandleResourceDidFinishLoading:(NSURLHandle *)sender
```

Parameters

sender

The URL handle sending the message.

Availability

Deprecated in Mac OS X v10.4 and later.

Declared In

NSURLHandle.h

NSURLProtocolClient Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSURLProtocol.h
Availability	Available in Mac OS X v10.2 with Safari 1.0 installed. Available in Mac OS X v10.2.7 and later.
Companion guide	URL Loading System

Overview

The `NSURLProtocolClient` protocol provides the interface used by `NSURLProtocol` subclasses to communicate with the URL loading system. An application should never have the need to implement this protocol.

Tasks

Protocol Methods

- `URLProtocol:cachedResponseIsValid:` (page 2138)
  Sent to indicate to the URL loading system that a cached response is valid.
- `URLProtocol:didCancelAuthenticationChallenge:` (page 2138)
  Sent to indicate to the URL loading system that an authentication challenge has been canceled.
- `URLProtocol:didFailWithError:` (page 2139)
  Sent when the load request fails due to an error.
- `URLProtocol:didLoadData:` (page 2139)
  An NSURLProtocol subclass instance, *`protocol`*, sends this message to `[protocol client]` as it loads *`data`*.
- `URLProtocol:didReceiveAuthenticationChallenge:` (page 2139)
  Sent to indicate to the URL loading system that an authentication challenge has been received.
- `URLProtocol:didReceiveResponse:cacheStoragePolicy:` (page 2140)
  Sent to indicate to the URL loading system that the protocol implementation has created a response object for the request.

- URLProtocol:wasRedirectedToRequest:redirectResponse: (page 2140)
  Sent to indicate to the URL loading system that the protocol implementation has been redirected.
- URLProtocolDidFinishLoading: (page 2141)
  Sent to indicate to the URL loading system that the protocol implementation has finished loading.

Instance Methods

URLProtocol:cachedResponseIsValid:

Sent to indicate to the URL loading system that a cached response is valid.

```
- (void)URLProtocol:(NSURLProtocol *)protocol
    cachedResponseIsValid:(NSCachedURLResponse *)cachedResponse
```

Parameters

protocol
The URL protocol object sending the message.

cachedResponse
The cached response whose validity is being communicated.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSURLProtocol.h

URLProtocol:didCancelAuthenticationChallenge:

Sent to indicate to the URL loading system that an authentication challenge has been canceled.

```
- (void)URLProtocol:(NSURLProtocol *)protocol
    didCancelAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
```

Parameters

protocol
The URL protocol object sending the message.

challenge
The authentication challenge that was canceled.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSURLProtocol.h

URLProtocol:didFailWithError:

Sent when the load request fails due to an error.

```
- (void)URLProtocol:(NSURLProtocol *)protocol didFailWithError:(NSError *)error
```

Parameters

protocol

The URL protocol object sending the message.

error

The error that caused the failure of the load request.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

NSURLProtocol.h

URLProtocol:didLoadData:

An NSURLProtocol subclass instance, *protocol*, sends this message to `[protocol client]` as it loads *data*.

```
- (void)URLProtocol:(NSURLProtocol *)protocol didLoadData:(NSData *)data
```

Discussion

The data object must contain only new data loaded since the previous invocation of this method.

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

NSURLProtocol.h

URLProtocol:didReceiveAuthenticationChallenge:

Sent to indicate to the URL loading system that an authentication challenge has been received.

```
- (void)URLProtocol:(NSURLProtocol *)protocol
    didReceiveAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
```

Parameters

protocol

The URL protocol object sending the message.

challenge

The authentication challenge that has been received.

Discussion

The protocol client guarantees that it will answer the request on the same thread that called this method. The client may add a default credential to the challenge it issues to the connection delegate, if *protocol* did not provide one.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSURLProtocol.h

URLProtocol:didReceiveResponse:cacheStoragePolicy:

Sent to indicate to the URL loading system that the protocol implementation has created a response object for the request.

```
- (void)URLProtocol:(NSURLProtocol *)protocol didReceiveResponse:(NSURLResponse
    *)response cacheStoragePolicy:(NSURLCacheStoragePolicy)policy
```

Parameters

protocol
The URL protocol object sending the message.

response
The newly available response object.

policy
The cache storage policy for the response.

Discussion
The implementation should provide the NSURLCacheStoragePolicy that should be used if the response is to be stored in a cache as the *policy* value.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSURLProtocol.h

URLProtocol:wasRedirectedToRequest:redirectResponse:

Sent to indicate to the URL loading system that the protocol implementation has been redirected.

```
- (void)URLProtocol:(NSURLProtocol *)protocol wasRedirectedToRequest:(NSURLRequest
    *)request redirectResponse:(NSURLResponse *)redirectResponse
```

Parameters

protocol
The URL protocol object sending the message.

request
The new request that the original request was redirected to.

redirectResponse
The response from the original request that caused the redirect.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In
NSURLProtocol.h

URLProtocolDidFinishLoading:

Sent to indicate to the URL loading system that the protocol implementation has finished loading.

```
- (void)URLProtocolDidFinishLoading:(NSURLProtocol *)protocol
```

Parameters

protocol
The URL protocol object sending the message.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
NSURLProtocol.h

Functions

Foundation Functions Reference

Framework: Foundation/Foundation.h

Overview

This chapter describes the functions and function-like macros defined in the Foundation Framework.

Functions by Task

Assertions

For additional information about Assertions, see *Assertions and Logging*.

`NSAssert` (page 2159)
Generates an assertion if a given condition is false.

`NSAssert1` (page 2160)
Generates an assertion if a given condition is false.

`NSAssert2` (page 2161)
Generates an assertion if a given condition is false.

`NSAssert3` (page 2162)
Generates an assertion if a given condition is false.

`NSAssert4` (page 2163)
Generates an assertion if a given condition is false.

`NSAssert5` (page 2165)
Generates an assertion if a given condition is false.

`NSCAssert` (page 2166)
Generates an assertion if the given condition is false.

`NSCAssert1` (page 2166)
`NSCAssert1` is one of a series of macros that generate assertions if the given condition is false.

`NSCAssert2` (page 2167)
`NSCAssert2` is one of a series of macros that generate assertions if the given condition is false.

`NSCAssert3` (page 2168)
`NSCAssert3` is one of a series of macros that generate assertions if the given condition is false.

`NSCAssert4` (page 2169)
`NSCAssert4` is one of a series of macros that generate assertions if the given condition is false.

`NSCAssert5` (page 2169)
`NSCAssert5` is one of a series of macros that generate assertions if the given condition is false.

`NSCParameterAssert` (page 2177)
Evaluates the specified parameter.

`NSParameterAssert` (page 2225)
Validates the specified parameter.

Bundles

For additional information on generating strings files see Strings Files in *Internationalization Programming Topics*.

`NSLocalizedString` (page 2209)
Returns a localized version of a string.

`NSLocalizedStringFromTable` (page 2209)
Returns a localized version of a string.

`NSLocalizedStringFromTableInBundle` (page 2210)
Returns a localized version of a string.

`NSLocalizedStringWithDefaultValue` (page 2210)
Returns a localized version of a string.

Byte Ordering

`NSConvertHostDoubleToSwapped` (page 2172)
Performs a type conversion.

`NSConvertHostFloatToSwapped` (page 2172)
Performs a type conversion.

`NSConvertSwappedDoubleToHost` (page 2173)
Performs a type conversion.

`NSConvertSwappedFloatToHost` (page 2173)
Performs a type conversion.

`NSHostByteOrder` (page 2200)
Returns the endian format.

`NSSwapBigDoubleToHost` (page 2242)
A utility for swapping the bytes of a number.

`NSSwapBigFloatToHost` (page 2243)
A utility for swapping the bytes of a number.

`NSSwapBigIntToHost` (page 2243)
A utility for swapping the bytes of a number.

`NSSwapBigLongLongToHost` (page 2243)
A utility for swapping the bytes of a number.

`NSSwapBigLongToHost` (page 2244)
A utility for swapping the bytes of a number.

`NSSwapLong` (page 2254)
A utility for swapping the bytes of a number.

`NSSwapLongLong` (page 2254)
A utility for swapping the bytes of a number.

`NSSwapShort` (page 2255)
A utility for swapping the bytes of a number.

Decimals

You can also use the class `NSDecimalNumber` for decimal arithmetic.

`NSDecimalAdd` (page 2181)
Adds two decimal values.

`NSDecimalCompact` (page 2182)
Compacts the decimal structure for efficiency.

`NSDecimalCompare` (page 2182)
Compares two decimal values.

`NSDecimalCopy` (page 2183)
Copies the value of a decimal number.

`NSDecimalDivide` (page 2183)
Divides one decimal value by another.

`NSDecimalIsNotANumber` (page 2183)
Returns a Boolean that indicates whether a given decimal contains a valid number.

`NSDecimalMultiply` (page 2184)
Multiplies two decimal numbers together.

`NSDecimalMultiplyByPowerOf10` (page 2184)
Multiplies a decimal by the specified power of 10.

`NSDecimalNormalize` (page 2185)
Normalizes the internal format of two decimal numbers to simplify later operations.

`NSDecimalPower` (page 2185)
Raises the decimal value to the specified power.

`NSDecimalRound` (page 2186)
Rounds off the decimal value.

`NSDecimalString` (page 2186)
Returns a string representation of the decimal value.

`NSDecimalSubtract` (page 2187)
Subtracts one decimal value from another.

Exception Handling

You can find the following macros implemented in `NSException.h`. *Exception Programming Topics for Cocoa* discusses these macros and gives examples of their usage. These macros are useful for code that needs to run on versions of the system prior to Mac OS X v10.3 For later versions of the operating system, you should use the Objective-C compiler directives `@try`, `@catch`, `@throw`, and `@finally`; for information about these directives, see Exception Handling in *The Objective-C 2.0 Programming Language*.

NS_DURING (page 2261)
Marks the start of the exception-handling domain.

NS_ENDHANDLER (page 2261)
Marks the end of the local event handler.

NS_HANDLER (page 2262)
Marks the end of the exception-handling domain and the start of the local exception handler.

NS_VALUERETURN (page 2262)
Permits program control to exit from an exception-handling domain with a value of a specified type.

NS_VOIDRETURN (page 2262)
Permits program control to exit from an exception-handling domain.

Java Setup

NSJavaBundleCleanup (page 2205) Deprecated in Mac OS X v10.5
This function has been deprecated.

NSJavaBundleSetup (page 2205) Deprecated in Mac OS X v10.5
This function has been deprecated.

NSJavaClassesForBundle (page 2205) Deprecated in Mac OS X v10.5
Loads the Java classes located in the specified bundle.

NSJavaClassesFromPath (page 2206) Deprecated in Mac OS X v10.5
Loads the Java classes located at the specified path.

NSJavaNeedsToLoadClasses (page 2206) Deprecated in Mac OS X v10.5
Returns a Boolean value that indicates whether a virtual machine is needed or if Java classes are provided.

NSJavaNeedsVirtualMachine (page 2207) Deprecated in Mac OS X v10.5
Returns a Boolean value that indicates whether a Java virtual machine is required.

NSJavaObjectNamedInPath (page 2207) Deprecated in Mac OS X v10.5
Creates an instance of the named class using the class loader previously specified at the given path.

NSJavaProvidesClasses (page 2207) Deprecated in Mac OS X v10.5
Returns a Boolean value that indicates whether Java classes are provided.

NSJavaSetup (page 2208) Deprecated in Mac OS X v10.5
Loads the Java virtual machine with specified parameters.

NSJavaSetupVirtualMachine (page 2208) Deprecated in Mac OS X v10.5
Sets up the Java virtual machine.

Hash Tables

NSAllHashTableObjects (page 2157)
Returns all of the elements in the specified hash table.

NSCompareHashTables (page 2171)
Returns a Boolean value that indicates whether the elements of two hash tables are equal.

NSCopyHashTableWithZone (page 2174)
Performs a shallow copy of the specified hash table.

NSCountHashTable (page 2176)
Returns the number of elements in a hash table.

NSCreateHashTable (page 2177)
Creates and returns a new hash table.

NSCreateHashTableWithZone (page 2178)
Creates a new hash table in a given zone.

NSEndHashTableEnumeration (page 2189)
Used when finished with an enumerator.

NSEnumerateHashTable (page 2190)
Creates an enumerator for the specified hash table.

NSFreeHashTable (page 2193)
Deletes the specified hash table.

NSHashGet (page 2195)
Returns an element of the hash table.

NSHashInsert (page 2196)
Adds an element to the specified hash table.

NSHashInsertIfAbsent (page 2196)
Adds an element to the specified hash table only if the table does not already contain the element.

NSHashInsertKnownAbsent (page 2197)
Adds an element to the specified hash table.

NSHashRemove (page 2197)
Removes an element from the specified hash table.

NSNextHashEnumeratorItem (page 2223)
Returns the next hash-table element in the enumeration.

NSResetHashTable (page 2232)
Deletes the elements of the specified hash table.

NSStringFromHashTable (page 2239)
Returns a string describing the hash table's contents.

HFS File Types

NSFileTypeForHFSTypeCode (page 2193)
Returns a string encoding a file type code.

NSHFSTypeCodeFromFileType (page 2198)
Returns a file type code.

NSHFSTypeOfFile (page 2199)
Returns a string encoding a file type.

Managing Map Tables

NSAllMapTableKeys (page 2157)
Returns all of the keys in the specified map table.

`NSAllMapTableValues` (page 2158)
Returns all of the values in the specified table.

`NSCompareMapTables` (page 2171)
Compares the elements of two map tables for equality.

`NSCopyMapTableWithZone` (page 2174)
Performs a shallow copy of the specified map table.

`NSCountMapTable` (page 2176)
Returns the number of elements in a map table.

`NSCreateMapTable` (page 2178)
Creates a new map table in the default zone.

`NSCreateMapTableWithZone` (page 2179)
Creates a new map table in the specified zone.

`NSEndMapTableEnumeration` (page 2190)
Used when finished with an enumerator.

`NSEnumerateMapTable` (page 2190)
Creates an enumerator for the specified map table.

`NSFreeMapTable` (page 2194)
Deletes the specified map table.

`NSMapGet` (page 2215)
Returns a map table value for the specified key.

`NSMapInsert` (page 2216)
Inserts a key-value pair into the specified table.

`NSMapInsertIfAbsent` (page 2216)
Inserts a key-value pair into the specified table.

`NSMapInsertKnownAbsent` (page 2217)
Inserts a key-value pair into the specified table if the pair had not been previously added.

`NSMapMember` (page 2218)
Indicates whether a given table contains a given key.

`NSMapRemove` (page 2218)
Removes a key and corresponding value from the specified table.

`NSNextMapEnumeratorPair` (page 2223)
Returns a Boolean value that indicates whether the next map-table pair in the enumeration are set.

`NSResetMapTable` (page 2232)
Deletes the elements of the specified map table.

`NSStringFromMapTable` (page 2239)
Returns a string describing the map table's contents.

Managing Object Allocation and Deallocation

`NSAllocateObject` (page 2159)
Creates and returns a new instance of a given class.

`NSCopyObject` (page 2175)
Creates an exact copy of an object.

`NSDeallocateObject` (page 2181)
Destroys an existing object.

`NSDecrementExtraRefCountWasZero` (page 2187)
Decrements the specified object's reference count.

`NSExtraRefCount` (page 2193)
Returns the specified object's reference count.

`NSIncrementExtraRefCount` (page 2201)
Increments the specified object's reference count.

`NSShouldRetainWithZone` (page 2236)
Indicates whether an object should be retained.

Interacting with the Objective-C Runtime

`NSGetSizeAndAlignment` (page 2195)
Obtains the actual size and the aligned size of an encoded type.

`NSClassFromString` (page 2170)
Obtains a class by name.

`NSStringFromClass` (page 2238)
Returns the name of a class as a string.

`NSSelectorFromString` (page 2234)
Returns the selector with a given name.

`NSStringFromSelector` (page 2241)
Returns a string representation of a given selector.

`NSStringFromProtocol` (page 2240)
Returns the name of a protocol as a string.

`NSProtocolFromString` (page 2228)
Returns a the protocol with a given name.

Logging Output

`NSLog` (page 2211)
Logs an error message to the Apple System Log facility.

`NSLogv` (page 2212)
Logs an error message to the Apple System Log facility.

Managing File Paths

`NSFullUserName` (page 2194)
Returns a string containing the full name of the current user.

`NSHomeDirectory` (page 2199)
Returns the path to the current user's home directory.

`NSHomeDirectoryForUser` (page 2200)
Returns the path to a given user's home directory.

`NSOpenStepRootDirectory` (page 2225)
Returns the root directory of the user's system.

`NSSearchPathForDirectoriesInDomains` (page 2234)
Creates a list of directory search paths.

`NSTemporaryDirectory` (page 2255)
Returns the path of the temporary directory for the current user.

`NSUserName` (page 2257)
Returns the logon name of the current user.

Managing Points

`NSEqualPoints` (page 2191)
Returns a Boolean value that indicates whether two points are equal.

`NSMakePoint` (page 2214)
Creates a new `NSPoint` from the specified values.

`NSPointFromString` (page 2227)
Returns a point from a text-based representation.

`NSStringFromPoint` (page 2239)
Returns a string representation of a point.

`NSPointFromCGPoint` (page 2226)
Returns an `NSPoint` typecast from a `CGPoint`.

`NSPointToCGPoint` (page 2228)
Returns a `CGPoint` typecast from an `NSPoint`.

Managing Ranges

`NSEqualRanges` (page 2191)
Returns a Boolean value that indicates whether two given ranges are equal.

`NSIntersectionRange` (page 2203)
Returns the intersection of the specified ranges.

`NSLocationInRange` (page 2211)
Returns a Boolean value that indicates whether a specified position is in a given range.

`NSMakeRange` (page 2214)
Creates a new NSRange from the specified values.

`NSMaxRange` (page 2219)
Returns the number 1 greater than the maximum value within the range.

`NSRangeFromString` (page 2229)
Returns a range from a text-based representation.

`NSStringFromRange` (page 2240)
Returns a string representation of a range.

`NSUnionRange` (page 2256)
Returns the union of the specified ranges.

Managing Rectangles

`NSContainsRect` (page 2172)
Returns a Boolean value that indicates whether one rectangle completely encloses another.

`NSDivideRect` (page 2188)
Divides a rectangle into two new rectangles.

`NSEqualRects` (page 2192)
Returns a Boolean value that indicates whether the two rectangles are equal.

`NSIsEmptyRect` (page 2204)
Returns a Boolean value that indicates whether a given rectangle is empty.

`NSHeight` (page 2198)
Returns the height of a given rectangle.

`NSInsetRect` (page 2201)
Insets a rectangle by a specified amount.

`NSIntegralRect` (page 2202)
Adjusts the sides of a rectangle to integer values.

`NSIntersectionRect` (page 2203)
Calculates the intersection of two rectangles.

`NSIntersectsRect` (page 2204)
Returns a Boolean value that indicates whether two rectangles intersect.

`NSMakeRect` (page 2214)
Creates a new `NSRect` from the specified values.

`NSMaxX` (page 2219)
Returns the largest x coordinate of a given rectangle.

`NSMaxY` (page 2220)
Returns the largest y coordinate of a given rectangle.

`NSMidX` (page 2220)
Returns the x coordinate of a given rectangle's midpoint.

`NSMidY` (page 2221)
Returns the y coordinate of a given rectangle's midpoint.

`NSMinX` (page 2221)
Returns the smallest x coordinate of a given rectangle.

`NSMinY` (page 2222)
Returns the smallest y coordinate of a given rectangle.

`NSMouseInRect` (page 2222)
Returns a Boolean value that indicates whether the point is in the specified rectangle.

`NSOffsetRect` (page 2224)
Offsets the rectangle by the specified amount.

`NSPointInRect` (page 2227)
Returns a Boolean value that indicates whether a given point is in a given rectangle.

`NSRectFromString` (page 2231)
Returns a rectangle from a text-based representation.

`NSStringFromRect` (page 2241)
Returns a string representation of a rectangle.

`NSRectFromCGRect` (page 2230)
Returns an `NSRect` typecast from a `CGRect`.

`NSRectToCGRect` (page 2231)
Returns a `CGRect` typecast from an `NSRect`.

`NSUnionRect` (page 2256)
Calculates the union of two rectangles.

`NSWidth` (page 2257)
Returns the width of the specified rectangle.

Managing Sizes

`NSEqualSizes` (page 2192)
Returns a Boolean that indicates whether two size values are equal.

`NSMakeSize` (page 2215)
Returns a new `NSSize` from the specified values.

`NSSizeFromString` (page 2237)
Returns an `NSSize` from a text-based representation.

`NSStringFromSize` (page 2242)
Returns a string representation of a size.

`NSSizeFromCGSize` (page 2237)
Returns an `NSSize` typecast from a `CGSize`.

`NSSizeToCGSize` (page 2237)
Returns a `CGSize` typecast from an `NSSize`.

Uncaught Exception Handlers

Whether there's an uncaught exception handler function, any uncaught exceptions cause the program to terminate, unless the exception is raised during the posting of a notification.

`NSGetUncaughtExceptionHandler` (page 2195)
Returns the top-level error handler.

`NSSetUncaughtExceptionHandler` (page 2235)
Changes the top-level error handler.

Managing Memory

`NSDefaultMallocZone` (page 2188)
Returns the default zone.

`NSAllocateCollectable` (page 2158)
Allocates collectable memory.

`NSReallocateCollectable` (page 2229)
Reallocates collectable memory.

`NSMakeCollectable` (page 2213)
Makes a newly allocated Core Foundation object eligible for collection.

`NSAllocateMemoryPages` (page 2158)
Allocates a new block of memory.

`NSCopyMemoryPages` (page 2175)
Copies a block of memory.

`NSDeallocateMemoryPages` (page 2180)
Deallocates the specified block of memory.

`NSLogPageSize` (page 2212)
Returns the binary log of the page size.

`NSPageSize` (page 2225)
Returns the number of bytes in a page.

`NSRealMemoryAvailable` (page 2230)
Returns information about the user's system.

`NSRoundDownToMultipleOfPageSize` (page 2233)
Returns the specified number of bytes rounded down to a multiple of the page size.

`NSRoundUpToMultipleOfPageSize` (page 2233)
Returns the specified number of bytes rounded up to a multiple of the page size.

Managing Zones

`NSCreateZone` (page 2180)
Creates a new zone.

`NSRecycleZone` (page 2232)
Frees memory in a zone.

`NSSetZoneName` (page 2236)
Sets the name of the specified zone.

`NSZoneCalloc` (page 2258)
Allocates memory in a zone.

`NSZoneFree` (page 2258)
Deallocates a block of memory in the specified zone.

`NSZoneFromPointer` (page 2259)
Gets the zone for a given block of memory.

`NSZoneMalloc` (page 2259)
Allocates memory in a zone.

`NSZoneName` (page 2260)
Returns the name of the specified zone.

`NSZoneRealloc` (page 2260)
Allocates memory in a zone.

Functions

NSAllHashTableObjects

Returns all of the elements in the specified hash table.

```
NSArray * NSAllHashTableObjects (
   NSHashTable *table
);
```

Return Value
An array object containing all the elements of *table*.

Discussion
This function should be called only when the table elements are objects, not when they're any other data type.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSCreateHashTable (page 2177)
NSFreeHashTable (page 2193)

Declared In
NSHashTable.h

NSAllMapTableKeys

Returns all of the keys in the specified map table.

```
NSArray * NSAllMapTableKeys (
   NSMapTable *table
);
```

Return Value
An array object containing all the keys in *table*. This function should be called only when *table* keys are objects, not when they're any other type of pointer.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSMapMember (page 2218)
NSMapGet (page 2215)
NSEnumerateMapTable (page 2190)
NSNextMapEnumeratorPair (page 2223)
NSAllMapTableValues (page 2158)

Declared In
NSMapTable.h

NSAllMapTableValues

Returns all of the values in the specified table.

```
NSArray * NSAllMapTableValues (
   NSMapTable *table
);
```

Return Value
An array object containing all the values in *table*. This function should be called only when *table* values are objects, not when they're any other type of pointer.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSMapMember (page 2218)
NSMapGet (page 2215)
NSEnumerateMapTable (page 2190)
NSNextMapEnumeratorPair (page 2223)
NSAllMapTableKeys (page 2157)

Declared In
NSMapTable.h

NSAllocateCollectable

Allocates collectable memory.

```
void *__strong NSAllocateCollectable (
   NSUInteger size,
   NSUInteger options
);
```

Parameters

size
The number of bytes of memory to allocate.

options
0 or NSScannedOption: A value of 0 allocates nonscanned memory; a value of NSScannedOption allocates scanned memory.

Return Value
A pointer to the allocated memory, or NULL if the function is unable to allocate the requested memory.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSZone.h

NSAllocateMemoryPages

Allocates a new block of memory.

```
void * NSAllocateMemoryPages (
   NSUInteger bytes
);
```

Discussion
Allocates the integral number of pages whose total size is closest to, but not less than, *byteCount*. The allocated pages are guaranteed to be filled with zeros. If the allocation fails, raises `NSInvalidArgumentException`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSCopyMemoryPages` (page 2175)
`NSDeallocateMemoryPages` (page 2180)

Declared In
`NSZone.h`

NSAllocateObject

Creates and returns a new instance of a given class.

```
id NSAllocateObject (
   Class aClass,
   NSUInteger extraBytes,
   NSZone *zone
);
```

Parameters

aClass
The class of which to create an instance.

extraBytes
The number of extra bytes required for indexed instance variables (this value is typically `0`).

zone
The zone in which to create the new instance (pass `NULL` to specify the default zone).

Return Value
A new instance of *aClass* or `nil` if an instance could not be created.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSCopyObject` (page 2175)
`NSDeallocateObject` (page 2181)

Declared In
`NSObject.h`

NSAssert

Generates an assertion if a given condition is false.

```
#define NSAssert(condition, desc)
```

Parameters

condition

An expression that evaluates to `YES` or `NO`.

desc

An `NSString` object that contains an error message describing the failure condition.

Discussion

The `NSAssert` macro evaluates the condition and serves as a front end to the assertion handler.

Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes the method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception. If *condition* evaluates to `NO`, the macro invokes `handleFailureInMethod:object:file:lineNumber:description:` (page 145) on the assertion handler for the current thread, passing *desc* as the description string.

This macro should be used only within Objective-C methods.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined.

Availability

Available in Mac OS X v10.0 and later.

See Also

`NSLog` (page 2211)
`NSLogv` (page 2212)
`NSAssert1` (page 2160)
`NSCAssert` (page 2166)
`NSCParameterAssert` (page 2177)
`NSParameterAssert` (page 2225)

Related Sample Code

CocoaVideoFrameToGWorld
CocoaVideoFrameToNSImage
ColorMatching
SGDevices
SimpleThreads

Declared In

`NSException.h`

NSAssert1

Generates an assertion if a given condition is false.

```
#define NSAssert1(condition, desc, arg1)
```

Parameters

condition

An expression that evaluates to `YES` or `NO`.

desc

An NSString object that contains a printf-style string containing an error message describing the failure condition and a placeholder for a single argument.

arg1

An argument to be inserted, in place, into *desc*.

Discussion

The NSAssert1 macro evaluates the condition and serves as a front end to the assertion handler.

Each thread has its own assertion handler, which is an object of class NSAssertionHandler. When invoked, an assertion handler prints an error message that includes the method and class names (or the function name). It then raises an NSInternalInconsistencyException exception. If *condition* evaluates to NO, the macro invokes handleFailureInMethod:object:file:lineNumber:description: (page 145) on the assertion handler for the current thread, passing *desc* as the description string and *arg1* as a substitution variable.

This macro should be used only within Objective-C methods.

Assertions are disabled if the preprocessor macro NS_BLOCK_ASSERTIONS is defined.

Availability

Available in Mac OS X v10.0 and later.

See Also

NSLog (page 2211)
NSLogv (page 2212)
NSAssert (page 2159)
NSAssert2 (page 2161)
NSAssert3 (page 2162)
NSAssert4 (page 2163)
NSAssert5 (page 2165)
NSCAssert (page 2166)
NSCParameterAssert (page 2177)
NSParameterAssert (page 2225)

Related Sample Code

CocoaDVDPlayer
Core Data HTML Store

Declared In

NSException.h

NSAssert2

Generates an assertion if a given condition is false.

```
#define NSAssert2(condition, desc, arg1, arg2)
```

Parameters

condition

An expression that evaluates to YES or NO.

desc

An NSString object that contains a printf-style string containing an error message describing the failure condition and placeholders for two arguments.

arg1

An argument to be inserted, in place, into *desc*.

arg2

An argument to be inserted, in place, into *desc*.

Discussion
The NSAssert2 macro evaluates the condition and serves as a front end to the assertion handler.

Each thread has its own assertion handler, which is an object of class NSAssertionHandler. When invoked, an assertion handler prints an error message that includes the method and class names (or the function name). It then raises an NSInternalInconsistencyException exception. If *condition* evaluates to NO, the macro invokes handleFailureInMethod:object:file:lineNumber:description: (page 145) on the assertion handler for the current thread, passing *desc* as the description string and *arg1* and *arg2* as substitution variables.

This macro should be used only within Objective-C methods.

Assertions are disabled if the preprocessor macro NS_BLOCK_ASSERTIONS is defined.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSLog (page 2211)
NSLogv (page 2212)
NSAssert (page 2159)
NSAssert1 (page 2160)
NSAssert3 (page 2162)
NSAssert4 (page 2163)
NSAssert5 (page 2165)
NSCAssert (page 2166)
NSCParameterAssert (page 2177)
NSParameterAssert (page 2225)

Related Sample Code
CoreRecipes

Declared In
NSException.h

NSAssert3

Generates an assertion if a given condition is false.

```
#define NSAssert3(condition, desc, arg1, arg2, arg3)
```

Parameters

condition

An expression that evaluates to `YES` or `NO`.

desc

An `NSString` object that contains a `printf`-style string containing an error message describing the failure condition and placeholders for three arguments.

arg1

An argument to be inserted, in place, into *desc*.

arg2

An argument to be inserted, in place, into *desc*.

arg3

An argument to be inserted, in place, into *desc*.

Discussion

The `NSAssert3` macro evaluates the condition and serves as a front end to the assertion handler.

Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes the method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception. If *condition* evaluates to `NO`, the macro invokes `handleFailureInMethod:object:file:lineNumber:description:` (page 145) on the assertion handler for the current thread, passing *desc* as the description string and *arg1*, *arg2*, and *arg3* as substitution variables.

This macro should be used only within Objective-C methods.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined.

Availability

Available in Mac OS X v10.0 and later.

See Also

`NSLog` (page 2211)
`NSLogv` (page 2212)
`NSAssert` (page 2159)
`NSAssert1` (page 2160)
`NSAssert2` (page 2161)
`NSAssert4` (page 2163)
`NSAssert5` (page 2165)
`NSCAssert` (page 2166)
`NSCParameterAssert` (page 2177)
`NSParameterAssert` (page 2225)

Declared In

`NSException.h`

NSAssert4

Generates an assertion if a given condition is false.

```
#define NSAssert4(condition, desc, arg1, arg2, arg3, arg4)
```

Parameters

condition

An expression that evaluates to `YES` or `NO`.

desc

An `NSString` object that contains a `printf`-style string containing an error message describing the failure condition and placeholders for four arguments.

arg1

An argument to be inserted, in place, into *desc*.

arg2

An argument to be inserted, in place, into *desc*.

arg3

An argument to be inserted, in place, into *desc*.

arg4

An argument to be inserted, in place, into *desc*.

Discussion

The `NSAssert4` macro evaluates the condition and serves as a front end to the assertion handler.

Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes the method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception. If *condition* evaluates to `NO`, the macro invokes `handleFailureInMethod:object:file:lineNumber:description:` (page 145) on the assertion handler for the current thread, passing *desc* as the description string and *arg1*, *arg2*, *arg3*, and *arg4* as substitution variables.

This macro should be used only within Objective-C methods.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined.

Availability

Available in Mac OS X v10.0 and later.

See Also

`NSLog` (page 2211)
`NSLogv` (page 2212)
`NSAssert` (page 2159)
`NSAssert1` (page 2160)
`NSAssert2` (page 2161)
`NSAssert3` (page 2162)
`NSAssert5` (page 2165)
`NSCAssert` (page 2166)
`NSCParameterAssert` (page 2177)
`NSParameterAssert` (page 2225)

Declared In

`NSException.h`

NSAssert5

Generates an assertion if a given condition is false.

```
#define NSAssert5(condition, desc, arg1, arg2, arg3, arg4, arg5)
```

Parameters

condition

An expression that evaluates to `YES` or `NO`.

desc

An `NSString` object that contains a `printf`-style string containing an error message describing the failure condition and placeholders for five arguments.

arg1

An argument to be inserted, in place, into *desc*.

arg2

An argument to be inserted, in place, into *desc*.

arg3

An argument to be inserted, in place, into *desc*.

arg4

An argument to be inserted, in place, into *desc*.

arg5

An argument to be inserted, in place, into *desc*.

Discussion

The `NSAssert5` macro evaluates the condition and serves as a front end to the assertion handler.

Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes the method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception. If *condition* evaluates to `NO`, the macro invokes `handleFailureInMethod:object:file:lineNumber:description:` (page 145) on the assertion handler for the current thread, passing *desc* as the description string and *arg1*, *arg2*, *arg3*, *arg4*, and *arg5* as substitution variables.

This macro should be used only within Objective-C methods.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined.

Availability

Available in Mac OS X v10.0 and later.

See Also

`NSLog` (page 2211)
`NSLogv` (page 2212)
`NSAssert` (page 2159)
`NSAssert1` (page 2160)
`NSAssert2` (page 2161)
`NSAssert3` (page 2162)
`NSAssert4` (page 2163)
`NSCAssert` (page 2166)
`NSCParameterAssert` (page 2177)
`NSParameterAssert` (page 2225)

Declared In
NSException.h

NSCAssert

Generates an assertion if the given condition is false.

```
NSCAssert(condition, NSString *description)
```

Discussion
Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception.

The `NSCAssert` macro evaluates the condition and serves as a front end to the assertion handler. This macro should be used only within C functions. `NSCAssert` takes no arguments other than the condition and format string.

The *`condition`* must be an expression that evaluates to true or false. *`description`* is a printf-style format string that describes the failure condition.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined. All macros return `void`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSLog` (page 2211)
`NSLogv` (page 2212)
`NSAssert` (page 2159)
`NSCAssert1` (page 2166)
`NSCParameterAssert` (page 2177)
`NSParameterAssert` (page 2225)

Related Sample Code
EnhancedAudioBurn

Declared In
NSException.h

NSCAssert1

`NSCAssert1` is one of a series of macros that generate assertions if the given condition is false.

```
NSCAssert1(condition, NSString *description, arg1)
```

Discussion
Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception.

The `NSCAssert1` macro evaluates the condition and serves as a front end to the assertion handler. This macro should be used only within C functions.

The *`condition`* expression must evaluate to true or false. *`description`* is a printf-style format string that describes the failure condition. *`arg1`* is an argument to be inserted, in place, into the description.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined. All macros return `void`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSLog` (page 2211)
`NSLogv` (page 2212)
`NSCAssert` (page 2166)
`NSCAssert2` (page 2167)
`NSCAssert3` (page 2168)
`NSCAssert4` (page 2169)
`NSCAssert5` (page 2169)
`NSCParameterAssert` (page 2177)
`NSParameterAssert` (page 2225)

Declared In
`NSException.h`

NSCAssert2

`NSCAssert2` is one of a series of macros that generate assertions if the given condition is false.

```
NSCAssert2(condition, NSString *description, arg1, arg2)
```

Discussion
Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception.

The `NSCAssert2` macro evaluates the condition and serves as a front end to the assertion handler. This macro should be used only within C functions.

The *`condition`* expression must evaluate to true or false. *`description`* is a printf-style format string that describes the failure condition. Each *`argn`* is an argument to be inserted, in place, into the description.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined. All macros return `void`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSLog` (page 2211)
`NSLogv` (page 2212)
`NSCAssert` (page 2166)
`NSCAssert1` (page 2166)
`NSCAssert3` (page 2168)
`NSCAssert4` (page 2169)
`NSCAssert5` (page 2169)
`NSCParameterAssert` (page 2177)
`NSParameterAssert` (page 2225)

Declared In
`NSException.h`

NSCAssert3

`NSCAssert3` is one of a series of macros that generate assertions if the given condition is false.

```
NSCAssert3(condition, NSString *description, arg1, arg2, arg3)
```

Discussion
Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception.

The `NSCAssert3` macro evaluates the condition and serves as a front end to the assertion handler. This macro should be used only within C functions.

The *`condition`* expression must evaluate to true or false. *`description`* is a printf-style format string that describes the failure condition. Each *`argn`* is an argument to be inserted, in place, into the description.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined. All macros return `void`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSLog` (page 2211)
`NSLogv` (page 2212)
`NSCAssert` (page 2166)
`NSCAssert1` (page 2166)
`NSCAssert2` (page 2167)
`NSCAssert4` (page 2169)
`NSCAssert5` (page 2169)

NSCParameterAssert (page 2177)
NSParameterAssert (page 2225)

Declared In
NSException.h

NSCAssert4

NSCAssert4 is one of a series of macros that generate assertions if the given condition is false.

```
NSCAssert4(condition, NSString *description, arg1, arg2, arg3, arg4)
```

Discussion
Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class NSAssertionHandler. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an NSInternalInconsistencyException exception.

The NSCAssert4 macro evaluates the condition and serves as a front end to the assertion handler. This macro should be used only within C functions.

The *condition* expression must evaluate to true or false. *description* is a printf-style format string that describes the failure condition. Each *argn* is an argument to be inserted, in place, into the description.

Assertions are disabled if the preprocessor macro NS_BLOCK_ASSERTIONS is defined. All macros return void.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSLog (page 2211)
NSLogv (page 2212)
NSCAssert (page 2166)
NSCAssert1 (page 2166)
NSCAssert2 (page 2167)
NSCAssert3 (page 2168)
NSCAssert5 (page 2169)
NSCParameterAssert (page 2177)
NSParameterAssert (page 2225)

Declared In
NSException.h

NSCAssert5

NSCAssert5 is one of a series of macros that generate assertions if the given condition is false.

```
NSCAssert5(condition, NSString *description, arg1, arg2, arg3, arg4, arg5)
```

Discussion
Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception.

The `NSCAssert5` macro evaluates the condition and serves as a front end to the assertion handler. This macro should be used only within C functions.

The *`condition`* expression must evaluate to true or false. *`description`* is a printf-style format string that describes the failure condition. Each *`argn`* is an argument to be inserted, in place, into the description.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined. All macros return `void`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSLog` (page 2211)
`NSLogv` (page 2212)
`NSCAssert` (page 2166)
`NSCAssert1` (page 2166)
`NSCAssert2` (page 2167)
`NSCAssert3` (page 2168)
`NSCAssert4` (page 2169)
`NSCParameterAssert` (page 2177)
`NSParameterAssert` (page 2225)

Declared In
`NSException.h`

NSClassFromString

Obtains a class by name.

```
Class NSClassFromString (
   NSString *aClassName
);
```

Parameters
`aClassName`
The name of a class.

Return Value
The class object named by *`aClassName`*, or `nil` if no class by that name is currently loaded. If *`aClassName`* is `nil`, returns `nil`.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSStringFromClass (page 2238)
NSProtocolFromString (page 2228)
NSSelectorFromString (page 2234)

Related Sample Code
Sketch-112

Declared In
NSObjCRuntime.h

NSCompareHashTables

Returns a Boolean value that indicates whether the elements of two hash tables are equal.

```
BOOL NSCompareHashTables (
   NSHashTable *table1,
   NSHashTable *table2
);
```

Return Value
YES if the two hash tables are equal—that is, if each element of *table1* is in *table2* and the two tables are the same size, otherwise NO.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSCreateHashTable (page 2177)
NSCreateHashTableWithZone (page 2178)

Declared In
NSHashTable.h

NSCompareMapTables

Compares the elements of two map tables for equality.

```
BOOL NSCompareMapTables (
   NSMapTable *table1,
   NSMapTable *table2
);
```

Return Value
YES if each key of *table1* is in *table2*, and the two tables are the same size, otherwise NO.

Discussion
Note that this function does not compare values, only keys.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSCreateMapTable (page 2178)
NSCreateMapTableWithZone (page 2179)

Declared In
NSMapTable.h

NSContainsRect

Returns a Boolean value that indicates whether one rectangle completely encloses another.

```
BOOL NSContainsRect (
   NSRect aRect,
   NSRect bRect
);
```

Return Value
YES if *aRect* completely encloses *bRect*. For this condition to be true, *bRect* cannot be empty, and must not extend beyond *aRect* in any direction.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSGeometry.h

NSConvertHostDoubleToSwapped

Performs a type conversion.

```
NSSwappedDouble NSConvertHostDoubleToSwapped (
   double x
);
```

Discussion
Converts the double value in *x* to a value whose bytes can be swapped. This function does not actually swap the bytes of *x*. You should not need to call this function directly.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSwapHostDoubleToBig (page 2246)
NSSwapHostDoubleToLittle (page 2246)

Declared In
NSByteOrder.h

NSConvertHostFloatToSwapped

Performs a type conversion.

```
NSSwappedFloat NSConvertHostFloatToSwapped (
   float x
);
```

Discussion
Converts the float value in *x* to a value whose bytes can be swapped. This function does not actually swap the bytes of *x*. You should not need to call this function directly.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSSwapHostFloatToBig` (page 2246)
`NSSwapHostFloatToLittle` (page 2247)

Declared In
`NSByteOrder.h`

NSConvertSwappedDoubleToHost

Performs a type conversion.

```
double NSConvertSwappedDoubleToHost (
   NSSwappedDouble x
);
```

Discussion
Converts the value in *x* to a double value. This function does not actually swap the bytes of *x*. You should not need to call this function directly.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSSwapBigDoubleToHost` (page 2242)
`NSSwapLittleDoubleToHost` (page 2251)

Declared In
`NSByteOrder.h`

NSConvertSwappedFloatToHost

Performs a type conversion.

```
float NSConvertSwappedFloatToHost (
   NSSwappedFloat x
);
```

Discussion
Converts the value in *x* to a float value. This function does not actually swap the bytes of *x*. You should not need to call this function directly.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSwapBigFloatToHost (page 2243)
NSSwapLittleFloatToHost (page 2252)

Declared In
NSByteOrder.h

NSCopyHashTableWithZone

Performs a shallow copy of the specified hash table.

```
NSHashTable * NSCopyHashTableWithZone (
   NSHashTable *table,
   NSZone *zone
);
```

Return Value
A pointer to a new copy of *table*, created in *zone* and containing pointers to the data elements of *table*.

Discussion
If *zone* is NULL, the new table is created in the default zone.

The new table adopts the callback functions of *table* and calls the hash and retain callback functions as appropriate when inserting elements into the new table.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSCreateHashTable (page 2177)
NSCreateHashTableWithZone (page 2178)
NSHashTableCallBacks (page 2270) (structure)

Declared In
NSHashTable.h

NSCopyMapTableWithZone

Performs a shallow copy of the specified map table.

```
NSMapTable * NSCopyMapTableWithZone (
   NSMapTable *table,
   NSZone *zone
);
```

Return Value
A pointer to a new copy of *table*, created in *zone* and containing pointers to the keys and values of *table*.

Discussion
If *zone* is NULL, the new table is created in the default zone.

The new table adopts the callback functions of *table* and calls the hash and retain callback functions as appropriate when inserting elements into the new table.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSCreateMapTable (page 2178)
NSCreateMapTableWithZone (page 2179)
NSMapTableKeyCallBacks (page 2272) (structure)
NSMapTableValueCallBacks (page 2273) (structure)

Declared In
NSMapTable.h

NSCopyMemoryPages

Copies a block of memory.

```
void NSCopyMemoryPages (
   const void *source,
   void *dest,
   NSUInteger bytes
);
```

Discussion
Copies (or copies on write) *byteCount* bytes from *source* to *destination*.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSAllocateMemoryPages (page 2158)
NSDeallocateMemoryPages (page 2180)

Declared In
NSZone.h

NSCopyObject

Creates an exact copy of an object.

```
id NSCopyObject (
   id object,
   NSUInteger extraBytes,
   NSZone *zone
);
```

Parameters

object
The object to copy.

extraBytes
The number of extra bytes required for indexed instance variables (this value is typically 0).

zone
The zone in which to create the new instance (pass NULL to specify the default zone).

Return Value
A new object that's an exact copy of *anObject*, or `nil` if *object* is `nil` or if *object* could not be copied.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSAllocateObject` (page 2159)
`NSDeallocateObject` (page 2181)

Declared In
`NSObject.h`

NSCountHashTable

Returns the number of elements in a hash table.

```
NSUInteger NSCountHashTable (
   NSHashTable *table
);
```

Return Value
The number of elements currently in *table*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSHashTable.h`

NSCountMapTable

Returns the number of elements in a map table.

```
NSUInteger NSCountMapTable (
   NSMapTable *table
);
```

Parameters
table
A reference to a map table structure.

Return Value
The number of key-value pairs currently in *table*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSMapTable.h`

NSCParameterAssert

Evaluates the specified parameter.

```
NSCParameterAssert(condition)
```

Discussion
Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception.

This macro validates a parameter for a C function. Simply provide the parameter as the condition argument. The macro evaluates the parameter and, if the parameter evaluates to false, logs an error message that includes the parameter and then raises an exception.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined. All macros return `void`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSLog` (page 2211)
`NSLogv` (page 2212)
`NSAssert` (page 2159)
`NSCAssert` (page 2166)
`NSParameterAssert` (page 2225)

Declared In
`NSException.h`

NSCreateHashTable

Creates and returns a new hash table.

```
NSHashTable * NSCreateHashTable (
   NSHashTableCallBacks callBacks,
   NSUInteger capacity
);
```

Return Value
A pointer to an NSHashTable created in the default zone.

Discussion
The table's size is dependent on (but generally not equal to) *`capacity`*. If *`capacity`* is 0, a small hash table is created. The `NSHashTableCallBacks` (page 2270) structure *`callBacks`* has five pointers to functions, with the following defaults: pointer hashing, if `hash` is `NULL`; pointer equality, if `isEqual` is `NULL`; no callback upon adding an element, if `retain` is `NULL`; no callback upon removing an element, if `release` is `NULL`; and a function returning a pointer's hexadecimal value as a string, if `describe` is `NULL`. The hashing function must be defined such that if two data elements are equal, as defined by the comparison function, the values produced by hashing on these elements must also be equal. Also, data elements must remain invariant if the value of the hashing function depends on them; for example, if the hashing function operates directly on the characters of a string, that string can't change.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSCopyHashTableWithZone (page 2174)
NSCreateHashTableWithZone (page 2178)

Declared In
NSHashTable.h

NSCreateHashTableWithZone

Creates a new hash table in a given zone.

```
NSHashTable * NSCreateHashTableWithZone (
   NSHashTableCallBacks callBacks,
   NSUInteger capacity,
   NSZone *zone
);
```

Return Value
A pointer to a new hash table created in the specified zone. If *zone* is NULL, the hash table is created in the default zone.

Discussion
The table's size is dependent on (but generally not equal to) *capacity*. If *capacity* is 0, a small hash table is created. The NSHashTableCallBacks (page 2270) structure *callBacks* has five pointers to functions, with the following defaults: pointer hashing, if hash is NULL; pointer equality, if isEqual is NULL; no callback upon adding an element, if retain is NULL; no callback upon removing an element, if release is NULL; and a function returning a pointer's hexadecimal value as a string, if describe is NULL. The hashing function must be defined such that if two data elements are equal, as defined by the comparison function, the values produced by hashing on these elements must also be equal. Also, data elements must remain invariant if the value of the hashing function depends on them; for example, if the hashing function operates directly on the characters of a string, that string can't change.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSCreateHashTable (page 2177)

Declared In
NSHashTable.h

NSCreateMapTable

Creates a new map table in the default zone.

```
NSMapTable * NSCreateMapTable (
   NSMapTableKeyCallBacks keyCallBacks,
   NSMapTableValueCallBacks valueCallBacks,
   NSUInteger capacity
);
```

Discussion
Creates and returns a pointer to an `NSMapTable` structure in the default zone; the table's size is dependent on (but generally not equal to) *`capacity`*. If *`capacity`* is 0, a small map table is created. The `NSMapTableKeyCallBacks` (page 2272) arguments are structures that are very similar to the callback structure used by `NSCreateHashTable` (page 2177)—they have the same defaults as documented for that function.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSCopyMapTableWithZone` (page 2174)
`NSCreateMapTableWithZone` (page 2179)

Declared In
`NSMapTable.h`

NSCreateMapTableWithZone

Creates a new map table in the specified zone.

```
NSMapTable * NSCreateMapTableWithZone (
   NSMapTableKeyCallBacks keyCallBacks,
   NSMapTableValueCallBacks valueCallBacks,
   NSUInteger capacity,
   NSZone *zone
);
```

Return Value
A new map table in allocated in *`zone`*. If *`zone`* is `NULL`, the hash table is created in the default zone.

Discussion
The table's size is dependent on (but generally not equal to) *`capacity`*. If *`capacity`* is 0, a small map table is created. The `NSMapTableKeyCallBacks` (page 2272) arguments are structures that are very similar to the callback structure used by `NSCreateHashTable` (page 2177); in fact, they have the same defaults as documented for that function.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSCopyMapTableWithZone` (page 2174)
`NSCreateMapTable` (page 2178)

Declared In
`NSMapTable.h`

NSCreateZone

Creates a new zone.

```
NSZone * NSCreateZone (
   NSUInteger startSize,
   NSUInteger granularity,
   BOOL canFree
);
```

Return Value
A pointer to a new zone of *startSize* bytes, which will grow and shrink by *granularity* bytes. If *canFree* is 0, the allocator will never free memory, and `malloc` will be fast. Returns `NULL` if a new zone could not be created.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSDefaultMallocZone` (page 2188)
`NSRecycleZone` (page 2232)
`NSSetZoneName` (page 2236)

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSZone.h`

NSDeallocateMemoryPages

Deallocates the specified block of memory.

```
void NSDeallocateMemoryPages (
   void *ptr,
   NSUInteger bytes
);
```

Discussion
This function deallocates memory that was allocated with `NSAllocateMemoryPages`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSCopyMemoryPages` (page 2175)
`NSAllocateMemoryPages` (page 2158)

Declared In
`NSZone.h`

NSDeallocateObject

Destroys an existing object.

```
void NSDeallocateObject (
   id object
);
```

Parameters

object

An object.

Discussion

This function deallocates *object*, which must have been allocated using `NSAllocateObject`.

Availability

Available in Mac OS X v10.0 and later.

See Also

`NSCopyObject` (page 2175)

`NSAllocateObject` (page 2159)

Declared In

`NSObject.h`

NSDecimalAdd

Adds two decimal values.

```
NSCalculationError NSDecimalAdd (
   NSDecimal *result,
   const NSDecimal *leftOperand,
   const NSDecimal *rightOperand,
   NSRoundingMode roundingMode
);
```

Discussion

Adds *leftOperand* to *rightOperand* and stores the sum in *result*.

An NSDecimal can represent a number with up to 38 significant digits. If a number is more precise than that, it must be rounded off. *roundingMode* determines how to round it off. There are four possible rounding modes:

`NSRoundDown`	Round return values down.
`NSRoundUp`	Round return values up.
`NSRoundPlain`	Round to the closest possible return value; when caught halfway between two positive numbers, round up; when caught between two negative numbers, round down.
`NSRoundBankers`	Round to the closest possible return value; when halfway between two possibilities, return the possibility whose last digit is even.

The return value indicates whether any machine limitations were encountered in the addition. If none were encountered, the function returns `NSCalculationNoError`. Otherwise it may return one of the following values: `NSCalculationLossOfPrecision`, `NSCalculationOverflow` or `NSCalculationUnderflow`. For descriptions of all these error conditions, see `exceptionDuringOperation:error:leftOperand:rightOperand:` (page 2044) in NSDecimalNumberBehaviors.

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSDecimal.h`

NSDecimalCompact

Compacts the decimal structure for efficiency.

```
void NSDecimalCompact (
   NSDecimal *number
);
```

Discussion
Formats number so that calculations using it will take up as little memory as possible. All the `NSDecimal...` arithmetic functions expect compact NSDecimal arguments.

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSDecimal.h`

NSDecimalCompare

Compares two decimal values.

```
NSComparisonResult NSDecimalCompare (
   const NSDecimal *leftOperand,
   const NSDecimal *rightOperand
);
```

Return Value
`NSOrderedDescending` if *`leftOperand`* is bigger than *`rightOperand`*; `NSOrderedAscending` if *`rightOperand`* is bigger than *`leftOperand`*; or `NSOrderedSame` if the two operands are equal.

Discussion
For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSDecimal.h

NSDecimalCopy

Copies the value of a decimal number.

```
void NSDecimalCopy (
   NSDecimal *destination,
   const NSDecimal *source
);
```

Discussion
Copies the value in *source* to *destination*.

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSDecimal.h

NSDecimalDivide

Divides one decimal value by another.

```
NSCalculationError NSDecimalDivide (
   NSDecimal *result,
   const NSDecimal *leftOperand,
   const NSDecimal *rightOperand,
   NSRoundingMode roundingMode
);
```

Discussion
Divides *leftOperand* by *rightOperand* and stores the quotient, possibly rounded off according to *roundingMode*, in *result*. If *rightOperand* is 0, returns NSDivideByZero.

For explanations of the possible return values and rounding modes, see NSDecimalAdd (page 2181).

Note that repeating decimals or numbers with a mantissa larger than 38 digits cannot be represented precisely.

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSDecimal.h

NSDecimalIsNotANumber

Returns a Boolean that indicates whether a given decimal contains a valid number.

```
BOOL NSDecimalIsNotANumber (
   const NSDecimal *dcm
);
```

Return Value
`YES` if the value in *`decimal`* represents a valid number, otherwise `NO`.

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSDecimal.h`

NSDecimalMultiply

Multiplies two decimal numbers together.

```
NSCalculationError NSDecimalMultiply (
   NSDecimal *result,
   const NSDecimal *leftOperand,
   const NSDecimal *rightOperand,
   NSRoundingMode roundingMode
);
```

Discussion
Multiplies *`rightOperand`* by *`leftOperand`* and stores the product, possibly rounded off according to *`roundingMode`*, in *`result`*.

For explanations of the possible return values and rounding modes, see `NSDecimalAdd` (page 2181).

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSDecimal.h`

NSDecimalMultiplyByPowerOf10

Multiplies a decimal by the specified power of 10.

```
NSCalculationError NSDecimalMultiplyByPowerOf10 (
   NSDecimal *result,
   const NSDecimal *number,
   short power,
   NSRoundingMode roundingMode
);
```

Discussion
Multiplies *`number`* by *`power`* of 10 and stores the product, possibly rounded off according to *`roundingMode`*, in *`result`*.

For explanations of the possible return values and rounding modes, see `NSDecimalAdd` (page 2181).

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSDecimal.h`

NSDecimalNormalize

Normalizes the internal format of two decimal numbers to simplify later operations.

```
NSCalculationError NSDecimalNormalize (
   NSDecimal *number1,
   NSDecimal *number2,
   NSRoundingMode roundingMode
);
```

Discussion
An NSDecimal is represented in memory as a mantissa and an exponent, expressing the value mantissa x 10exponent. A number can have many NSDecimal representations; for example, the following table lists several valid NSDecimal representations for the number 100:

Mantissa	Exponent
100	0
10	1
1	2

Format *number1* and *number2* so that they have equal exponents. This format makes addition and subtraction very convenient. Both `NSDecimalAdd` (page 2181) and `NSDecimalSubtract` (page 2187) call `NSDecimalNormalize`. You may want to use it if you write more complicated addition or subtraction routines.

For explanations of the possible return values, see `NSDecimalAdd` (page 2181).

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSDecimal.h`

NSDecimalPower

Raises the decimal value to the specified power.

```
NSCalculationError NSDecimalPower (
   NSDecimal *result,
   const NSDecimal *number,
   NSUInteger power,
   NSRoundingMode roundingMode
);
```

Discussion
Raises *number* to *power*, possibly rounding off according to *roundingMode*, and stores the resulting value in *result*.

For explanations of the possible return values and rounding modes, see `NSDecimalAdd` (page 2181).

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSDecimal.h`

NSDecimalRound

Rounds off the decimal value.

```
void NSDecimalRound (
   NSDecimal *result,
   const NSDecimal *number,
   NSInteger scale,
   NSRoundingMode roundingMode
);
```

Discussion
Rounds *number* off according to the parameters *scale* and *roundingMode* and stores the result in *result*.

The *scale* value specifies the number of digits *result* can have after its decimal point. *roundingMode* specifies the way that number is rounded off. There are four possible values for *roundingMode*: `NSRoundDown`, `NSRoundUp`, `NSRoundPlain`, and `NSRoundBankers`. For thorough discussions of *scale* and *roundingMode*, see `NSDecimalNumberBehaviors`.

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSDecimal.h`

NSDecimalString

Returns a string representation of the decimal value.

```
NSString * NSDecimalString (
   const NSDecimal *dcm,
   id locale
);
```

Discussion
Returns a string representation of *decimal*. *locale* determines the format of the decimal separator.

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSDecimal.h

NSDecimalSubtract

Subtracts one decimal value from another.

```
NSCalculationError NSDecimalSubtract (
   NSDecimal *result,
   const NSDecimal *leftOperand,
   const NSDecimal *rightOperand,
   NSRoundingMode roundingMode
);
```

Discussion
Subtracts *rightOperand* from *leftOperand* and stores the difference, possibly rounded off according to *roundingMode*, in *result*.

For explanations of the possible return values and rounding modes, see NSDecimalAdd (page 2181).

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSDecimal.h

NSDecrementExtraRefCountWasZero

Decrements the specified object's reference count.

```
BOOL NSDecrementExtraRefCountWasZero (
   id object
);
```

Parameters
object
An object.

Return Value
NO if *anObject* had an extra reference count, or YES if *anObject* didn't have an extra reference count—indicating that the object should be deallocated (with dealloc).

Discussion
Decrements the "extra reference" count of *anObject*. Newly created objects have only one actual reference, so that a single release message results in the object being deallocated. Extra references are those beyond the single original reference and are usually created by sending the object a retain message. Your code should generally not use these functions unless it is overriding the retain (page 2108) or release (page 2106) methods.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSExtraRefCount (page 2193)
NSIncrementExtraRefCount (page 2201)

Declared In
NSObject.h

NSDefaultMallocZone

Returns the default zone.

```
NSZone * NSDefaultMallocZone (void);
```

Return Value
The default zone, which is created automatically at startup.

Discussion
This zone is used by the standard C function malloc.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSCreateZone (page 2180)

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
NSZone.h

NSDivideRect

Divides a rectangle into two new rectangles.

```
void NSDivideRect (
   NSRect inRect,
   NSRect *slice,
   NSRect *rem,
   CGFloat amount,
   NSRectEdge edge
);
```

Discussion
Creates two rectangles—*slice* and *rem*—from *inRect*, by dividing *inRect* with a line that's parallel to the side of *inRect* specified by *edge*. The size of *slice* is determined by *amount*, which specifies the distance from *edge*.

slice and *rem* must not be `NULL`.

For more information, see `NSRectEdge` (page 2277).

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSInsetRect` (page 2201)
`NSIntegralRect` (page 2202)
`NSOffsetRect` (page 2224)

Related Sample Code
EnhancedDataBurn
ImageBackground
QTKitMovieShuffler
QTSSInspector
TrackBall

Declared In
`NSGeometry.h`

NSEndHashTableEnumeration

Used when finished with an enumerator.

```
void NSEndHashTableEnumeration (
   NSHashEnumerator *enumerator
);
```

Discussion
This function should be called when you have finished using the enumeration struct *enumerator*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSHashTable.h`

NSEndMapTableEnumeration

Used when finished with an enumerator.

```
void NSEndMapTableEnumeration (
   NSMapEnumerator *enumerator
);
```

Discussion
This function should be called when you have finished using the enumeration struct *enumerator*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSMapTable.h

NSEnumerateHashTable

Creates an enumerator for the specified hash table.

```
NSHashEnumerator NSEnumerateHashTable (
   NSHashTable *table
);
```

Return Value
An NSHashEnumerator structure that will cause successive elements of *table* to be returned each time this enumerator is passed to NSNextHashEnumeratorItem.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSNextHashEnumeratorItem (page 2223)

Declared In
NSHashTable.h

NSEnumerateMapTable

Creates an enumerator for the specified map table.

```
NSMapEnumerator NSEnumerateMapTable (
   NSMapTable *table
);
```

Parameters

table
A reference to a map table structure.

Return Value
An NSMapEnumerator structure that will cause successive key-value pairs of *table* to be visited each time this enumerator is passed to NSNextMapEnumeratorPair.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSNextMapEnumeratorPair` (page 2223)
`NSMapMember` (page 2218)
`NSMapGet` (page 2215)
`NSAllMapTableKeys` (page 2157)
`NSAllMapTableValues` (page 2158)

Declared In
`NSMapTable.h`

NSEqualPoints

Returns a Boolean value that indicates whether two points are equal.

```
BOOL NSEqualPoints (
   NSPoint aPoint,
   NSPoint bPoint
);
```

Return Value
`YES` if the two points *`aPoint`* and *`bPoint`* are identical, otherwise `NO`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
DragItemAround
GLChildWindowDemo
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus

Declared In
`NSGeometry.h`

NSEqualRanges

Returns a Boolean value that indicates whether two given ranges are equal.

```
BOOL NSEqualRanges (
   NSRange range1,
   NSRange range2
);
```

Return Value
`YES` if *`range1`* and *`range2`* have the same locations and lengths.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSRange.h

NSEqualRects

Returns a Boolean value that indicates whether the two rectangles are equal.

```
BOOL NSEqualRects (
   NSRect aRect,
   NSRect bRect
);
```

Return Value
YES if *aRect* and *bRect* are identical, otherwise NO.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
iSpend
JSPong
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus

Declared In
NSGeometry.h

NSEqualSizes

Returns a Boolean that indicates whether two size values are equal.

```
BOOL NSEqualSizes (
   NSSize aSize,
   NSSize bSize
);
```

Return Value
YES if *aSize* and *bSize* are identical, otherwise NO.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Aperture Edit Plugin - Borders & Titles
Quartz Composer QCTV
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus

Declared In
NSGeometry.h

NSExtraRefCount

Returns the specified object's reference count.

```
NSUInteger NSExtraRefCount (
   id object
);
```

Parameters

object
An object.

Return Value

The current reference count of *object*.

Discussion

This function is used in conjunction with `NSIncrementExtraRefCount` (page 2201) and `NSDecrementExtraRefCountWasZero` (page 2187) in situations where you need to override an object's `retain` (page 2108) and `release` (page 2106) methods.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSObject.h`

NSFileTypeForHFSTypeCode

Returns a string encoding a file type code.

```
NSString * NSFileTypeForHFSTypeCode (
   OSType hfsFileTypeCode
);
```

Parameters

hfsFileTypeCode
An HFS file type code.

Return Value

A string that encodes *hfsFileTypeCode*.

Discussion

For more information, see HFS File Types.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSHFSFileTypes.h`

NSFreeHashTable

Deletes the specified hash table.

```
void NSFreeHashTable (
   NSHashTable *table
);
```

Discussion
Releases each element of the specified hash table and frees the table itself.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSResetHashTable (page 2232)

Declared In
NSHashTable.h

NSFreeMapTable

Deletes the specified map table.

```
void NSFreeMapTable (
   NSMapTable *table
);
```

Parameters

table
A reference to a map table structure.

Discussion
Releases each key and value of the specified map table and frees the table itself.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSResetMapTable (page 2232)

Declared In
NSMapTable.h

NSFullUserName

Returns a string containing the full name of the current user.

```
NSString * NSFullUserName (void);
```

Return Value
A string containing the full name of the current user.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSUserName (page 2257)

Declared In
NSPathUtilities.h

NSGetSizeAndAlignment

Obtains the actual size and the aligned size of an encoded type.

```
const char * NSGetSizeAndAlignment (
   const char *typePtr,
   NSUInteger *sizep,
   NSUInteger *alignp
);
```

Discussion
Obtains the actual size and the aligned size of the first data type represented by *typePtr* and returns a pointer to the position of the next data type in *typePtr*. You can specify NULL for either *sizep* or *alignp* to ignore the corresponding information.

The value returned in *alignp* is the aligned size of the data type; for example, on some platforms, the aligned size of a char might be 2 bytes while the actual physical size is 1 byte.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSObjCRuntime.h

NSGetUncaughtExceptionHandler

Returns the top-level error handler.

```
NSUncaughtExceptionHandler * NSGetUncaughtExceptionHandler (void);
```

Return Value
A pointer to the top-level error-handling function where you can perform last-minute logging before the program terminates.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSetUncaughtExceptionHandler (page 2235)

Declared In
NSException.h

NSHashGet

Returns an element of the hash table.

```
void * NSHashGet (
   NSHashTable *table,
   const void *pointer
);
```

Return Value
The pointer in the table that matches *pointer* (as defined by the `isEqual` callback function). If there is no matching element, returns `NULL`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSHashTable.h`

NSHashInsert

Adds an element to the specified hash table.

```
void NSHashInsert (
   NSHashTable *table,
   const void *pointer
);
```

Discussion
Inserts *pointer*, which must not be `NULL`, into *table*. If *pointer* matches an item already in the table, the previous pointer is released using the `release` callback function that was specified when the table was created.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSHashRemove` (page 2197)
`NSHashInsertKnownAbsent` (page 2197)
`NSHashInsertIfAbsent` (page 2196)

Declared In
`NSHashTable.h`

NSHashInsertIfAbsent

Adds an element to the specified hash table only if the table does not already contain the element.

```
void * NSHashInsertIfAbsent (
   NSHashTable *table,
   const void *pointer
);
```

Return Value
If *pointer* matches an item already in *table*, returns the preexisting pointer; otherwise, *pointer* is added to the *table* and returns `NULL`.

Discussion
You must not specify `NULL` for *pointer*.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSHashRemove` (page 2197)
`NSHashInsert` (page 2196)
`NSHashInsertKnownAbsent` (page 2197)

Declared In
`NSHashTable.h`

NSHashInsertKnownAbsent

Adds an element to the specified hash table.

```
void NSHashInsertKnownAbsent (
   NSHashTable *table,
   const void *pointer
);
```

Discussion
Inserts *pointer*, which must not be `NULL`, into *table*. Unlike `NSHashInsert`, this function raises `NSInvalidArgumentException` if *table* already includes an element that matches *pointer*.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSHashRemove` (page 2197)
`NSHashInsert` (page 2196)
`NSHashInsertIfAbsent` (page 2196)

Declared In
`NSHashTable.h`

NSHashRemove

Removes an element from the specified hash table.

```
void NSHashRemove (
   NSHashTable *table,
   const void *pointer
);
```

Discussion
If *pointer* matches an item already in *table*, this function releases the preexisting item.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSHashInsert (page 2196)
NSHashInsertKnownAbsent (page 2197)
NSHashInsertIfAbsent (page 2196)

Declared In
NSHashTable.h

NSHeight

Returns the height of a given rectangle.

```
CGFloat NSHeight (
   NSRect aRect
);
```

Return Value
The height of *aRect*.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSMaxX (page 2219)
NSMaxY (page 2220)
NSMidX (page 2220)
NSMidY (page 2221)
NSMinX (page 2221)
NSMinY (page 2222)
NSWidth (page 2257)

Related Sample Code
Clock Control
CocoaVideoFrameToGWorld
iSpend
OpenGLCompositorLab
WebKitDOMElementPlugIn

Declared In
NSGeometry.h

NSHFSTypeCodeFromFileType

Returns a file type code.

```
OSType NSHFSTypeCodeFromFileType (
   NSString *fileTypeString
);
```

Parameters

fileTypeString

A string of the sort encoded by `NSFileTypeForHFSTypeCode()`.

Return Value
The HFS file type code corresponding to *fileTypeString*, or `0` if it cannot be found.

Discussion
For more information, see HFS File Types.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSHFSFileTypes.h`

NSHFSTypeOfFile

Returns a string encoding a file type.

```
NSString * NSHFSTypeOfFile (
   NSString *fullFilePath
);
```

Parameters

fullFilePath

The full absolute path of a file.

Return Value
A string that encodes *fullFilePath*'s HFS file type, or `nil` if the operation was not successful

Discussion
For more information, see HFS File Types.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
DeskPictAppDockMenu

Declared In
`NSHFSFileTypes.h`

NSHomeDirectory

Returns the path to the current user's home directory.

```
NSString * NSHomeDirectory (void);
```

Return Value
The path to the current user's home directory.

Discussion
For more information on file-system utilities, see *Low-Level File Management Programming Topics*.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSFullUserName` (page 2194)
`NSUserName` (page 2257)
`NSHomeDirectoryForUser` (page 2200)

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSPathUtilities.h`

NSHomeDirectoryForUser

Returns the path to a given user's home directory.

```
NSString * NSHomeDirectoryForUser (
   NSString *userName
);
```

Parameters

userName
The name of a user.

Return Value
The path to the home directory for the user specified by *userName*.

Discussion
For more information on file system utilities, see *Low-Level File Management Programming Topics*.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSFullUserName` (page 2194)
`NSUserName` (page 2257)
`NSHomeDirectory` (page 2199)

Declared In
`NSPathUtilities.h`

NSHostByteOrder

Returns the endian format.

```
long NSHostByteOrder (void);
```

Return Value
The endian format, either `NS_LittleEndian` or `NS_BigEndian`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSByteOrder.h`

NSIncrementExtraRefCount

Increments the specified object's reference count.

```
void NSIncrementExtraRefCount (
   id object
);
```

Parameters

object
An object.

Discussion
This function increments the "extra reference" count of *object*. Newly created objects have only one actual reference, so that a single release message results in the object being deallocated. Extra references are those beyond the single original reference and are usually created by sending the object a retain message. Your code should generally not use these functions unless it is overriding the retain or release methods.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSExtraRefCount` (page 2193)
`NSDecrementExtraRefCountWasZero` (page 2187)

Declared In
`NSObject.h`

NSInsetRect

Insets a rectangle by a specified amount.

```
NSRect NSInsetRect (
   NSRect aRect,
   CGFloat dX,
   CGFloat dY
);
```

Return Value
A copy of *aRect*, altered by moving the two sides that are parallel to the y axis inward by *dX*, and the two sides parallel to the x axis inwards by *dY*.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSDivideRect (page 2188)
NSIntegralRect (page 2202)
NSOffsetRect (page 2224)

Related Sample Code
Dicey
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus
WebKitDOMElementPlugIn

Declared In
NSGeometry.h

NSIntegralRect

Adjusts the sides of a rectangle to integer values.

```
NSRect NSIntegralRect (
   NSRect aRect
);
```

Return Value
A copy of *aRect*, expanded outward just enough to ensure that none of its four defining values (x, y, width, and height) have fractional parts. If the width or height of *aRect* is 0 or negative, this function returns a rectangle with origin at (0.0, 0.0) and with zero width and height.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSDivideRect (page 2188)
NSInsetRect (page 2201)
NSOffsetRect (page 2224)

Related Sample Code
CIAnnotation
FilterDemo
PDF Annotation Editor
PDFKitLinker2
Worm

Declared In
NSGeometry.h

NSIntersectionRange

Returns the intersection of the specified ranges.

```
NSRange NSIntersectionRange (
   NSRange range1,
   NSRange range2
);
```

Return Value
A range describing the intersection of *range1* and *range2*—that is, a range containing the indices that exist in both ranges.

Discussion
If the returned range's length field is 0, then the two ranges don't intersect, and the value of the location field is undefined.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSUnionRange (page 2256)

Related Sample Code
LayoutManagerDemo

Declared In
NSRange.h

NSIntersectionRect

Calculates the intersection of two rectangles.

```
NSRect NSIntersectionRect (
   NSRect aRect,
   NSRect bRect
);
```

Return Value
The graphic intersection of *aRect* and *bRect*. If the two rectangles don't overlap, the returned rectangle has its origin at (0.0, 0.0) and zero width and height (including situations where the intersection is a point or a line segment).

Availability
Available in Mac OS X v10.0 and later.

See Also
NSUnionRect (page 2256)

Related Sample Code
Cropped Image
FilterDemo
Link Snoop
Sketch-112
TextLinks

Declared In
NSGeometry.h

NSIntersectsRect

Returns a Boolean value that indicates whether two rectangles intersect.

```
BOOL NSIntersectsRect (
   NSRect aRect,
   NSRect bRect
);
```

Return Value
YES if *aRect* intersects *bRect*, otherwise NO. Returns NO if either *aRect* or *bRect* has a width or height that is 0.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSIntersectionRect (page 2203)

Related Sample Code
JSPong
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus
Worm

Declared In
NSGeometry.h

NSIsEmptyRect

Returns a Boolean value that indicates whether a given rectangle is empty.

```
BOOL NSIsEmptyRect (
   NSRect aRect
);
```

Return Value
YES if *aRect* encloses no area at all—that is, if its width or height is 0 or negative, otherwise NO.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CIVideoDemoGL
Cropped Image
Dicey
IBFragmentView
Sketch-112

Declared In
NSGeometry.h

NSJavaBundleCleanup

This function has been deprecated. (Deprecated in Mac OS X v10.5.)

```
void NSJavaBundleCleanup (
   NSBundle *bundle,
   NSDictionary *plist
);
```

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.5.

Declared In
NSJavaSetup.h

NSJavaBundleSetup

This function has been deprecated. (Deprecated in Mac OS X v10.5.)

```
id NSJavaBundleSetup (
   NSBundle *bundle,
   NSDictionary *plist
);
```

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.5.

Declared In
NSJavaSetup.h

NSJavaClassesForBundle

Loads the Java classes located in the specified bundle. (Deprecated in Mac OS X v10.5.)

```
NSArray * NSJavaClassesForBundle (
   NSBundle *bundle,
   BOOL usesyscl,
   id *vm
);
```

Discussion
Loads and returns the Java classes in the specified bundle. If the Java virtual machine is not loaded, load it first. A reference to the Java virtual machine is returned in the *vm* parameter. You can pass `nil` for the *vm* parameter if you do not want this information. Pass `NO` for *usesyscl* if you want to use a new instance of the class loader to load the classes; otherwise, the system can reuse an existing instance of the class loader. If you pass `NO` for *usesyscl*, the new class loader will be released when you are done with it; otherwise, the class loader will be cached for use next time.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.5.

Declared In
`NSJavaSetup.h`

NSJavaClassesFromPath

Loads the Java classes located at the specified path. (Deprecated in Mac OS X v10.5.)

```
NSArray * NSJavaClassesFromPath (
   NSArray *path,
   NSArray *wanted,
   BOOL usesyscl,
   id *vm
);
```

Discussion
Loads and returns the Java classes in the specified bundle. If the Java virtual machine is not loaded, load it first. A reference to the Java virtual machine is returned in the *vm* parameter. You can pass `nil` for the *vm* parameter if you do not want this information. Pass an array of names of classes to load in the *wanted* parameter. If you pass `nil` for the *wanted* parameter, all classes at the specified path will be loaded. Pass `NO` for *usesyscl* if you want to use a new instance of the class loader to load the classes; otherwise, the system can reuse an existing instance of the class loader. If you pass `NO` for *usesyscl*, the new class loader will be released when you are done with it; otherwise, the class loader will be cached for use next time.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.5.

Declared In
`NSJavaSetup.h`

NSJavaNeedsToLoadClasses

Returns a Boolean value that indicates whether a virtual machine is needed or if Java classes are provided. (Deprecated in Mac OS X v10.5.)

```
BOOL NSJavaNeedsToLoadClasses (
   NSDictionary *plist
);
```

Discussion
Returns `YES` if a virtual machine is needed or if a virtual machine already exists and there's an indication that Java classes are provided.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`NSJavaSetup.h`

NSJavaNeedsVirtualMachine

Returns a Boolean value that indicates whether a Java virtual machine is required. (Deprecated in Mac OS X v10.5.)

```
BOOL NSJavaNeedsVirtualMachine (
   NSDictionary *plist
);
```

Discussion
Returns `YES` if *`plist`* contains a key saying that it requires Java.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`NSJavaSetup.h`

NSJavaObjectNamedInPath

Creates an instance of the named class using the class loader previously specified at the given path. (Deprecated in Mac OS X v10.5.)

```
id NSJavaObjectNamedInPath (
   NSString *name,
   NSArray *path
);
```

Discussion
Returns a new instance of the class *`name`*. The class loader must be already be set up for the specified *`path`* (you can do this using a function such as `NSJavaClassesFromPath` (page 2206)).

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.5.

Declared In
`NSJavaSetup.h`

NSJavaProvidesClasses

Returns a Boolean value that indicates whether Java classes are provided. (Deprecated in Mac OS X v10.5.)

```
BOOL NSJavaProvidesClasses (
   NSDictionary *plist
);
```

Discussion
Returns `YES` if *`plist`* contains an `NSJavaPath` key.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
NSJavaSetup.h

NSJavaSetup

Loads the Java virtual machine with specified parameters. (Deprecated in Mac OS X v10.5.)

```
id NSJavaSetup (
   NSDictionary *plist
);
```

Discussion
Part of the Java-to-Objective-C bridge. You normally shouldn't use it yourself.

You can pass `nil` for the *plist* dictionary, in which case the Java virtual machine will not be loaded, so you should probably just use `NSJavaSetupVirtualMachine` (page 2208) instead. The *plist* dictionary may contain the following key-value pairs.

- `NSJavaRoot`—An NSString indicating the root of the location where the application's classes are.
- `NSJavaPath`—An NSArray of NSStrings, each string containing one component of a class path whose components will be prepended by `NSJavaRoot` if they are not absolute locations.
- `NSJavaUserPath`—An NSString indicating another segment of the class path so that the application developer can customize where the class loader should search for classes. When searching for classes, this path is searched after the application's class path so that one cannot replace the classes used by the application.
- `NSJavaLibraryPath`—An NSArray of NSStrings, each string containing one component of a path to search for dynamic shared libraries needed by Java wrappers.
- `NSJavaClasses`—An NSArray of NSStrings, each string containing the name of one class that the VM should load so that their associated frameworks will be loaded.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.5.

Declared In
NSJavaSetup.h

NSJavaSetupVirtualMachine

Sets up the Java virtual machine. (Deprecated in Mac OS X v10.5.)

```
id NSJavaSetupVirtualMachine (void);
```

Discussion
Sets up and returns a reference to the Java virtual machine.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.5.

Declared In
NSJavaSetup.h

NSLocalizedString

Returns a localized version of a string.

```
NSString *NSLocalizedString(NSString *key, NSString *comment)
```

Return Value
The result of invoking `localizedStringForKey:value:table:` (page 180) on the main bundle and a `nil` table.

Discussion
You can specify Unicode characters in *key* using `\\Uxxxx`—see the `-u` option for for the `genstrings` utility.

For more information, see `NSBundle`.

Special Considerations

In Mac OS X v10.4 and earlier, to ensure correct parsing by the `genstrings` utility, the *key* parameter must not contain any high-ASCII characters.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
EnhancedAudioBurn
GridCalendar
Quartz Composer WWDC 2005 TextEdit
TextEditPlus
TrackBall

Declared In
NSBundle.h

NSLocalizedStringFromTable

Returns a localized version of a string.

```
NSString *NSLocalizedStringFromTable(NSString *key, NSString *tableName, NSString
 *comment)
```

Return Value
The result of invoking `localizedStringForKey:value:table:` (page 180) on the main bundle, passing it the specified *key* and *tableName*.

Discussion
You can specify Unicode characters in *key* using `\\Uxxxx`—see the `-u` option for for the `genstrings` utility.

For more information, see `NSBundle`.

Special Considerations

In Mac OS X v10.4 and earlier, to ensure correct parsing by the `genstrings` utility, the *key* parameter must not contain any high-ASCII characters.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BindingsJoystick
Mountains
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSBundle.h`

NSLocalizedStringFromTableInBundle

Returns a localized version of a string.

```
NSString *NSLocalizedStringFromTableInBundle(NSString *key, NSString *tableName,
NSBundle *bundle, NSString *comment)
```

Return Value
The result of invoking `localizedStringForKey:value:table:` (page 180) on *bundle*, passing it the specified *key* and *tableName*.

Discussion
You can specify Unicode characters in *key* using `\\Uxxxx`—see the `-u` option for for the `genstrings` utility.

For more information, see `NSBundle`.

Special Considerations

In Mac OS X v10.4 and earlier, to ensure correct parsing by the `genstrings` utility, the *key* parameter must not contain any high-ASCII characters.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSBundle.h`

NSLocalizedStringWithDefaultValue

Returns a localized version of a string.

```
NSString NSLocalizedStringWithDefaultValue(NSString *key, NSString *tableName,
NSBundle *bundle, NSString *value, NSString *comment)
```

Return Value
The result of invoking `localizedStringForKey:value:table:` (page 180) on *bundle*, passing it the specified *key*, *value*, and *tableName*.

Discussion
You can specify Unicode characters in *key* using `\\Uxxxx`—see the `-u` option for for the `genstrings` utility.

If you use `genstrings` to parse your code for localizable strings, you can use this method to specify an initial value that is different from `key`.

For more information, see `NSBundle`.

Special Considerations
In Mac OS X v10.4 and earlier, to ensure correct parsing by the `genstrings` utility, the *key* parameter must not contain any high-ASCII characters.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`NSBundle.h`

NSLocationInRange

Returns a Boolean value that indicates whether a specified position is in a given range.

```
BOOL NSLocationInRange (
   NSUInteger loc,
   NSRange range
);
```

Return Value
`YES` if *loc* lies within *range*—that is, if it's greater than or equal to `range.location` and less than `range.location` plus `range.length`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSRange.h`

NSLog

Logs an error message to the Apple System Log facility.

```
void NSLog (
   NSString *format,
   ...
);
```

Discussion
Simply calls `NSLogv` (page 2212), passing it a variable number of arguments.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSLogv` (page 2212)

Related Sample Code
GLSLShowpiece
OpenGLCaptureToMovie
Quartz Composer QCTV
Quartz Composer WWDC 2005 TextEdit
StickiesExample

Declared In
`NSObjCRuntime.h`

NSLogPageSize

Returns the binary log of the page size.

```
NSUInteger NSLogPageSize (void);
```

Return Value
The binary log of the page size.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSRoundDownToMultipleOfPageSize` (page 2233)
`NSRoundUpToMultipleOfPageSize` (page 2233)
`NSPageSize` (page 2225)

Declared In
`NSZone.h`

NSLogv

Logs an error message to the Apple System Log facility.

```
void NSLogv (
   NSString *format,
   va_list args
);
```

Discussion
Logs an error message to the Apple System Log facility (see `man 3 asl`). If the `STDERR_FILENO` file descriptor has been redirected away from the default or is going to a tty, it will also be written there. If you want to direct output elsewhere, you need to use a custom logging facility.

The message consists of a timestamp and the process ID prefixed to the string you pass in. You compose this string with a format string, *`format`*, and one or more arguments to be inserted into the string. The format specification allowed by these functions is that which is understood by `NSString`'s formatting capabilities (which is not necessarily the set of format escapes and flags understood by `printf`). The supported format specifiers are described in String Format Specifiers. A final hard return is added to the error message if one is not present in the format.

In general, you should use the `NSLog` (page 2211) function instead of calling this function directly. If you do use this function directly, you must have prepared the variable argument list in the *`args`* argument by calling the standard C macro `va_start`. Upon completion, you must similarly call the standard C macro `va_end` for this list.

Output from `NSLogv` is serialized, in that only one thread in a process can be doing the writing/logging described above at a time. All attempts at writing/logging a message complete before the next thread can begin its attempts.

The effects of `NSLogv` are not serialized with subsystems other than those discussed above (such as the standard I/O package) and do not produce side effects on those subsystems (such as causing buffered output to be flushed, which may be undesirable).

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSLog` (page 2211)

Declared In
`NSObjCRuntime.h`

NSMakeCollectable

Makes a newly allocated Core Foundation object eligible for collection.

```
NS_INLINE id NSMakeCollectable(CFTypeRef cf) {
    return cf ? (id)CFMakeCollectable(cf) : nil;
}
```

Discussion
This function is a wrapper for `CFMakeCollectable`, but its return type is `id`—avoiding the need for casting when using Cocoa objects.

This function may be useful when returning Core Foundation objects in code that must support both garbage-collected and non-garbage-collected environments, as illustrated in the following example.

```
- (CFDateRef)foo {
    CFDateRef aCFDate;
    // ...
    return [NSMakeCollectable(aCFDate) autorelease];
}
```

CFTypeRef style objects are garbage collected, yet only sometime after the last `CFRelease` is performed. Particularly for fully-bridged CFTypeRef objects such as CFStrings and collections (such as CFDictionary), you must call either `CFMakeCollectable` or the more type safe `NSMakeCollectable`, preferably right upon allocation.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSZone.h`

NSMakePoint

Creates a new `NSPoint` from the specified values.

```
NSPoint NSMakePoint (
   CGFloat x,
   CGFloat y
);
```

Return Value
An `NSPoint` having the coordinates *x* and *y*.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Clock Control
Dicey
Reducer
Sketch-112
WhackedTV

Declared In
`NSGeometry.h`

NSMakeRange

Creates a new NSRange from the specified values.

```
NSRange NSMakeRange (
   NSUInteger loc,
   NSUInteger len
);
```

Return Value
An `NSRange` with location *location* and length *length*.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CoreRecipes
iSpend
LayoutManagerDemo
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSRange.h`

NSMakeRect

Creates a new `NSRect` from the specified values.

```
NSRect NSMakeRect (
   CGFloat x,
   CGFloat y,
   CGFloat w,
   CGFloat h
);
```

Return Value
An `NSRect` having the specified origin of [*x*, *y*] and size of [*w*, *h*].

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Dicey
FilterDemo
GLSLShowpiece
WhackedTV
Worm

Declared In
`NSGeometry.h`

NSMakeSize

Returns a new `NSSize` from the specified values.

```
NSSize NSMakeSize (
   CGFloat w,
   CGFloat h
);
```

Return Value
An `NSSize` having the specified *`width`* and *`height`*.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTKitPlayer
QTQuartzPlayer
Quartz Composer QCTV
Reducer
Sketch-112

Declared In
`NSGeometry.h`

NSMapGet

Returns a map table value for the specified key.

```
void * NSMapGet (
   NSMapTable *table,
   const void *key
);
```

Return Value
The value that *table* maps to *key*, or NULL if *table* doesn't contain *key*.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSMapMember (page 2218)
NSEnumerateMapTable (page 2190)
NSNextMapEnumeratorPair (page 2223)
NSAllMapTableKeys (page 2157)
NSAllMapTableValues (page 2158)

Declared In
NSMapTable.h

NSMapInsert

Inserts a key-value pair into the specified table.

```
void NSMapInsert (
   NSMapTable *table,
   const void *key,
   const void *value
);
```

Discussion
Inserts *key* and *value* into *table*. If *key* matches a key already in *table*, *value* is retained and the previous value is released, using the retain and release callback functions that were specified when the table was created. Raises NSInvalidArgumentException if *key* is equal to the notAKeyMarker field of the table's NSMapTableKeyCallBacks (page 2272) structure.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSMapRemove (page 2218)
NSMapInsertIfAbsent (page 2216)
NSMapInsertKnownAbsent (page 2217)

Declared In
NSMapTable.h

NSMapInsertIfAbsent

Inserts a key-value pair into the specified table.

```
void * NSMapInsertIfAbsent (
   NSMapTable *table,
   const void *key,
   const void *value
);
```

Return Value
If *key* matches a key already in *table*, the preexisting key; otherwise, *key* and *value* are added to *table* and returns NULL.

Discussion
Raises NSInvalidArgumentException if *key* is equal to the notAKeyMarker field of the table's NSMapTableKeyCallBacks structure.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSMapRemove (page 2218)
NSMapInsert (page 2216)
NSMapInsertKnownAbsent (page 2217)

Declared In
NSMapTable.h

NSMapInsertKnownAbsent

Inserts a key-value pair into the specified table if the pair had not been previously added.

```
void NSMapInsertKnownAbsent (
   NSMapTable *table,
   const void *key,
   const void *value
);
```

Discussion
Inserts *key* (which must not be notAKeyMarker) and *value* into *table*. Unlike NSMapInsert, this function raises NSInvalidArgumentException if *table* already includes a key that matches *key*.

key is compared with notAKeyMarker using pointer comparison; if *key* is identical to notAKeyMarker, raises NSInvalidArgumentException.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSMapRemove (page 2218)
NSMapInsert (page 2216)
NSMapInsertIfAbsent (page 2216)

Declared In
NSMapTable.h

NSMapMember

Indicates whether a given table contains a given key.

```
BOOL NSMapMember (
   NSMapTable *table,
   const void *key,
   void **originalKey,
   void **value
);
```

Return Value

`YES` if *`table`* contains a key equal to *`key`*, otherwise `NO`.

Discussion

If *`table`* contains a key equal to *`key`*, *`originalKey`* is set to *`key`*, and *`value`* is set to the value that *`table`* maps to *`key`*.

Availability

Available in Mac OS X v10.0 and later.

See Also

`NSMapGet` (page 2215)
`NSEnumerateMapTable` (page 2190)
`NSNextMapEnumeratorPair` (page 2223)
`NSAllMapTableKeys` (page 2157)
`NSAllMapTableValues` (page 2158)

Declared In

`NSMapTable.h`

NSMapRemove

Removes a key and corresponding value from the specified table.

```
void NSMapRemove (
   NSMapTable *table,
   const void *key
);
```

Discussion

If *`key`* matches a key already in *`table`*, this function releases the preexisting key and its corresponding value.

Availability

Available in Mac OS X v10.0 and later.

See Also

`NSMapInsert` (page 2216)
`NSMapInsertIfAbsent` (page 2216)
`NSMapInsertKnownAbsent` (page 2217)

Declared In

`NSMapTable.h`

NSMaxRange

Returns the number 1 greater than the maximum value within the range.

```
NSUInteger NSMaxRange (
   NSRange range
);
```

Return Value
`range.location` + `range.length`—in other words, the number 1 greater than the maximum value within the range.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
iSpend
Quartz Composer WWDC 2005 TextEdit
TextEditPlus
TextLinks
TipWrapper

Declared In
`NSRange.h`

NSMaxX

Returns the largest x coordinate of a given rectangle.

```
CGFloat NSMaxX (
   NSRect aRect
);
```

Return Value
The largest x coordinate value within *aRect*.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSWidth` (page 2257)
`NSHeight` (page 2198)
`NSMaxY` (page 2220)

Related Sample Code
Dicey
QTQuartzPlayer
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus

Declared In
`NSGeometry.h`

NSMaxY

Returns the largest y coordinate of a given rectangle.

```
CGFloat NSMaxY (
   NSRect aRect
);
```

Return Value
The largest y coordinate value within *aRect*.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSWidth (page 2257)
NSHeight (page 2198)
NSMaxX (page 2219)

Related Sample Code
Dicey
QTQuartzPlayer
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus

Declared In
NSGeometry.h

NSMidX

Returns the x coordinate of a given rectangle's midpoint.

```
CGFloat NSMidX (
   NSRect aRect
);
```

Return Value
Returns the x coordinate of the center of *aRect*.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSWidth (page 2257)
NSHeight (page 2198)
NSMidY (page 2221)

Related Sample Code
CALayerEssentials
Polygons
QTQuartzPlayer
Sketch-112

TrackBall

Declared In
NSGeometry.h

NSMidY

Returns the y coordinate of a given rectangle's midpoint.

```
CGFloat NSMidY (
   NSRect aRect
);
```

Return Value
The y coordinate of *aRect*'s center point.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSWidth (page 2257)
NSHeight (page 2198)
NSMidX (page 2220)

Related Sample Code
CALayerEssentials
PDFKitLinker2
Polygons
Sketch-112
TrackBall

Declared In
NSGeometry.h

NSMinX

Returns the smallest x coordinate of a given rectangle.

```
CGFloat NSMinX (
   NSRect aRect
);
```

Return Value
The smallest x coordinate value within *aRect*.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSWidth (page 2257)
NSHeight (page 2198)
NSMinY (page 2222)

Related Sample Code
Clock Control
Dicey
OpenGLCompositorLab
QTQuartzPlayer
Sketch-112

Declared In
NSGeometry.h

NSMinY

Returns the smallest y coordinate of a given rectangle.

```
CGFloat NSMinY (
   NSRect aRect
);
```

Return Value
The smallest y coordinate value within *aRect*.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSWidth (page 2257)
NSHeight (page 2198)
NSMinX (page 2221)

Related Sample Code
Clock Control
Dicey
OpenGLCompositorLab
QTQuartzPlayer
Sketch-112

Declared In
NSGeometry.h

NSMouseInRect

Returns a Boolean value that indicates whether the point is in the specified rectangle.

```
BOOL NSMouseInRect (
   NSPoint aPoint,
   NSRect aRect,
   BOOL flipped
);
```

Return Value
YES if the hot spot of the cursor lies inside a given rectangle, otherwise NO.

Discussion
This method assumes an unscaled and unrotated coordinate system. Specify `YES` for *`isFlipped`* if the underlying view uses a flipped coordinate system.

Point-in-rectangle functions generally assume that the bottom edge of a rectangle is outside of the rectangle boundaries, while the upper edge is inside the boundaries. This method views *`aRect`* from the point of view of the user—that is, this method always treats the bottom edge of the rectangle as the one closest to the bottom edge of the user's screen. By making this adjustment, this function ensures consistent mouse-detection behavior from the user's perspective.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSPointInRect` (page 2227)

Related Sample Code
ImageMapExample

Declared In
`NSGeometry.h`

NSNextHashEnumeratorItem

Returns the next hash-table element in the enumeration.

```
void * NSNextHashEnumeratorItem (
   NSHashEnumerator *enumerator
);
```

Return Value
The next element in the table that *`enumerator`* is associated with, or `NULL` if *`enumerator`* has already iterated over all the elements.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSEnumerateHashTable` (page 2190)

Declared In
`NSHashTable.h`

NSNextMapEnumeratorPair

Returns a Boolean value that indicates whether the next map-table pair in the enumeration are set.

```
BOOL NSNextMapEnumeratorPair (
   NSMapEnumerator *enumerator,
   void **key,
   void **value
);
```

Return Value
`NO` if *`enumerator`* has already iterated over all the elements in the table that *`enumerator`* is associated with; otherwise, sets *`key`* and *`value`* to match the next key-value pair in the table and returns `YES`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSEnumerateMapTable` (page 2190)
`NSMapMember` (page 2218)
`NSMapGet` (page 2215)
`NSAllMapTableKeys` (page 2157)
`NSAllMapTableValues` (page 2158)

Declared In
`NSMapTable.h`

NSOffsetRect

Offsets the rectangle by the specified amount.

```
NSRect NSOffsetRect (
   NSRect aRect,
   CGFloat dX,
   CGFloat dY
);
```

Return Value
A copy of *`aRect`*, with its location shifted by *`dX`* along the x axis and by *`dY`* along the y axis.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSDivideRect` (page 2188)
`NSInsetRect` (page 2201)
`NSIntegralRect` (page 2202)

Related Sample Code
PDFView
Sketch-112
TextLinks

Declared In
`NSGeometry.h`

NSOpenStepRootDirectory

Returns the root directory of the user's system.

```
NSString * NSOpenStepRootDirectory (void);
```

Return Value
A string identifying the root directory of the user's system.

Discussion
For more information on file system utilities, see *Low-Level File Management Programming Topics*.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSHomeDirectory` (page 2199)
`NSHomeDirectoryForUser` (page 2200)

Declared In
`NSPathUtilities.h`

NSPageSize

Returns the number of bytes in a page.

```
NSUInteger NSPageSize (void);
```

Return Value
The number of bytes in a page.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSRoundDownToMultipleOfPageSize` (page 2233)
`NSRoundUpToMultipleOfPageSize` (page 2233)
`NSLogPageSize` (page 2212)

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSZone.h`

NSParameterAssert

Validates the specified parameter.

```
NSParameterAssert(condition)
```

Discussion
Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception.

This macro validates a parameter for an Objective-C method. Simply provide the parameter as the *`condition`* argument. The macro evaluates the parameter and, if it is false, it logs an error message that includes the parameter and then raises an exception.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined. All assertion macros return void.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSLog` (page 2211)
`NSLogv` (page 2212)
`NSAssert` (page 2159)
`NSCAssert` (page 2166)
`NSCParameterAssert` (page 2177)

Related Sample Code
MethodReplacement
NewsReader
Sketch-112
TimelineToTC
TrackBall

Declared In
`NSException.h`

NSPointFromCGPoint

Returns an `NSPoint` typecast from a `CGPoint`.

```
NSPoint NSPointFromCGPoint(CGPoint cgpoint) {
    return (*(NSPoint *)&(cgpoint));
}
```

Return Value
An `NSPoint` typecast from a `CGPoint`.

Availability
Available in Mac OS X v10.5 and later.

See Also
`NSPointToCGPoint` (page 2228)
`NSRectFromCGRect` (page 2230)

NSSizeFromCGSize (page 2237)

Declared In
NSGeometry.h

NSPointFromString

Returns a point from a text-based representation.

```
NSPoint NSPointFromString (
   NSString *aString
);
```

Parameters

aString
A string of the form "{*x*, *y*}".

Return Value
If *aString* is of the form "{*x*, *y*}" an NSPoint structure that uses x and y as the x and y coordinates, in that order.

If *aString* only contains a single number, it is used as the x coordinate. If *aString* does not contain any numbers, returns an NSPoint object whose x and y coordinates are both 0.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSStringFromPoint (page 2239)

Declared In
NSGeometry.h

NSPointInRect

Returns a Boolean value that indicates whether a given point is in a given rectangle.

```
BOOL NSPointInRect (
   NSPoint aPoint,
   NSRect aRect
);
```

Return Value
YES if *aPoint* is located within the rectangle represented by *aRect*, otherwise NO.

Discussion
Point-in-rectangle functions generally assume that the "upper" and "left" edges of a rectangle are inside the rectangle boundaries, while the "lower" and "right" edges are outside the boundaries. This method treats the "upper" and "left" edges of the rectangle as the ones containing the origin of the rectangle.

Special Considerations

The meanings of "upper" and "lower" (and "left" and "right") are relative to the current coordinate system and the location of the rectangle. For a rectangle of positive height located in positive x and y coordinates:

- In the default Mac OS X desktop coordinate system—where the origin is at the bottom left—the rectangle edge closest to the bottom of the screen is the "upper" edge (and is considered inside the rectangle).
- On iPhone OS and in a flipped coordinate system on Mac OS X desktop—where the origin is at the top left—the rectangle edge closest to the bottom of the screen is the "lower" edge (and is considered outside the rectangle).

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSMouseInRect` (page 2222)

Related Sample Code
FunkyOverlayWindow
LiveVideoMixer2
LiveVideoMixer3
Sketch-112
TrackBall

Declared In
`NSGeometry.h`

NSPointToCGPoint

Returns a `CGPoint` typecast from an `NSPoint`.

```
CGPoint NSPointToCGPoint(NSPoint nspoint) {
    union _ {NSPoint ns; CGPoint cg;};
    return ((union _ *)&nspoint)->cg;
}
```

Return Value
A `CGPoint` typecast from an `NSPoint`.

Availability
Available in Mac OS X v10.5 and later.

See Also
`NSPointFromCGPoint` (page 2226)
`NSRectToCGRect` (page 2231)
`NSSizeToCGSize` (page 2237)

Declared In
`NSGeometry.h`

NSProtocolFromString

Returns a the protocol with a given name.

```
Protocol *NSProtocolFromString (
    NSString *namestr
);
```

Parameters

namestr

The name of a protocol.

Return Value

The protocol object named by *namestr*, or `nil` if no protocol by that name is currently loaded. If *namestr* is `nil`, returns `nil`.

Availability

Available in Mac OS X v10.5 and later.

See Also

`NSStringFromProtocol` (page 2240)
`NSClassFromString` (page 2170)
`NSSelectorFromString` (page 2234)

Declared In

`NSObjCRuntime.h`

NSRangeFromString

Returns a range from a text-based representation.

```
NSRange NSRangeFromString (
   NSString *aString
);
```

Discussion

Scans *aString* for two integers which are used as the location and length values, in that order, to create an `NSRange` struct. If *aString* only contains a single integer, it is used as the location value. If *aString* does not contain any integers, this function returns an `NSRange` struct whose location and length values are both 0.

Availability

Available in Mac OS X v10.0 and later.

See Also

`NSStringFromRange` (page 2240)

Declared In

`NSRange.h`

NSReallocateCollectable

Reallocates collectable memory.

```
void *__strong NSReallocateCollectable (
   void *ptr,
   NSUInteger size,
   NSUInteger options
);
```

Discussion
Changes the size of the block of memory pointed to by *ptr* to *size* bytes. It may allocate new memory to replace the old, in which case it moves the contents of the old memory block to the new block, up to a maximum of *size* bytes.

options can be `0` or `NSScannedOption`: A value of `0` allocates nonscanned memory; a value of `NSScannedOption` allocates scanned memory.

This function returns `NULL` if it's unable to allocate the requested memory.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSZone.h`

NSRealMemoryAvailable

Returns information about the user's system.

```
NSUInteger NSRealMemoryAvailable (void);
```

Return Value
The number of bytes available in RAM.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSZone.h`

NSRectFromCGRect

Returns an `NSRect` typecast from a `CGRect`.

```
NSRect NSRectFromCGRect(CGRect cgrect) {
    return (*(NSRect *)&(cgrect));
}
```

Return Value
An `NSRect` typecast from a `CGRect`.

Availability
Available in Mac OS X v10.5 and later.

See Also
`NSRectToCGRect` (page 2231)
`NSPointFromCGPoint` (page 2226)

NSSizeFromCGSize (page 2237)

Declared In
NSGeometry.h

NSRectFromString

Returns a rectangle from a text-based representation.

```
NSRect NSRectFromString (
   NSString *aString
);
```

Discussion
Scans *aString* for four numbers which are used as the x and y coordinates and the width and height, in that order, to create an NSPoint object. If *aString* does not contain four numbers, those numbers that were scanned are used, and 0 is used for the remaining values. If *aString* does not contain any numbers, this function returns an NSRect object with a rectangle whose origin is (0, 0) and width and height are both 0.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSStringFromRect (page 2241)

Related Sample Code
DynamicProperties

Declared In
NSGeometry.h

NSRectToCGRect

Returns a CGRect typecast from an NSRect.

```
CGRect NSRectToCGRect(NSRect nsrect) {
    return (*(CGRect *)&(nsrect));
}
```

Return Value
A CGRect typecast from an NSRect.

Availability
Available in Mac OS X v10.5 and later.

See Also
NSRectFromCGRect (page 2230)
NSPointToCGPoint (page 2228)
NSSizeToCGSize (page 2237)

Declared In
NSGeometry.h

NSRecycleZone

Frees memory in a zone.

```
void NSRecycleZone (
   NSZone *zone
);
```

Discussion
Frees *zone* after adding any of its pointers still in use to the default zone. (This strategy prevents retained objects from being inadvertently destroyed.)

Availability
Available in Mac OS X v10.0 and later.

See Also
NSCreateZone (page 2180)
NSZoneMalloc (page 2259)

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
NSZone.h

NSResetHashTable

Deletes the elements of the specified hash table.

```
void NSResetHashTable (
   NSHashTable *table
);
```

Discussion
Releases each element but doesn't deallocate *table*. This function is useful for preserving the capacity of *table*.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSFreeHashTable (page 2193)

Declared In
NSHashTable.h

NSResetMapTable

Deletes the elements of the specified map table.

```
void NSResetMapTable (
   NSMapTable *table
);
```

Parameters

table

A reference to a map table structure.

Discussion

Releases each key and value but doesn't deallocate *table*. This method is useful for preserving the capacity of *table*.

Availability

Available in Mac OS X v10.0 and later.

See Also

NSFreeMapTable (page 2194)

Declared In

NSMapTable.h

NSRoundDownToMultipleOfPageSize

Returns the specified number of bytes rounded down to a multiple of the page size.

```
NSUInteger NSRoundDownToMultipleOfPageSize (
   NSUInteger bytes
);
```

Return Value

In bytes, the multiple of the page size that is closest to, but not greater than, *byteCount* (that is, the number of bytes rounded down to a multiple of the page size).

Availability

Available in Mac OS X v10.0 and later.

See Also

NSPageSize (page 2225)

NSLogPageSize (page 2212)

NSRoundUpToMultipleOfPageSize (page 2233)

Declared In

NSZone.h

NSRoundUpToMultipleOfPageSize

Returns the specified number of bytes rounded up to a multiple of the page size.

```
NSUInteger NSRoundUpToMultipleOfPageSize (
   NSUInteger bytes
);
```

Return Value
In bytes, the multiple of the page size that is closest to, but not less than, *byteCount* (that is, the number of bytes rounded up to a multiple of the page size).

Availability
Available in Mac OS X v10.0 and later.

See Also
NSPageSize (page 2225)
NSLogPageSize (page 2212)
NSRoundDownToMultipleOfPageSize (page 2233)

Declared In
NSZone.h

NSSearchPathForDirectoriesInDomains

Creates a list of directory search paths.

```
NSArray * NSSearchPathForDirectoriesInDomains (
   NSSearchPathDirectory directory,
   NSSearchPathDomainMask domainMask,
   BOOL expandTilde
);
```

Discussion
Creates a list of path strings for the specified directories in the specified domains. The list is in the order in which you should search the directories. If *expandTilde* is YES, tildes are expanded as described in stringByExpandingTildeInPath (page 1602).

For more information on file system utilities, see Locating Directories on the System.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BundleLoader
Core Data HTML Store
CoreRecipes
SampleScannerApp
SpotlightFortunes

Declared In
NSPathUtilities.h

NSSelectorFromString

Returns the selector with a given name.

```
SEL NSSelectorFromString (
   NSString *aSelectorName
);
```

Parameters

aSelectorName

A string of any length, with any characters, that represents the name of a selector.

Return Value

The selector named by *aSelectorName*. If *aSelectorName* is `nil`, or cannot be converted to UTF-8 (this should be only due to insufficient memory), returns `(SEL)0`.

Discussion

To make a selector, `NSSelectorFromString` passes a UTF-8 encoded character representation of *aSelectorName* to `sel_registerName` and returns the value returned by that function. Note, therefore, that if the selector does not exist it is registered and the newly-registered selector is returned.

Recall that a colon (":") is part of a method name; `setHeight` is not the same as `setHeight:`. For more about methods names, see The Language in *The Objective-C 2.0 Programming Language*.

Availability

Available in Mac OS X v10.0 and later.

See Also

`NSStringFromSelector` (page 2241)

`NSProtocolFromString` (page 2228)

`NSClassFromString` (page 2170)

Related Sample Code

CoreRecipes

ImageMapExample

Declared In

`NSObjCRuntime.h`

NSSetUncaughtExceptionHandler

Changes the top-level error handler.

```
void NSSetUncaughtExceptionHandler (
   NSUncaughtExceptionHandler *
);
```

Discussion

Sets the top-level error-handling function where you can perform last-minute logging before the program terminates.

Availability

Available in Mac OS X v10.0 and later.

See Also

`NSGetUncaughtExceptionHandler` (page 2195)

`reportException:` (`NSApplication`)

Declared In
NSException.h

NSSetZoneName

Sets the name of the specified zone.

```
void NSSetZoneName (
   NSZone *zone,
   NSString *name
);
```

Discussion
Sets the name of *zone* to *name*, which can aid in debugging.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSZoneName (page 2260)

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
NSZone.h

NSShouldRetainWithZone

Indicates whether an object should be retained.

```
BOOL NSShouldRetainWithZone (
   id anObject,
   NSZone *requestedZone
);
```

Parameters

anObject
An object.

requestedZone
A memory zone.

Return Value
Returns YES if *requestedZone* is NULL, the default zone, or the zone in which *anObject* was allocated; otherwise NO.

Discussion
This function is typically called from inside an NSObject's copyWithZone: (page 1157), when deciding whether to retain *anObject* as opposed to making a copy of it.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSObject.h

NSSizeFromCGSize

Returns an NSSize typecast from a CGSize.

```
NSSize NSSizeFromCGSize(CGSize cgsize) {
    return (*(NSSize *)&(cgsize));
}
```

Return Value
An NSSize typecast from a CGSize.

Availability
Available in Mac OS X v10.5 and later.

See Also
NSSizeToCGSize (page 2237)
NSPointFromCGPoint (page 2226)
NSRectFromCGRect (page 2230)

Declared In
NSGeometry.h

NSSizeFromString

Returns an NSSize from a text-based representation.

```
NSSize NSSizeFromString (
   NSString *aString
);
```

Discussion
Scans *aString* for two numbers which are used as the width and height, in that order, to create an NSSize struct. If *aString* only contains a single number, it is used as the width. The *aString* argument should be formatted like the output of NSStringFromSize (page 2242), for example, @"{10,20}". If *aString* does not contain any numbers, this function returns an NSSize struct whose width and height are both 0.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSStringFromSize (page 2242)

Declared In
NSGeometry.h

NSSizeToCGSize

Returns a CGSize typecast from an NSSize.

```
CGSize NSSizeToCGSize(NSSize nssize) {
    return (*(CGSize *)&(nssize));
}
```

Return Value
A `CGSize` typecast from an `NSSize`.

Availability
Available in Mac OS X v10.5 and later.

See Also
`NSSizeFromCGSize` (page 2237)
`NSPointToCGPoint` (page 2228)
`NSRectToCGRect` (page 2231)

Related Sample Code
Quartz 2D Shadings

Declared In
`NSGeometry.h`

NSStringFromClass

Returns the name of a class as a string.

```
NSString * NSStringFromClass (
   Class aClass
);
```

Parameters

aClass
A class.

Return Value
A string containing the name of *aClass*. If *aClass* is `nil`, returns `nil`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSClassFromString` (page 2170)
`NSStringFromProtocol` (page 2240)
`NSStringFromSelector` (page 2241)

Related Sample Code
Sketch-112
ToolbarSample

Declared In
`NSObjCRuntime.h`

NSStringFromHashTable

Returns a string describing the hash table's contents.

```
NSString * NSStringFromHashTable (
   NSHashTable *table
);
```

Return Value
A string describing *table*'s contents.

Discussion
The function iterates over the elements of *table*, and for each one appends the string returned by the `describe` callback function. If `NULL` was specified for the callback function, the hexadecimal value of each pointer is added to the string.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSHashTable.h`

NSStringFromMapTable

Returns a string describing the map table's contents.

```
NSString * NSStringFromMapTable (
   NSMapTable *table
);
```

Parameters
table
A reference to a map table structure.

Return Value
A string describing the map table's contents.

Discussion
The function iterates over the key-value pairs of *table* and for each one appends the string "*a* = *b*;\n", where *a* and *b* are the key and value strings returned by the corresponding `describe` callback functions. If `NULL` was specified for the callback function, a and b are the key and value pointers, expressed as hexadecimal numbers.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSMapTable.h`

NSStringFromPoint

Returns a string representation of a point.

```
NSString * NSStringFromPoint (
   NSPoint aPoint
);
```

Parameters

aPoint

A point structure.

Return Value

A string of the form "{*a*, *b*}", where *a* and *b* are the x and y coordinates of *aPoint*.

Availability

Available in Mac OS X v10.0 and later.

See Also

NSPointFromString (page 2227)

Declared In

NSGeometry.h

NSStringFromProtocol

Returns the name of a protocol as a string.

```
NSString * NSStringFromProtocol (
    Protocol *proto
);
```

Parameters

proto

A protocol.

Return Value

A string containing the name of *proto*.

Availability

Available in Mac OS X v10.5 and later.

See Also

NSProtocolFromString (page 2228)

NSStringFromClass (page 2238)

NSStringFromSelector (page 2241)

Declared In

NSObjCRuntime.h

NSStringFromRange

Returns a string representation of a range.

```
NSString * NSStringFromRange (
   NSRange range
);
```

Return Value
A string of the form "{*a*, *b*}", where *a* and *b* are non-negative integers representing `aRange`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSRange.h`

NSStringFromRect

Returns a string representation of a rectangle.

```
NSString * NSStringFromRect (
   NSRect aRect
);
```

Discussion
Returns a string of the form "{{*a*, *b*}, {*c*, *d*}}", where *a*, *b*, *c*, and *d* are the x and y coordinates and the width and height, respectively, of `aRect`.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSRectFromString` (page 2231)

Related Sample Code
DynamicProperties

Declared In
`NSGeometry.h`

NSStringFromSelector

Returns a string representation of a given selector.

```
NSString *NSStringFromSelector (
    SEL *aSelector
);
```

Parameters
`aSelector`
A selector.

Return Value
A string representation of `aSelector`.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSelectorFromString (page 2234)
NSStringFromProtocol (page 2240)
NSStringFromClass (page 2238)

Related Sample Code
CallJS
EnhancedAudioBurn
QT Capture Widget
SpecialPictureProtocol
WebKitPluginWithJavaScript

Declared In
NSObjCRuntime.h

NSStringFromSize

Returns a string representation of a size.

```
NSString * NSStringFromSize (
   NSSize aSize
);
```

Return Value
A string of the form "{a, b}", where a and b are the width and height, respectively, of *aSize*.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSizeFromString (page 2237)

Declared In
NSGeometry.h

NSSwapBigDoubleToHost

A utility for swapping the bytes of a number.

```
double NSSwapBigDoubleToHost (
   NSSwappedDouble x
);
```

Discussion
Converts the big-endian value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes of *x*, this function calls NSSwapDouble (page 2245) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSwapHostDoubleToBig (page 2246)

NSSwapLittleDoubleToHost (page 2251)

Declared In
NSByteOrder.h

NSSwapBigFloatToHost

A utility for swapping the bytes of a number.

```
float NSSwapBigFloatToHost (
   NSSwappedFloat x
);
```

Discussion
Converts the big-endian value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes of *x*, this function calls NSSwapFloat (page 2245) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSwapHostFloatToBig (page 2246)
NSSwapLittleFloatToHost (page 2252)

Declared In
NSByteOrder.h

NSSwapBigIntToHost

A utility for swapping the bytes of a number.

```
unsigned int NSSwapBigIntToHost (
   unsigned int x
);
```

Discussion
Converts the big-endian value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes of *x*, this function calls NSSwapInt (page 2251) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSwapHostIntToBig (page 2247)
NSSwapLittleIntToHost (page 2252)

Declared In
NSByteOrder.h

NSSwapBigLongLongToHost

A utility for swapping the bytes of a number.

```
unsigned long long NSSwapBigLongLongToHost (
   unsigned long long x
);
```

Discussion
Converts the big-endian value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes of *x*, this function calls `NSSwapLongLong` (page 2254) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSSwapHostLongLongToBig` (page 2248)
`NSSwapLittleLongLongToHost` (page 2252)

Declared In
`NSByteOrder.h`

NSSwapBigLongToHost

A utility for swapping the bytes of a number.

```
unsigned long NSSwapBigLongToHost (
   unsigned long x
);
```

Discussion
Converts the big-endian value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes of *x*, this function calls `NSSwapLong` (page 2254) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSSwapHostLongToBig` (page 2249)
`NSSwapLittleLongToHost` (page 2253)

Declared In
`NSByteOrder.h`

NSSwapBigShortToHost

A utility for swapping the bytes of a number.

```
unsigned short NSSwapBigShortToHost (
   unsigned short x
);
```

Discussion
Converts the big-endian value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes of *x*, this function calls `NSSwapShort` (page 2255) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSwapHostShortToBig (page 2250)
NSSwapLittleShortToHost (page 2253)

Declared In
NSByteOrder.h

NSSwapDouble

A utility for swapping the bytes of a number.

```
NSSwappedDouble NSSwapDouble (
   NSSwappedDouble x
);
```

Discussion
Swaps the bytes of *x* and returns the resulting value. Bytes are swapped from each low-order position to the corresponding high-order position and vice versa. For example, if the bytes of *x* are numbered from 1 to 8, this function swaps bytes 1 and 8, bytes 2 and 7, bytes 3 and 6, and bytes 4 and 5.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSwapLongLong (page 2254)
NSSwapFloat (page 2245)

Declared In
NSByteOrder.h

NSSwapFloat

A utility for swapping the bytes of a number.

```
NSSwappedFloat NSSwapFloat (
   NSSwappedFloat x
);
```

Discussion
Swaps the bytes of *x* and returns the resulting value. Bytes are swapped from each low-order position to the corresponding high-order position and vice versa. For example, if the bytes of *x* are numbered from 1 to 4, this function swaps bytes 1 and 4, and bytes 2 and 3.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSwapLong (page 2254)
NSSwapDouble (page 2245)

Declared In
NSByteOrder.h

NSSwapHostDoubleToBig

A utility for swapping the bytes of a number.

```
NSSwappedDouble NSSwapHostDoubleToBig (
   double x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to big-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapDouble` (page 2245) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSSwapBigDoubleToHost` (page 2242)
`NSSwapHostDoubleToLittle` (page 2246)

Declared In
`NSByteOrder.h`

NSSwapHostDoubleToLittle

A utility for swapping the bytes of a number.

```
NSSwappedDouble NSSwapHostDoubleToLittle (
   double x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to little-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapDouble` (page 2245) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSSwapLittleDoubleToHost` (page 2251)
`NSSwapHostDoubleToBig` (page 2246)

Declared In
`NSByteOrder.h`

NSSwapHostFloatToBig

A utility for swapping the bytes of a number.

```
NSSwappedFloat NSSwapHostFloatToBig (
   float x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to big-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapFloat` (page 2245) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSSwapBigFloatToHost` (page 2243)
`NSSwapHostFloatToLittle` (page 2247)

Declared In
`NSByteOrder.h`

NSSwapHostFloatToLittle

A utility for swapping the bytes of a number.

```
NSSwappedFloat NSSwapHostFloatToLittle (
   float x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to little-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapFloat` (page 2245) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSSwapLittleFloatToHost` (page 2252)
`NSSwapHostFloatToBig` (page 2246)

Declared In
`NSByteOrder.h`

NSSwapHostIntToBig

A utility for swapping the bytes of a number.

```
unsigned int NSSwapHostIntToBig (
   unsigned int x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to big-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapInt` (page 2251) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSwapBigIntToHost (page 2243)
NSSwapHostIntToLittle (page 2248)

Related Sample Code
QTMetadataEditor

Declared In
NSByteOrder.h

NSSwapHostIntToLittle

A utility for swapping the bytes of a number.

```
unsigned int NSSwapHostIntToLittle (
   unsigned int x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to little-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls NSSwapInt (page 2251) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSwapLittleIntToHost (page 2252)
NSSwapHostIntToBig (page 2247)

Declared In
NSByteOrder.h

NSSwapHostLongLongToBig

A utility for swapping the bytes of a number.

```
unsigned long long NSSwapHostLongLongToBig (
   unsigned long long x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to big-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls NSSwapLongLong (page 2254) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSwapBigLongLongToHost (page 2243)
NSSwapHostLongLongToLittle (page 2249)

Declared In
NSByteOrder.h

NSSwapHostLongLongToLittle

A utility for swapping the bytes of a number.

```
unsigned long long NSSwapHostLongLongToLittle (
   unsigned long long x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to little-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls NSSwapLongLong (page 2254) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSwapLittleLongLongToHost (page 2252)
NSSwapHostLongLongToBig (page 2248)

Declared In
NSByteOrder.h

NSSwapHostLongToBig

A utility for swapping the bytes of a number.

```
unsigned long NSSwapHostLongToBig (
   unsigned long x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to big-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls NSSwapLong (page 2254) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSwapBigLongToHost (page 2244)
NSSwapHostLongToLittle (page 2249)

Declared In
NSByteOrder.h

NSSwapHostLongToLittle

A utility for swapping the bytes of a number.

```
unsigned long NSSwapHostLongToLittle (
   unsigned long x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to little-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapLong` (page 2254) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSSwapLittleLongToHost` (page 2253)
`NSSwapHostLongToBig` (page 2249)

Declared In
`NSByteOrder.h`

NSSwapHostShortToBig

A utility for swapping the bytes of a number.

```
unsigned short NSSwapHostShortToBig (
   unsigned short x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to big-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapShort` (page 2255) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSSwapBigShortToHost` (page 2244)
`NSSwapHostShortToLittle` (page 2250)

Declared In
`NSByteOrder.h`

NSSwapHostShortToLittle

A utility for swapping the bytes of a number.

```
unsigned short NSSwapHostShortToLittle (
   unsigned short x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to little-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapShort` (page 2255) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSwapLittleShortToHost (page 2253)
NSSwapHostShortToBig (page 2250)

Related Sample Code
AudioBurn

Declared In
NSByteOrder.h

NSSwapInt

A utility for swapping the bytes of a number.

```
unsigned int NSSwapInt (
   unsigned int inv
);
```

Discussion
Swaps the bytes of *inv* and returns the resulting value. Bytes are swapped from each low-order position to the corresponding high-order position and vice versa. For example, if the bytes of *inv* are numbered from 1 to 4, this function swaps bytes 1 and 4, and bytes 2 and 3.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSwapShort (page 2255)
NSSwapLong (page 2254)
NSSwapLongLong (page 2254)

Declared In
NSByteOrder.h

NSSwapLittleDoubleToHost

A utility for swapping the bytes of a number.

```
double NSSwapLittleDoubleToHost (
   NSSwappedDouble x
);
```

Discussion
Converts the little-endian formatted value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes of *x*, this function calls NSSwapDouble (page 2245) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSwapHostDoubleToLittle (page 2246)
NSSwapBigDoubleToHost (page 2242)
NSConvertSwappedDoubleToHost (page 2173)

Declared In
NSByteOrder.h

NSSwapLittleFloatToHost

A utility for swapping the bytes of a number.

```
float NSSwapLittleFloatToHost (
   NSSwappedFloat x
);
```

Discussion
Converts the little-endian formatted value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes of *x*, this function calls `NSSwapFloat` (page 2245) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSSwapHostFloatToLittle` (page 2247)
`NSSwapBigFloatToHost` (page 2243)
`NSConvertSwappedFloatToHost` (page 2173)

Declared In
NSByteOrder.h

NSSwapLittleIntToHost

A utility for swapping the bytes of a number.

```
unsigned int NSSwapLittleIntToHost (
   unsigned int x
);
```

Discussion
Converts the little-endian formatted value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapInt` (page 2251) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSSwapHostIntToLittle` (page 2248)
`NSSwapBigIntToHost` (page 2243)

Declared In
NSByteOrder.h

NSSwapLittleLongLongToHost

A utility for swapping the bytes of a number.

```
unsigned long long NSSwapLittleLongLongToHost (
   unsigned long long x
);
```

Discussion
Converts the little-endian formatted value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapLongLong` (page 2254) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSSwapHostLongLongToLittle` (page 2249)
`NSSwapBigLongLongToHost` (page 2243)

Declared In
`NSByteOrder.h`

NSSwapLittleLongToHost

A utility for swapping the bytes of a number.

```
unsigned long NSSwapLittleLongToHost (
   unsigned long x
);
```

Discussion
Converts the little-endian formatted value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes of x, this function calls `NSSwapLong` (page 2254) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
`NSSwapHostLongToLittle` (page 2249)
`NSSwapBigLongToHost` (page 2244)
`NSSwapLong` (page 2254)

Declared In
`NSByteOrder.h`

NSSwapLittleShortToHost

A utility for swapping the bytes of a number.

```
unsigned short NSSwapLittleShortToHost (
   unsigned short x
);
```

Discussion
Converts the little-endian formatted value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes of *x*, this function calls `NSSwapShort` (page 2255) to perform the swap.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSwapHostShortToLittle (page 2250)
NSSwapBigShortToHost (page 2244)

Declared In
NSByteOrder.h

NSSwapLong

A utility for swapping the bytes of a number.

```
unsigned long NSSwapLong (
   unsigned long inv
);
```

Discussion
Swaps the bytes of *inv* and returns the resulting value. Bytes are swapped from each low-order position to the corresponding high-order position and vice versa. For example, if the bytes of *inv* are numbered from 1 to 4, this function swaps bytes 1 and 4, and bytes 2 and 3.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSwapLongLong (page 2254)
NSSwapInt (page 2251)
NSSwapFloat (page 2245)

Declared In
NSByteOrder.h

NSSwapLongLong

A utility for swapping the bytes of a number.

```
unsigned long long NSSwapLongLong (
   unsigned long long inv
);
```

Discussion
Swaps the bytes of *inv* and returns the resulting value. Bytes are swapped from each low-order position to the corresponding high-order position and vice versa. For example, if the bytes of *inv* are numbered from 1 to 8, this function swaps bytes 1 and 8, bytes 2 and 7, bytes 3 and 6, and bytes 4 and 5.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSwapLong (page 2254)
NSSwapDouble (page 2245)

Declared In
NSByteOrder.h

NSSwapShort

A utility for swapping the bytes of a number.

```
unsigned short NSSwapShort (
   unsigned short inv
);
```

Discussion
Swaps the low-order and high-order bytes of *inv* and returns the resulting value.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSwapInt (page 2251)
NSSwapLong (page 2254)

Declared In
NSByteOrder.h

NSTemporaryDirectory

Returns the path of the temporary directory for the current user.

```
NSString * NSTemporaryDirectory (void);
```

Return Value
A string containing the path of the temporary directory for the current user. If no such directory is currently available, returns nil.

Discussion
For more information on file system utilities, see *Low-Level File Management Programming Topics*.

The temporary directory is determined by confstr(3) passing the _CS_DARWIN_USER_TEMP_DIR flag. The erase rules are whatever match that directory.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSearchPathForDirectoriesInDomains (page 2234)
NSHomeDirectory (page 2199)

Related Sample Code
Core Data HTML Store
QTRecorder
SpotlightFortunes

Declared In
NSPathUtilities.h

NSUnionRange

Returns the union of the specified ranges.

```
NSRange NSUnionRange (
   NSRange range1,
   NSRange range2
);
```

Return Value
A range covering all indices in and between *range1* and *range2*. If one range is completely contained in the other, the returned range is equal to the larger range.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSIntersectionRange (page 2203)

Declared In
NSRange.h

NSUnionRect

Calculates the union of two rectangles.

```
NSRect NSUnionRect (
   NSRect aRect,
   NSRect bRect
);
```

Discussion
Returns the smallest rectangle that completely encloses both *aRect* and *bRect*. If one of the rectangles has 0 (or negative) width or height, a copy of the other rectangle is returned; but if both have 0 (or negative) width or height, the returned rectangle has its origin at (0.0, 0.0) and has 0 width and height.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSIntersectionRect (page 2203)

Related Sample Code
CarbonCocoaCoreImageTab
PDFKitLinker2
Reducer
Sketch-112
Worm

Declared In
NSGeometry.h

NSUserName

Returns the logon name of the current user.

```
NSString * NSUserName (void);
```

Return Value
The logon name of the current user.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSFullUserName (page 2194)
NSHomeDirectory (page 2199)
NSHomeDirectoryForUser (page 2200)

Declared In
NSPathUtilities.h

NSWidth

Returns the width of the specified rectangle.

```
CGFloat NSWidth (
   NSRect aRect
);
```

Return Value
The width of *aRect*.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSMaxX (page 2219)
NSMaxY (page 2220)
NSMidX (page 2220)
NSMidY (page 2221)
NSMinX (page 2221)
NSMinY (page 2222)
NSHeight (page 2198)

Related Sample Code
Aperture Edit Plugin - Borders & Titles
Clock Control
CocoaVideoFrameToGWorld
iSpend

TrackBall

Declared In
NSGeometry.h

NSZoneCalloc

Allocates memory in a zone.

```
void * NSZoneCalloc (
   NSZone *zone,
   NSUInteger numElems,
   NSUInteger byteSize
);
```

Discussion
Allocates enough memory from *zone* for *numElems* elements, each with a size *numBytes* bytes, and returns a pointer to the allocated memory. The memory is initialized with zeros. This function returns NULL if it was unable to allocate the requested memory.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSDefaultMallocZone (page 2188)
NSRecycleZone (page 2232)
NSZoneFree (page 2258)
NSZoneMalloc (page 2259)
NSZoneRealloc (page 2260)

Declared In
NSZone.h

NSZoneFree

Deallocates a block of memory in the specified zone.

```
void NSZoneFree (
   NSZone *zone,
   void *ptr
);
```

Discussion
Returns memory to the *zone* from which it was allocated. The standard C function free does the same, but spends time finding which zone the memory belongs to.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSRecycleZone (page 2232)
NSZoneMalloc (page 2259)
NSZoneCalloc (page 2258)

NSZoneRealloc (page 2260)

Related Sample Code
AudioBurn

Declared In
NSZone.h

NSZoneFromPointer

Gets the zone for a given block of memory.

```
NSZone * NSZoneFromPointer (
   void *ptr
);
```

Return Value
The zone for the block of memory indicated by *pointer*, or NULL if the block was not allocated from a zone.

Discussion
pointer must be one that was returned by a prior call to an allocation function.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSZoneCalloc (page 2258)
NSZoneMalloc (page 2259)
NSZoneRealloc (page 2260)

Declared In
NSZone.h

NSZoneMalloc

Allocates memory in a zone.

```
void * NSZoneMalloc (
   NSZone *zone,
   NSUInteger size
);
```

Discussion
Allocates *size* bytes in *zone* and returns a pointer to the allocated memory. This function returns NULL if it was unable to allocate the requested memory.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSDefaultMallocZone (page 2188)
NSRecycleZone (page 2232)
NSZoneFree (page 2258)

NSZoneCalloc (page 2258)
NSZoneRealloc (page 2260)

Related Sample Code
AudioBurn

Declared In
NSZone.h

NSZoneName

Returns the name of the specified zone.

```
NSString * NSZoneName (
   NSZone *zone
);
```

Return Value
A string containing the name associated with *zone*. If *zone* is nil, the default zone is used. If no name is associated with *zone*, the returned string is empty.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSSetZoneName (page 2236)

Declared In
NSZone.h

NSZoneRealloc

Allocates memory in a zone.

```
void * NSZoneRealloc (
   NSZone *zone,
   void *ptr,
   NSUInteger size
);
```

Discussion
Changes the size of the block of memory pointed to by *ptr* to *size* bytes. It may allocate new memory to replace the old, in which case it moves the contents of the old memory block to the new block, up to a maximum of *size* bytes. *ptr* may be NULL. This function returns NULL if it was unable to allocate the requested memory.

Availability
Available in Mac OS X v10.0 and later.

See Also
NSDefaultMallocZone (page 2188)
NSRecycleZone (page 2232)
NSZoneFree (page 2258)

NSZoneCalloc (page 2258)
NSZoneMalloc (page 2259)

Declared In
NSZone.h

NS_DURING

Marks the start of the exception-handling domain.

```
NS_DURING
```

Discussion
The NS_DURING macro marks the start of the exception-handling domain for a section of code. (The NS_HANDLER (page 2262)macro marks the end of the domain.) Within the exception-handling domain you can raise an exception, giving the local exception handler (or lower exception handlers) a chance to handle it.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
EnhancedAudioBurn
StickiesExample

Declared In
NSException.h

NS_ENDHANDLER

Marks the end of the local event handler.

```
NS_ENDHANDLER
```

Discussion
The NS_ENDHANDLER marks the end of a section of code that is a local exception handler. (The NS_HANDLER (page 2262)macros marks the beginning of this section.) If an exception is raised in the exception handling domain marked off by the NS_DURING (page 2261) and NS_HANDLER (page 2262), the local exception handler (if specified) is given a chance to handle the exception.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
EnhancedAudioBurn
StickiesExample

Declared In
NSException.h

NS_HANDLER

Marks the end of the exception-handling domain and the start of the local exception handler.

```
NS_HANDLER
```

Discussion
The NS_HANDLER macro marks end of a section of code that is an exception-handling domain while at the same time marking the beginning of a section of code that is a local exception handler for that domain. (The NS_DURING (page 2261) macro marks the beginning of the exception-handling domain; the NS_ENDHANDLER (page 2261) marks the end of the local exception handler.) If an exception is raised in the exception-handling domain, the local exception handler is first given the chance to handle the exception before lower-level handlers are given a chance.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
EnhancedAudioBurn
StickiesExample

Declared In
NSException.h

NS_VALUERETURN

Permits program control to exit from an exception-handling domain with a value of a specified type.

```
NS_VALUERETURN(val, type)
```

Parameters

val
A value to preserve beyond the exception-handling domain.

type
The type of the value specified in val.

Discussion
The NS_VALUERETURN macro returns program control to the caller out of the exception-handling domain—that is, a section of code between the NS_DURING (page 2261) and NS_HANDLER (page 2262) macros that might raise an exception. The specified value (of the specified type) is returned to the caller. The standard return statement does not work as expected in the exception-handling domain.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSException.h

NS_VOIDRETURN

Permits program control to exit from an exception-handling domain.

NS_VOIDRETURN

Discussion
The NS_VOIDRETURN macro returns program control to the caller out of the exception-handling domain—that is, a section of code between the NS_DURING (page 2261) and NS_HANDLER (page 2262) macros that might raise an exception. The standard return statement does not work as expected in the exception-handling domain.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSException.h

Data Types

Foundation Data Types Reference

Framework: Foundation/Foundation.h

Overview

This document describes the data types and constants found in the Foundation framework.

Data Types

NSAppleEventManagerSuspensionID

Identifies an Apple event whose handling has been suspended. Can be used to resume handling of the Apple event.

```
typedef const struct __NSAppleEventManagerSuspension
*NSAppleEventManagerSuspensionID;
```

Availability
Available in Mac OS X v10.3 and later.

Declared In
`NSAppleEventManager.h`

NSByteOrder

These constants specify an endian format.

```
enum _NSByteOrder {
    NS_UnknownByteOrder = CFByteOrderUnknown,
    NS_LittleEndian = CFByteOrderLittleEndian,
    NS_BigEndian = CFByteOrderBigEndian
};
```

Constants
`NS_UnknownByteOrder`
The byte order is unknown.

Available in Mac OS X v10.0 and later.

Declared in `NSByteOrder.h`.

`NS_LittleEndian`
The byte order is little endian.

Available in Mac OS X v10.0 and later.

Declared in `NSByteOrder.h`.

`NS_BigEndian`
The byte order is big endian.

Available in Mac OS X v10.0 and later.

Declared in `NSByteOrder.h`.

Discussion
These constants are returned by `NSHostByteOrder` (page 2200).

Declared In
`NSByteOrder.h`

NSComparisonResult

These constants are used to indicate how items in a request are ordered.

```
typedef enum _NSComparisonResult {
    NSOrderedAscending = -1,
    NSOrderedSame,
    NSOrderedDescending
} NSComparisonResult;
```

Constants
`NSOrderedAscending`
The left operand is smaller than the right operand.

Available in Mac OS X v10.0 and later.

Declared in `NSObjCRuntime.h`.

`NSOrderedSame`
The two operands are equal.

Available in Mac OS X v10.0 and later.

Declared in `NSObjCRuntime.h`.

`NSOrderedDescending`
The left operand is greater than the right operand.

Available in Mac OS X v10.0 and later.

Declared in `NSObjCRuntime.h`.

Discussion
These constants are used to indicate how items in a request are ordered, from the first one given in a method invocation or function call to the last (that is, left to right in code).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSObjCRuntime.h`

NSDecimal

Used to describe a decimal number.

```
typedef struct {
    signed int _exponent:8;
    unsigned int _length:4;
    unsigned int _isNegative:1;
    unsigned int _isCompact:1;
    unsigned int _reserved:18;
    unsigned short _mantissa[NSDecimalMaxSize];
} NSDecimal;
```

Discussion
The fields of `NSDecimal` are private.

Used by the functions described in "Decimals" (page 2148).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSDecimal.h`

NSHashEnumerator

Allows successive elements of a hash table to be returned each time this structure is passed to `NSNextHashEnumeratorItem` (page 2223).

```
typedef struct {
    unsigned _pi;
    unsigned _si void *_bs;
} NSHashEnumerator;
```

Discussion
The fields of `NSHashEnumerator` are private.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSHashTable.h`

NSHashTable

The opaque data type used by the functions described in "Hash Tables" (page 2149).

```
typedef struct _NSHashTable NSHashTable;
```

Discussion
For Mac OS X v10.5 and later, see also `NSHashTable`.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
NSHashTable.h

NSHashTableCallBacks

Defines a structure that contains the function pointers used to configure behavior of `NSHashTable` with respect to elements within a hash table.

```
typedef struct {
    unsigned (*hash)(NSHashTable *table, const void *);
    BOOL (*isEqual)(NSHashTable *table, const void *, const void  *);
    void (*retain)(NSHashTable *table, const void *);
    void (*release)(NSHashTable *table, void *);
    NSString *(*describe)(NSHashTable *table, const void *);
} NSHashTableCallBacks;
```

Fields

`hash`
: Points to the function that must produce hash code for elements of the hash table. If `NULL`, the pointer value is used as the hash code. Second parameter is the element for which hash code should be produced.

`isEqual`
: Points to the function that compares second and third parameters. If `NULL`, then == is used for comparison.

`retain`
: Points to the function that increments a reference count for the given element. If `NULL`, then nothing is done for reference counting.

`release`
: Points to the function that decrements a reference count for the given element, and if the reference count becomes 0, frees the given element. If `NULL`, then nothing is done for reference counting or releasing.

`describe`
: Points to the function that produces an autoreleased NSString * describing the given element. If `NULL`, then the hash table produces a generic string description.

Discussion
All functions must know the types of things in the hash table to be able to operate on them. Sets of predefined call backs are described in "NSHashTable Callbacks" (page 2299).

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSHashTable.h

NSHashTableOptions

Specifies a bitfield used to configure the behavior of elements in an instance of `NSHashTable`.

```
typedef NSUInteger NSHashTableOptions
```

Declared In
NSHashTable.h

NSInteger

Used to describe an integer.

```
#if __LP64__
typedef long NSInteger;
#else
typedef int NSInteger;
endif
```

Discussion
When building 32-bit applications, NSInteger is a 32-bit integer. A 64-bit application treats NSInteger as a 64-bit integer.

Availability
Available in Mac OS X v10.5 and later.

Declared In
IKPictureTaker.h

NSMapEnumerator

Allows successive elements of a map table to be returned each time this structure is passed to `NSNextMapEnumeratorPair` (page 2223).

```
typedef struct {
    unsigned _pi;
    unsigned _si;
    void *_bs;
} NSMapEnumerator;
```

Discussion
The fields of `NSMapEnumerator` are private.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSMapTable.h

NSMapTable

The opaque data type used by the functions described in "Map Tables" (page 2150).

```
typedef struct _NSMapTable NSMapTable;
```

Discussion
For Mac OS X v10.5 and later, see also `NSMapTable`.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`NSMapTable.h`

NSMapTableKeyCallBacks

The function pointers used to configure behavior of `NSMapTable` with respect to key elements within a map table.

```
typedef struct {
    unsigned (*hash)(NSMapTable *table, const void *);
    BOOL (*isEqual)(NSMapTable *table, const void *, const void  *);
    void (*retain)(NSMapTable *table, const void *);
    void (*release)(NSMapTable *table, void *);
    NSString *(*describe)(NSMapTable *table, const void *);
    const void *notAKeyMarker;
} NSMapTableKeyCallBacks;
```

Fields

`hash`
: Points to the function which must produce hash code for key elements of the map table. If `NULL`, the pointer value is used as the hash code. Second parameter is the element for which hash code should be produced.

`isEqual`
: Points to the function which compares second and third parameters. If `NULL`, then == is used for comparison.

`retain`
: Points to the function which increments a reference count for the given element. If `NULL`, then nothing is done for reference counting.

`release`
: Points to the function which decrements a reference count for the given element, and if the reference count becomes zero, frees the given element. If `NULL`, then nothing is done for reference counting or releasing.

`describe`
: Points to the function which produces an autoreleased NSString * describing the given element. If `NULL`, then the map table produces a generic string description.

`notAKeyMarker`
: No key put in map table can be this value. An exception is raised if attempt is made to use this value as a key

Discussion
All functions must know the types of things in the map table to be able to operate on them. Sets of predefined call backs are described in "NSMapTable Key Call Backs" (page 2300).

Two predefined values to use for `notAKeyMarker` are `NSNotAnIntMapKey` and `NSNotAPointerMapKey`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSMapTable.h

NSMapTableOptions

Specifies a bitfield used to configure the behavior of elements in an instance of `NSMapTable`.

```
typedef NSUInteger NSMapTableOptions
```

Declared In
NSMapTable.h

NSMapTableValueCallBacks

The function pointers used to configure behavior of `NSMapTable` with respect to value elements within a map table.

```
typedef struct {
    void (*retain)(NSMapTable *table, const void *);
    void (*release)(NSMapTable *table, void *);
    NSString *(*describe)(NSMapTable *table, const void *);
} NSMapTableValueCallBacks;
```

Fields

`retain`
: Points to the function that increments a reference count for the given element. If `NULL`, then nothing is done for reference counting.

`release`
: Points to the function that decrements a reference count for the given element, and if the reference count becomes zero, frees the given element. If `NULL`, then nothing is done for reference counting or releasing.

`describe`
: Points to the function that produces an autoreleased NSString * describing the given element. If `NULL`, then the map table produces a generic string description.

Discussion
All functions must know the types of things in the map table to be able to operate on them. Sets of predefined call backs are described in "NSMapTable Value Callbacks" (page 2302).

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSMapTable.h

NSObjCValue

This structure is defined for use by `NSInvocation`—you should not use it directly.

```
typedef struct {
    enum _NSObjCValueType type;
    union {
        char charValue;
        short shortValue;
        long longValue;
        long long longlongValue;
        float floatValue;
        double doubleValue;
        bool boolValue;
        SEL selectorValue;
        id objectValue;
        void *pointerValue;
        void *structLocation;
        char *cStringLocation;
    } value;
} NSObjCValue;
```

Discussion
The fields of `NSObjCValue` are private.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSInvocation.h`

NSPoint

Represents a point in a Cartesian coordinate system.

```
typedef struct _NSPoint {
    CGFloat x;
    CGFloat y;
} NSPoint;
```

Fields

`x`
: The x coordinate.

`y`
: The y coordinate.

Special Considerations

Prior to Mac OS X v10.5 the coordinates were represented by `float` values rather than `CGFloat` values.

When building for 64 bit systems, or building 32 bit like 64 bit, `NSPoint` is typedef'd to `CGPoint`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSGeometry.h`

NSPointArray

Type indicating a parameter is array of `NSPoint` structures.

```
typedef NSPoint *NSPointArray;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSGeometry.h`

NSPointPointer

Type indicating a parameter is a pointer to an `NSPoint` structure.

```
typedef NSPoint *NSPointPointer;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSGeometry.h`

NSRange

A structure used to describe a portion of a series—such as characters in a string or objects in an `NSArray` object.

```
typedef struct _NSRange {
    NSUInteger location;
    NSUInteger length;
} NSRange;
```

Fields

`location`
The start index (`0` is the first, as in C arrays).

`length`
The number of items in the range (can be `0`).

Discussion
Foundation functions that operate on ranges include the following:

- `NSEqualRanges` (page 2191)
- `NSIntersectionRange` (page 2203)
- `NSLocationInRange` (page 2211)
- `NSMakeRange` (page 2214)
- `NSMaxRange` (page 2219)
- `NSRangeFromString` (page 2229)
- `NSStringFromRange` (page 2240)
- `NSUnionRange` (page 2256)

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSRange.h`

NSRangePointer

Type indicating a parameter is a pointer to an `NSRange` structure.

```
typedef NSRange *NSRangePointer;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSRange.h`

NSRect

Represents a rectangle.

```
typedef struct _NSRect {
    NSPoint origin;
    NSSize size;
} NSRect;
```

Fields

`origin`
The origin of the rectangle (its starting x coordinate and y coordinate).

`size`
The width and height of the rectangle, as measured from the origin.

Special Considerations

When building for 64 bit systems, or building 32 bit like 64 bit, `NSRect` is typedef'd to `CGRect`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSGeometry.h`

NSRectArray

Type indicating a parameter is array of `NSRect` structures.

```
typedef NSRect *NSRectArray;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSGeometry.h`

NSRectEdge

Identifiers used by `NSDivideRect` (page 2188) to specify the edge of the input rectangle from which the division is measured.

```
typedef enum _NSRectEdge {
    NSMinXEdge = 0,
    NSMinYEdge = 1,
    NSMaxXEdge = 2,
    NSMaxYEdge = 3
} NSRectEdge;
```

Constants

`NSMinXEdge`

Specifies the left edge of the input rectangle.

The input rectangle is divided vertically, and the leftmost rectangle with the width of `amount` is placed in `slice`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `NSGeometry.h`.

`NSMinYEdge`

Specifies the bottom edge of the input rectangle.

The input rectangle is divided horizontally, and the bottom rectangle with the height of `amount` is placed in `slice`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `NSGeometry.h`.

`NSMaxXEdge`

Specifies the right edge of the input rectangle.

The input rectangle is divided vertically, and the rightmost rectangle with the width of `amount` is placed in `slice`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `NSGeometry.h`.

`NSMaxYEdge`

Specifies the top edge of the input rectangle.

The input rectangle is divided horizontally, and the top rectangle with the height of `amount` is placed in `slice`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `NSGeometry.h`.

Discussion

The parameters `amount` and `slice` are defined by `NSDivideRect` (page 2188).

Availability

Available in Mac OS X v10.0 and later.

Declared In
NSGeometry.h

NSRectPointer

Type indicating a parameter is a pointer to an `NSRect` structure.

```
typedef NSRect *NSRectPointer;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSGeometry.h

NSSearchPathDirectory

These constants specify the location of a variety of directories.

```
typedef enum {
    NSApplicationDirectory = 1,
    NSDemoApplicationDirectory,
    NSDeveloperApplicationDirectory,
    NSAdminApplicationDirectory,
    NSLibraryDirectory,
    NSDeveloperDirectory,
    NSUserDirectory,
    NSDocumentationDirectory,
    NSDocumentDirectory,
    NSCoreServiceDirectory,
    NSDesktopDirectory = 12,
    NSCachesDirectory = 13,
    NSApplicationSupportDirectory = 14,
    NSDownloadsDirectory = 15,
    NSAllApplicationsDirectory = 100,
    NSAllLibrariesDirectory = 101
} NSSearchPathDirectory;
```

Constants

`NSApplicationDirectory`
Supported applications (`/Applications`).
Available in Mac OS X v10.0 and later.
Declared in `NSPathUtilities.h`.

`NSDemoApplicationDirectory`
Unsupported applications and demonstration versions.
Available in Mac OS X v10.0 and later.
Declared in `NSPathUtilities.h`.

`NSDeveloperApplicationDirectory`
Developer applications (`/Developer/Applications`).
Available in Mac OS X v10.0 and later.
Declared in `NSPathUtilities.h`.

`NSAdminApplicationDirectory`
: System and network administration applications.

Available in Mac OS X v10.0 and later.

Declared in `NSPathUtilities.h`.

`NSLibraryDirectory`
: Various user-visible documentation, support, and configuration files (`/Library`).

Available in Mac OS X v10.0 and later.

Declared in `NSPathUtilities.h`.

`NSDeveloperDirectory`
: Developer resources (`/Developer`).

Available in Mac OS X v10.0 and later.

Declared in `NSPathUtilities.h`.

`NSUserDirectory`
: User home directories (`/Users`).

Available in Mac OS X v10.0 and later.

Declared in `NSPathUtilities.h`.

`NSDocumentationDirectory`
: Documentation.

Available in Mac OS X v10.0 and later.

Declared in `NSPathUtilities.h`.

`NSDocumentDirectory`
: Document directory.

Available in Mac OS X v10.2 and later.

Declared in `NSPathUtilities.h`.

`NSCoreServiceDirectory`
: Location of core services (`System/Library/CoreServices`).

Available in Mac OS X v10.4 and later.

Declared in `NSPathUtilities.h`.

`NSDesktopDirectory`
: Location of user's desktop directory.

Available in Mac OS X v10.4 and later.

Declared in `NSPathUtilities.h`.

`NSCachesDirectory`
: Location of discardable cache files (`Library/Caches`).

Available in Mac OS X v10.4 and later.

Declared in `NSPathUtilities.h`.

`NSApplicationSupportDirectory`
: Location of application support files (`Library/Application Support`).

Available in Mac OS X v10.4 and later.

Declared in `NSPathUtilities.h`.

`NSDownloadsDirectory`
: Location of the user's downloads directory.

  The `NSDownloadsDirectory` flag will only produce a path only when the `NSUserDomainMask` is provided.

  Available in Mac OS X v10.5 and later.

  Declared in `NSPathUtilities.h`.

`NSAllApplicationsDirectory`
: All directories where applications can occur.

  Available in Mac OS X v10.0 and later.

  Declared in `NSPathUtilities.h`.

`NSAllLibrariesDirectory`
: All directories where resources can occur.

  Available in Mac OS X v10.0 and later.

  Declared in `NSPathUtilities.h`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSPathUtilities.h`

NSSearchPathDomainMask

Search path domain constants specifying base locations for the NSSearchPathDirectory (page 2278) type.

```
typedef enum {
    NSUserDomainMask = 1,
    NSLocalDomainMask = 2,
    NSNetworkDomainMask = 4,
    NSSystemDomainMask = 8,
    NSAllDomainsMask = 0x0ffff,
} NSSearchPathDomainMask;
```

Constants
`NSUserDomainMask`
: The user's home directory—the place to install user's personal items (~).

  Available in Mac OS X v10.0 and later.

  Declared in `NSPathUtilities.h`.

`NSLocalDomainMask`
: Local to the current machine—the place to install items available to everyone on this machine.

  Available in Mac OS X v10.0 and later.

  Declared in `NSPathUtilities.h`.

`NSNetworkDomainMask`
: Publicly available location in the local area network—the place to install items available on the network (`/Network`).

  Available in Mac OS X v10.0 and later.

  Declared in `NSPathUtilities.h`.

`NSSystemDomainMask`
: Provided by Apple — can't be modified (`/System`) .

Available in Mac OS X v10.0 and later.

Declared in `NSPathUtilities.h`.

`NSAllDomainsMask`
: All domains.

Includes all of the above and future items.

Available in Mac OS X v10.0 and later.

Declared in `NSPathUtilities.h`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSPathUtilities.h`

NSSize

Represents a two-dimensional size.

```
typedef struct _NSSize {
    CGFloat width;
    CGFloat height;
} NSSize;
```

Fields
`width`
: The width.

`height`
: The height.

Discussion
Normally, the values of `width` and `height` are non-negative. The functions that create an `NSSize` structure do not prevent you from setting a negative value for these attributes. If the value of `width` or `height` is negative, however, the behavior of some methods may be undefined.

Special Considerations

Prior to Mac OS X v10.5 the width and height were represented by `float` values rather than `CGFloat` values.

When building for 64 bit systems, or building 32 bit like 64 bit, `NSSize` is typedef'd to `CGSize`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSGeometry.h`

NSSizeArray

Type indicating a parameter is array of `NSSize` structures.

```
typedef NSSize *NSSizeArray;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSGeometry.h`

NSSizePointer

Type indicating parameter is a pointer to an `NSSize` structure.

```
typedef NSSize *NSSizePointer;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSGeometry.h`

NSStringEncoding

Type representing string-encoding values.

```
typedef NSUInteger NSStringEncoding;
```

Discussion
See String Encodings (page 1619) for a list of values.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSString.h`

NSSwappedDouble

Opaque structure containing endian-independent `double` value.

```
typedef struct {
    unsigned long long v;
} NSSwappedDouble;
```

Discussion
The fields of an `NSSwappedDouble` are private.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSByteOrder.h`

NSSwappedFloat

Opaque type containing an endian-independent `float` value.

```
typedef struct {
    unsigned long v;
} NSSwappedFloat;
```

Discussion
The fields of an `NSSwappedFloat` are private.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSByteOrder.h`

NSTimeInterval

Used to specify a time interval, in seconds.

```
typedef double NSTimeInterval;
```

Discussion
`NSTimeInterval` is always specified in seconds; it yields sub-millisecond precision over a range of 10,000 years.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSDate.h`

NSUncaughtExceptionHandler

Used for the function handling exceptions outside of an exception-handling domain.

```
typedef volatile void NSUncaughtExceptionHandler(NSException *exception);
```

Discussion
You can set exception handlers using `NSSetUncaughtExceptionHandler` (page 2235).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSException.h`

NSUInteger

Used to describe an unsigned integer.

```
#if __LP64__
typedef long NSUInteger;
#else
typedef int NSUInteger;
endif
```

Discussion
When building 32-bit applications, NSUInteger is a 32-bit unsigned integer. A 64-bit application treats NSUInteger as a 64-bit unsigned integer

Availability
Available in Mac OS X v10.5 and later.

Declared In
`QTKitDefines.h`

NSZone

Used to identify and manage memory zones.

```
typedef struct _NSZone NSZone;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSZone.h`

Constants

Foundation Constants Reference

Framework: Foundation/Foundation.h

Overview

This document defines constants in the Foundation framework that are not associated with a particular class.

Constants

Enumerations

NSNotFound

Defines a value that indicates that an item requested couldn't be found or doesn't exist.

```
enum {
   NSNotFound = 0x7fffffff
};
```

Constants

`NSNotFound`

A value that indicates that an item requested couldn't be found or doesn't exist.

Available in Mac OS X v10.0 and later.

Declared in `NSObjCRuntime.h`.

Discussion

`NSNotFound` is typically used by various methods and functions that search for items in serial data and return indices, such as characters in a string object or `id`s in an `NSArray` object.

Declared In

`NSObjCRuntime.h`

Memory Allocation

These constants are used as components in a bitfield to specify the behavior of `NSAllocateCollectable` (page 2158) and `NSReallocateCollectable` (page 2229).

```
enum {
    NSScannedOption = (1<<0),
    NSCollectorDisabledOption = (2<<0),
};
```

Constants

`NSScannedOption`

Specifies allocation of scanned memory.

Available in Mac OS X v10.4 and later.

Declared in `NSZone.h`.

`NSCollectorDisabledOption`

Specifies that the block is retained, and therefore ineligible for collection.

Available in Mac OS X v10.5 and later.

Declared in `NSZone.h`.

Declared In

`NSGarbageCollector.h`

NSError Codes

`NSError` codes in the Cocoa error domain.

```
enum {
    NSFileNoSuchFileError = 4,
    NSFileLockingError = 255,
    NSFileReadUnknownError = 256,
    NSFileReadNoPermissionError = 257,
    NSFileReadInvalidFileNameError = 258,
    NSFileReadCorruptFileError = 259,
    NSFileReadNoSuchFileError = 260,
    NSFileReadInapplicableStringEncodingError = 261,
    NSFileReadUnsupportedSchemeError = 262,
    NSFileWriteUnknownError = 512,
    NSFileWriteNoPermissionError = 513,
    NSFileWriteInvalidFileNameError = 514,
    NSFileWriteInapplicableStringEncodingError = 517,
    NSFileWriteUnsupportedSchemeError = 518,
    NSFileWriteOutOfSpaceError = 640,
    NSKeyValueValidationError = 1024,
    NSFormattingError = 2048,
    NSUserCancelledError = 3072,

    NSFileErrorMinimum = 0,
    NSFileErrorMaximum = 1023,
    NSValidationErrorMinimum = 1024,
    NSValidationErrorMaximum = 2047,
    NSFormattingErrorMinimum = 2048,
    NSFormattingErrorMaximum = 2559,
```

```
    NSExecutableErrorMinimum = 3584,
    NSExecutableNotLoadableError = 3584,
    NSExecutableArchitectureMismatchError = 3585,
    NSExecutableRuntimeMismatchError = 3586,
    NSExecutableLoadError = 3587,
    NSExecutableLinkError = 3588,
    NSExecutableErrorMaximum = 3839
}
```

Constants

`NSFileNoSuchFileError`
File-system operation attempted on non-existent file

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSFileLockingError`
Failure to get a lock on file

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSFileReadUnknownError`
Read error, reason unknown

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSFileReadNoPermissionError`
Read error because of a permission problem

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSFileReadInvalidFileNameError`
Read error because of an invalid file name

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSFileReadCorruptFileError`
Read error because of a corrupted file, bad format, or similar reason

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSFileReadNoSuchFileError`
Read error because no such file was found

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSFileReadInapplicableStringEncodingError`
Read error because the string encoding was not applicable.

Access the bad encoding from the `userInfo` dictionary using the `NSStringEncodingErrorKey` key.

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSFileReadUnsupportedSchemeError`
Read error because the specified URL scheme is unsupported

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSFileWriteUnknownError`
Write error, reason unknown

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSFileWriteNoPermissionError`
Write error because of a permission problem

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSFileWriteInvalidFileNameError`
Write error because of an invalid file name

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSFileWriteInapplicableStringEncodingError`
Write error because the string encoding was not applicable.

Access the bad encoding from the `userInfo` dictionary using the `NSStringEncodingErrorKey` key.

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSFileWriteUnsupportedSchemeError`
Write error because the specified URL scheme is unsupported

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSFileWriteOutOfSpaceError`
Write error because of a lack of disk space

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSKeyValueValidationError`
Key-value coding validation error

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSFormattingError`
Formatting error (related to display of data)

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSUserCancelledError`
The user cancelled the operation (for example, by pressing Command-period).

This code is for errors that do not require a dialog displayed and might be candidates for special-casing.

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSFileErrorMinimum`
Marks the start of the range of error codes reserved for file errors

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSFileErrorMaximum`
Marks the end of the range of error codes reserved for file errors

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSValidationErrorMinimum`
Marks the start of the range of error codes reserved for validation errors.

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSValidationErrorMaximum`
Marks the start and end of the range of error codes reserved for validation errors.

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSFormattingErrorMinimum`
Marks the start of the range of error codes reserved for formatting errors.

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSFormattingErrorMaximum`
Marks end of the range of error codes reserved for formatting errors.

Available in Mac OS X v10.4 and later.

Declared in `FoundationErrors.h`.

`NSExecutableErrorMinimum`
Marks beginning of the range of error codes reserved for errors related to executable files.

Available in Mac OS X v10.5 and later.

Declared in `FoundationErrors.h`.

`NSExecutableNotLoadableError`
Executable is of a type that is not loadable in the current process.

Available in Mac OS X v10.5 and later.

Declared in `FoundationErrors.h`.

`NSExecutableArchitectureMismatchError`
Executable does not provide an architecture compatible with the current process.

Available in Mac OS X v10.5 and later.

Declared in `FoundationErrors.h`.

`NSExecutableRuntimeMismatchError`
Executable has Objective C runtime information incompatible with the current process.

Available in Mac OS X v10.5 and later.

Declared in `FoundationErrors.h`.

`NSExecutableLoadError`
: Executable cannot be loaded for some other reason, such as a problem with a library it depends on.

  Available in Mac OS X v10.5 and later.

  Declared in `FoundationErrors.h`.

`NSExecutableLinkError`
: Executable fails due to linking issues.

  Available in Mac OS X v10.5 and later.

  Declared in `FoundationErrors.h`.

`NSExecutableErrorMaximum`
: Marks end of the range of error codes reserved for errors related to executable files.

  Available in Mac OS X v10.5 and later.

  Declared in `FoundationErrors.h`.

Discussion
The constants in this enumeration are `NSError` code numbers in the Cocoa error domain (`NSCocoaErrorDomain`). Other frameworks, most notably the Application Kit, provide their own `NSCocoaErrorDomain` error codes.

The enumeration constants beginning with `NSFile` indicate file-system errors or errors related to file I/O operations. Use the key `NSFilePathErrorKey` or the `NSURLErrorKey` (whichever is appropriate) to access the file-system path or URL in the `userInfo` dictionary of the `NSError` object.

Declared In
`FoundationErrors.h`

URL Loading System Error Codes

These values are returned as the error code property of an `NSError` object with the domain "NSURLErrorDomain".

```
typedef enum
{
    NSURLErrorUnknown = -1,
    NSURLErrorCancelled = -999,
    NSURLErrorBadURL = -1000,
    NSURLErrorTimedOut = -1001,
    NSURLErrorUnsupportedURL = -1002,
    NSURLErrorCannotFindHost = -1003,
    NSURLErrorCannotConnectToHost = -1004,
    NSURLErrorDataLengthExceedsMaximum = -1103,
    NSURLErrorNetworkConnectionLost = -1005,
    NSURLErrorDNSLookupFailed = -1006,
    NSURLErrorHTTPTooManyRedirects = -1007,
    NSURLErrorResourceUnavailable = -1008,
    NSURLErrorNotConnectedToInternet = -1009,
    NSURLErrorRedirectToNonExistentLocation = -1010,
    NSURLErrorBadServerResponse = -1011,
    NSURLErrorUserCancelledAuthentication = -1012,
    NSURLErrorUserAuthenticationRequired = -1013,
    NSURLErrorZeroByteResource = -1014,
    NSURLErrorFileDoesNotExist = -1100,
    NSURLErrorFileIsDirectory = -1101,
    NSURLErrorNoPermissionsToReadFile = -1102,
    NSURLErrorSecureConnectionFailed = -1200,
    NSURLErrorServerCertificateHasBadDate = -1201,
    NSURLErrorServerCertificateUntrusted = -1202,
    NSURLErrorServerCertificateHasUnknownRoot = -1203,
    NSURLErrorServerCertificateNotYetValid = -1204,
    NSURLErrorClientCertificateRejected = -1205,
    NSURLErrorCannotLoadFromNetwork = -2000,
    NSURLErrorCannotCreateFile = -3000,
    NSURLErrorCannotOpenFile = -3001,
    NSURLErrorCannotCloseFile = -3002,
    NSURLErrorCannotWriteToFile = -3003,
    NSURLErrorCannotRemoveFile = -3004,
    NSURLErrorCannotMoveFile = -3005,
    NSURLErrorDownloadDecodingFailedMidStream = -3006,
    NSURLErrorDownloadDecodingFailedToComplete = -3007
}
```

Constants

`NSURLErrorUnknown`

Returned when the URL Loading system encounters an error that it cannot interpret.

This can occur when an error originates from a lower level framework or library. Whenever this error code is received, it is a bug, and should be reported to Apple.

Available in Mac OS X v10.2 and later.

Declared in `NSURLError.h`.

`NSURLErrorCancelled`

Returned when an asynchronous load is canceled.

A Web Kit framework delegate will receive this error when it performs a cancel operation on a loading resource. Note that an `NSURLConnection` or `NSURLDownload` delegate will not receive this error if the download is canceled.

Available in Mac OS X v10.2 and later.

Declared in `NSURLError.h`.

`NSURLErrorBadURL`
: Returned when a URL is sufficiently malformed that a URL request cannot be initiated

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorTimedOut`
: Returned when an asynchronous operation times out.

  `NSURLConnection` will send this error to its delegate when the timeoutInterval in NSURLRequest expires before a load can complete.

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorUnsupportedURL`
: Returned when a properly formed URL cannot be handled by the framework.

  The most likely cause is that there is no available protocol handler for the URL.

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorCannotFindHost`
: Returned when the host name for a URL cannot be resolved.

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorCannotConnectToHost`
: Returned when an attempt to connect to a host has failed.

  This can occur when a host name resolves, but the host is down or may not be accepting connections on a certain port.

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorDataLengthExceedsMaximum`
: Returned when the length of the resource data exceeds the maximum allowed.

  Available in Mac OS X v10.5 and later.

  Declared in `NSURLError.h`.

`NSURLErrorNetworkConnectionLost`
: Returned when a client or server connection is severed in the middle of an in-progress load.

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorDNSLookupFailed`
: See `NSURLErrorCannotFindHost`

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorHTTPTooManyRedirects`
: Returned when a redirect loop is detected or when the threshold for number of allowable redirects has been exceeded (currently 16).

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorResourceUnavailable`
Returned when a requested resource cannot be retrieved.

Examples are "file not found", and data decoding problems that prevent data from being processed correctly.

Available in Mac OS X v10.2 and later.

Declared in `NSURLError.h`.

`NSURLErrorNotConnectedToInternet`
Returned when a network resource was requested, but an internet connection is not established and cannot be established automatically, either through a lack of connectivity, or by the user's choice not to make a network connection automatically.

Available in Mac OS X v10.2 and later.

Declared in `NSURLError.h`.

`NSURLErrorRedirectToNonExistentLocation`
Returned when a redirect is specified by way of server response code, but the server does not accompany this code with a redirect URL.

Available in Mac OS X v10.2 and later.

Declared in `NSURLError.h`.

`NSURLErrorBadServerResponse`
Returned when the URL Loading system receives bad data from the server.

This is equivalent to the "500 Server Error" message sent by HTTP servers.

Available in Mac OS X v10.2 and later.

Declared in `NSURLError.h`.

`NSURLErrorUserCancelledAuthentication`
Returned when an asynchronous request for authentication is cancelled by the user.

This is typically incurred by clicking a "Cancel" button in a username/password dialog, rather than the user making an attempt to authenticate.

Available in Mac OS X v10.2 and later.

Declared in `NSURLError.h`.

`NSURLErrorUserAuthenticationRequired`
Returned when authentication is required to access a resource.

Available in Mac OS X v10.2 and later.

Declared in `NSURLError.h`.

`NSURLErrorZeroByteResource`
Returned when a server reports that a URL has a non-zero content length, but terminates the network connection "gracefully" without sending any data.

Available in Mac OS X v10.2 and later.

Declared in `NSURLError.h`.

`NSURLErrorFileDoesNotExist`
Returned when a file does not exist.

Available in Mac OS X v10.2 and later.

Declared in `NSURLError.h`.

`NSURLErrorFileIsDirectory`
: Returned when a request for an FTP file results in the server responding that the file is not a plain file, but a directory.

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorNoPermissionsToReadFile`
: Returned when a resource cannot be read due to insufficient permissions.

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorSecureConnectionFailed`
: Returned when an attempt to establish a secure connection fails for reasons which cannot be expressed more specifically.

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorServerCertificateHasBadDate`
: Returned when a server certificate has a date which indicates it has expired, or is not yet valid.

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorServerCertificateUntrusted`
: Returned when a server certificate is signed by a root server which is not trusted.

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorServerCertificateHasUnknownRoot`
: Returned when a server certificate is not signed by any root server.

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorServerCertificateNotYetValid`
: Returned when a server certificate is not yet valid.

  Available in Mac OS X v10.4 and later.

  Declared in `NSURLError.h`.

`NSURLErrorClientCertificateRejected`
: Returned when a server certificate is rejected.

  Available in Mac OS X v10.4 and later.

  Declared in `NSURLError.h`.

`NSURLErrorCannotLoadFromNetwork`
: Returned when a specific request to load an item only from the cache cannot be satisfied.

  This error is sent at the point when the library would go to the network accept for the fact that is has been blocked from doing so by the “load only from cache” directive.

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorCannotCreateFile`
: Returned when `NSURLDownload` object was unable to create the downloaded file on disk due to a I/O failure.

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorCannotOpenFile`
: Returned when `NSURLDownload` was unable to open the downloaded file on disk.

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorCannotCloseFile`
: Returned when `NSURLDownload` was unable to close the downloaded file on disk.

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorCannotWriteToFile`
: Returned when `NSURLDownload` was unable to write to the downloaded file on disk.

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorCannotRemoveFile`
: Returned when `NSURLDownload` was unable to remove a downloaded file from disk.

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorCannotMoveFile`
: Returned when `NSURLDownload` was unable to move a downloaded file on disk.

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorDownloadDecodingFailedMidStream`
: Returned when `NSURLDownload` failed to decode an encoded file during the download.

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

`NSURLErrorDownloadDecodingFailedToComplete`
: Returned when `NSURLDownload` failed to decode an encoded file after downloading.

  Available in Mac OS X v10.2 and later.

  Declared in `NSURLError.h`.

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLError.h`

Global Variables

Cocoa Error Domain

This constant defines the Cocoa error domain.

```
NSString *const NSCocoaErrorDomain;
```

Constants

`NSCocoaErrorDomain`

Application Kit and Foundation Kit errors.

Available in Mac OS X v10.4 and later.

Declared in `NSError.h`.

Declared In

`FoundationErrors.h`

NSJavaSetup Information

Keys into a dictionary providing information about the way to set up the Java virtual machine.

```
extern NSString *NSJavaClasses;
extern NSString *NSJavaRoot;
extern NSString *NSJavaPath;
extern NSString *NSJavaUserPath;
extern NSString *NSJavaLibraryPath;
extern NSString *NSJavaOwnVirtualMachine;
extern NSString *NSJavaPathSeparator;
```

Constants

`NSJavaClasses`

The classes that the virtual machine should load so that their associated frameworks will be loaded.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSJavaSetup.h`.

`NSJavaRoot`

The root of the location where the application's classes are.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSJavaSetup.h`.

`NSJavaPath`

A class path whose components will be prepended by `NSJavaRoot` if they are not absolute locations. This entry is an array.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSJavaSetup.h`.

`NSJavaUserPath`
: Another segment of the class path so that the application developer can customize where classes will be looked for.

  This path goes after the application path so that one cannot replace the classes used by the application.

  Available in Mac OS X v10.0 and later.

  Deprecated in Mac OS X v10.5.

  Declared in `NSJavaSetup.h`.

`NSJavaLibraryPath`
: The path where the runtime should look for dynamic libraries needed by Java wrappers.

  This path is an `NSArray` object.

  Available in Mac OS X v10.0 and later.

  Deprecated in Mac OS X v10.5.

  Declared in `NSJavaSetup.h`.

`NSJavaOwnVirtualMachine`
: An `NSString` object. If this string exists in the dictionary, `NSJavaSetup` attempts to create a new Java virtual machine rather than reusing the existing one. Set the value of this string to `"YES"`.

  Available in Mac OS X v10.0 and later.

  Deprecated in Mac OS X v10.5.

  Declared in `NSJavaSetup.h`.

`NSJavaPathSeparator`
: This path is not a dictionary key—it is a value indicating the separator placed between components of a pathname passed to `NSJavaSetup`.

  Available in Mac OS X v10.0 and later.

  Deprecated in Mac OS X v10.5.

  Declared in `NSJavaSetup.h`.

Declared In
`NSJavaSetup.h`

NSHashTable Callbacks

Predefined sets of callbacks for `NSHashTable`.

```
extern const NSHashTableCallBacks NSIntegerHashCallBacks;
extern const NSHashTableCallBacks NSIntHashCallBacks;
extern const NSHashTableCallBacks NSNonOwnedPointerHashCallBacks;
extern const NSHashTableCallBacks NSNonRetainedObjectHashCallBacks;
extern const NSHashTableCallBacks NSObjectHashCallBacks;
extern const NSHashTableCallBacks NSOwnedObjectIdentityHashCallBacks;
extern const NSHashTableCallBacks NSOwnedPointerHashCallBacks;
extern const NSHashTableCallBacks NSPointerToStructHashCallBacks;
```

Constants
`NSIntegerHashCallBacks`
: For sets of `NSInteger`-sized quantities or smaller (for example, `int`, `long`, or `unichar`).

  Available in Mac OS X v10.5 and later.

  Declared in `NSHashTable.h`.

`NSIntHashCallBacks`

For sets of pointer-sized quantities or smaller (for example, `int`, `long`, or `unichar`). (**Deprecated.** Use `NSIntegerHashCallBacks` instead.)

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSHashTable.h`.

`NSNonOwnedPointerHashCallBacks`

For sets of pointers, hashed by address.

Available in Mac OS X v10.0 and later.

Declared in `NSHashTable.h`.

`NSNonRetainedObjectHashCallBacks`

For sets of objects, but without retaining/releasing.

Available in Mac OS X v10.0 and later.

Declared in `NSHashTable.h`.

`NSObjectHashCallBacks`

For sets of objects (similar to `NSSet`).

Available in Mac OS X v10.0 and later.

Declared in `NSHashTable.h`.

`NSOwnedObjectIdentityHashCallBacks`

For sets of objects, with transfer of ownership upon insertion, using pointer equality.

Available in Mac OS X v10.0 and later.

Declared in `NSHashTable.h`.

`NSOwnedPointerHashCallBacks`

For sets of pointers, with transfer of ownership upon insertion.

Available in Mac OS X v10.0 and later.

Declared in `NSHashTable.h`.

`NSPointerToStructHashCallBacks`

For sets of pointers to structs, when the first field of the struct is `int`-sized.

Available in Mac OS X v10.0 and later.

Declared in `NSHashTable.h`.

Discussion
On Mac OS X v10.5 and later, see also the `NSHashTable` class.

Note that you can make your own callback by picking fields among the above callbacks.

Declared In
`NSHashTable.h`

NSMapTable Key Call Backs

Predefined sets of callbacks for `NSMapTable` keys.

```
extern const NSMapTableKeyCallBacks NSIntegerMapKeyCallBacks;
extern const NSMapTableKeyCallBacks NSIntMapKeyCallBacks;
extern const NSMapTableKeyCallBacks NSNonOwnedPointerMapKeyCallBacks;
extern const NSMapTableKeyCallBacks  NSNonOwnedPointerOrNullMapKeyCallBacks;
extern const NSMapTableKeyCallBacks NSNonRetainedObjectMapKeyCallBacks;
extern const NSMapTableKeyCallBacks NSObjectMapKeyCallBacks;
extern const NSMapTableKeyCallBacks NSOwnedPointerMapKeyCallBacks;
```

Constants

`NSIntegerMapKeyCallBacks`

For keys that are pointer-sized quantities or smaller (for example, `int`, `long`, or `unichar`).

Available in Mac OS X v10.5 and later.

Declared in `NSMapTable.h`.

`NSIntMapKeyCallBacks`

For keys that are pointer-sized quantities or smaller (for example, `int`, `long`, or `unichar`). (**Deprecated.** Use `NSIntegerMapKeyCallBacks` instead.)

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSMapTable.h`.

`NSNonOwnedPointerMapKeyCallBacks`

For keys that are pointers not freed.

Available in Mac OS X v10.0 and later.

Declared in `NSMapTable.h`.

`NSNonOwnedPointerOrNullMapKeyCallBacks`

For keys that are pointers not freed, or `NULL`.

Available in Mac OS X v10.0 and later.

Declared in `NSMapTable.h`.

`NSNonRetainedObjectMapKeyCallBacks`

For sets of objects, but without retaining/releasing.

Available in Mac OS X v10.0 and later.

Declared in `NSMapTable.h`.

`NSObjectMapKeyCallBacks`

For keys that are objects.

Available in Mac OS X v10.0 and later.

Declared in `NSMapTable.h`.

`NSOwnedPointerMapKeyCallBacks`

For keys that are pointers, with transfer of ownership upon insertion.

Available in Mac OS X v10.0 and later.

Declared in `NSMapTable.h`.

Discussion

On Mac OS X v10.5 and later, see also the `NSMapTable` class.

Note that you can make your own callback by picking fields among the above callbacks.

Declared In

`NSMapTable.h`

NSMapTable Value Callbacks

These are predefined sets of callbacks for `NSMapTable` values.

```
extern const NSMapTableValueCallBacks NSIntegerMapValueCallBacks;
extern const NSMapTableValueCallBacks NSIntMapValueCallBacks;
extern const NSMapTableValueCallBacks NSNonOwnedPointerMapValueCallBacks;
extern const NSMapTableValueCallBacks NSObjectMapValueCallBacks;
extern const NSMapTableValueCallBacks NSNonRetainedObjectMapValueCallBacks;
extern const NSMapTableValueCallBacks NSOwnedPointerMapValueCallBacks;
```

Constants

`NSIntegerMapValueCallBacks`

For values that are pointer-sized quantities, (for example, `int`, `long`, or `unichar`).

Available in Mac OS X v10.5 and later.

Declared in `NSMapTable.h`.

`NSIntMapValueCallBacks`

For values that are pointer-sized quantities, (for example, `int`, `long`, or `unichar`). (**Deprecated.** Use `NSIntegerMapValueCallBacks` instead.)

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSMapTable.h`.

`NSNonOwnedPointerMapValueCallBacks`

For values that are not owned pointers.

Available in Mac OS X v10.0 and later.

Declared in `NSMapTable.h`.

`NSOwnedPointerMapValueCallBacks`

For values that are owned pointers.

Available in Mac OS X v10.0 and later.

Declared in `NSMapTable.h`.

`NSNonRetainedObjectMapValueCallBacks`

For sets of objects, but without retaining/releasing.

Available in Mac OS X v10.0 and later.

Declared in `NSMapTable.h`.

`NSObjectMapValueCallBacks`

For values that are objects.

Available in Mac OS X v10.0 and later.

Declared in `NSMapTable.h`.

Discussion

On Mac OS X v10.5 and later, see also the `NSMapTable` class.

Note that you can make your own callback by picking fields among the above callbacks.

Declared In

`NSMapTable.h`

NSURL Domain

This error domain is defined for `NSURL`.

```
extern NSString * const NSURLErrorDomain;
```

Constants

`NSURLErrorDomain`

URL loading system errors

Available in Mac OS X v10.2 and later.

Declared in `NSURLError.h`.

Declared In

`NSURLError.h`

Zero Constants

These constants are defined as conveniences and can be used to compare with return values from functions.

```
extern const NSPoint NSZeroPoint;
extern const NSSize NSZeroSize;
extern const NSRect NSZeroRect;
```

Constants

`NSZeroPoint`

An `NSPoint` structure with both x and y coordinates set to `0`.

Available in Mac OS X v10.0 and later.

Declared in `NSGeometry.h`.

`NSZeroSize`

An `NSSize` structure set to `0` in both dimensions.

Available in Mac OS X v10.0 and later.

Declared in `NSGeometry.h`.

`NSZeroRect`

An `NSRect` structure set to `0` in width and height.

Available in Mac OS X v10.0 and later.

Declared in `NSGeometry.h`.

Declared In

`NSGeometry.h`

Numeric Constants

NSDecimal Constants

Constants used by `NSDecimal`.

```
#define NSDecimalMaxSize (8)
#define NSDecimalNoScale SHRT_MAX
```

Constants

`NSDecimalMaxSize`

The maximum size of NSDecimal (page 2269).

Gives a precision of at least 38 decimal digits, 128 binary positions.

Available in Mac OS X v10.0 and later.

Declared in `NSDecimal.h`.

`NSDecimalNoScale`

Specifies that the number of digits allowed after the decimal separator in a decimal number should not be limited.

Available in Mac OS X v10.0 and later.

Declared in `NSDecimal.h`.

Declared In

`NSDecimal.h`

NSMapTable Constants

Constants used by `NSMapTable`.

```
#define NSNotAnIntMapKey ((const void *)0x80000000)
#define NSNotAnIntegerMapKey    ((const void *)NSIntegerMin)
#define NSNotAPointerMapKey ((const void *)0xffffffff)
```

Constants

`NSNotAnIntMapKey`

Predefined `notAKeyMarker` for use with NSMapTableKeyCallBacks (page 2272). (**Deprecated.** Use `NSNotAnIntegerMapKey` instead.)

Available in Mac OS X v10.0 and later.

Declared in `NSMapTable.h`.

`NSNotAnIntegerMapKey`

Predefined `notAKeyMarker` for use with NSMapTableKeyCallBacks (page 2272).

Available in Mac OS X v10.5 and later.

Declared in `NSMapTable.h`.

`NSNotAPointerMapKey`

Predefined `notAKeyMarker` for use with NSMapTableKeyCallBacks (page 2272).

Available in Mac OS X v10.0 and later.

Declared in `NSMapTable.h`.

Discussion

On Mac OS X v10.5 and later, see also the `NSMapTable` class.

Declared In

`NSMapTable.h`

NSInteger and NSUInteger Maximum and Minimum Values

Constants representing the maximum and minimum values of `NSInteger` and `NSUInteger`.

```
#define NSIntegerMax    LONG_MAX
#define NSIntegerMin    LONG_MIN
#define NSUIntegerMax   ULONG_MAX
```

Constants

`NSIntegerMax`

The maximum value for an `NSInteger`.

Available in Mac OS X v10.5 and later.

Declared in `QTKitDefines.h`.

`NSIntegerMin`

The minimum value for an `NSInteger`.

Available in Mac OS X v10.5 and later.

Declared in `QTKitDefines.h`.

`NSUIntegerMax`

The maximum value for an `NSUInteger`.

Available in Mac OS X v10.5 and later.

Declared in `QTKitDefines.h`.

Declared In

`NSObjCRuntime.h`

Notifications

Java Setup Notification Names

Notifications sent by the Java bridge to registered observers when a virtual machine is created and initialized.

```
extern NSString *NSJavaWillSetupVirtualMachineNotification;
extern NSString *NSJavaDidSetupVirtualMachineNotification;
extern NSString *NSJavaWillCreateVirtualMachineNotification;
extern NSString *NSJavaDidCreateVirtualMachineNotification;
```

Constants

`NSJavaWillSetupVirtualMachineNotification`

Notification sent before the Java virtual machine is set up.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSJavaSetup.h`.

`NSJavaDidSetupVirtualMachineNotification`

Notification sent after the Java virtual machine is set up.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSJavaSetup.h`.

`NSJavaWillCreateVirtualMachineNotification`
Notification sent before the Java virtual machine is created.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSJavaSetup.h`.

`NSJavaDidCreateVirtualMachineNotification`
Notification sent after the Java virtual machine is created.

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared in `NSJavaSetup.h`.

Declared In
`NSJavaSetup.h`

Exceptions

General Exception Names

Exceptions defined by `NSException`.

```
extern NSString *NSGenericException;
extern NSString *NSRangeException;
extern NSString *NSInvalidArgumentException;
extern NSString *NSInternalInconsistencyException;
extern NSString *NSMallocException;
extern NSString *NSObjectInaccessibleException;
extern NSString *NSObjectNotAvailableException;
extern NSString *NSDestinationInvalidException;
extern NSString *NSPortTimeoutException;
extern NSString *NSInvalidSendPortException;
extern NSString *NSInvalidReceivePortException;
extern NSString *NSPortSendException;
extern NSString *NSPortReceiveException;
extern NSString *NSOldStyleException;
```

Constants
`NSGenericException`
A generic name for an exception.

You should typically use a more specific exception name.

Available in Mac OS X v10.0 and later.

Declared in `NSException.h`.

`NSRangeException`
Name of an exception that occurs when attempting to access outside the bounds of some data, such as beyond the end of a string.

Available in Mac OS X v10.0 and later.

Declared in `NSException.h`.

`NSInvalidArgumentException`
Name of an exception that occurs when you pass an invalid argument to a method, such as a `nil` pointer where a non-`nil` object is required.

Available in Mac OS X v10.0 and later.

Declared in `NSException.h`.

`NSInternalInconsistencyException`
Name of an exception that occurs when an internal assertion fails and implies an unexpected condition within the called code.

Available in Mac OS X v10.0 and later.

Declared in `NSException.h`.

`NSMallocException`
Obsolete; not currently used.

Available in Mac OS X v10.0 and later.

Declared in `NSException.h`.

`NSObjectInaccessibleException`
Name of an exception that occurs when a remote object is accessed from a thread that should not access it.

See `NSConnection`'s `enableMultipleThreads` (page 336).

Available in Mac OS X v10.0 and later.

Declared in `NSException.h`.

`NSObjectNotAvailableException`
Name of an exception that occurs when the remote side of the `NSConnection` refused to send the message to the object because the object has never been vended.

Available in Mac OS X v10.0 and later.

Declared in `NSException.h`.

`NSDestinationInvalidException`
Name of an exception that occurs when an internal assertion fails and implies an unexpected condition within the distributed objects.

This is a distributed objects–specific exception.

Available in Mac OS X v10.0 and later.

Declared in `NSException.h`.

`NSPortTimeoutException`
Name of an exception that occurs when a timeout set on a port expires during a send or receive operation.

This is a distributed objects–specific exception.

Available in Mac OS X v10.0 and later.

Declared in `NSException.h`.

`NSInvalidSendPortException`
Name of an exception that occurs when the send port of an `NSConnection` has become invalid.

This is a distributed objects–specific exception.

Available in Mac OS X v10.0 and later.

Declared in `NSException.h`.

`NSInvalidReceivePortException`
: Name of an exception that occurs when the receive port of an `NSConnection` has become invalid.

This is a distributed objects–specific exception.

Available in Mac OS X v10.0 and later.

Declared in `NSException.h`.

`NSPortSendException`
: Generic error occurred on send.

This is an `NSPort`-specific exception.

Available in Mac OS X v10.0 and later.

Declared in `NSException.h`.

`NSPortReceiveException`
: Generic error occurred on receive.

This is an `NSPort`-specific exception.

Available in Mac OS X v10.0 and later.

Declared in `NSException.h`.

`NSOldStyleException`
: No longer used.

Available in Mac OS X v10.0 and later.

Declared in `NSException.h`.

Declared In
`NSException.h`

Version Numbers

Foundation Version Number

Version of the Foundation framework in the current environment.

```
double NSFoundationVersionNumber;
```

Constants
`NSFoundationVersionNumber`
: The version of the Foundation framework in the current environment.

Available in Mac OS X v10.1 and later.

Declared in `NSObjCRuntime.h`.

Declared In
`NSObjCRuntime.h`

Foundation Framework Version Numbers

Constants to define Foundation Framework version numbers.

```
#define NSFoundationVersionNumber10_0    397.40
#define NSFoundationVersionNumber10_1    425.00
#define NSFoundationVersionNumber10_1_1    425.00
#define NSFoundationVersionNumber10_1_2    425.00
#define NSFoundationVersionNumber10_1_3    425.00
#define NSFoundationVersionNumber10_1_4    425.00
#define NSFoundationVersionNumber10_2    462.00
#define NSFoundationVersionNumber10_2_1    462.00
#define NSFoundationVersionNumber10_2_2    462.00
#define NSFoundationVersionNumber10_2_3    462.00
#define NSFoundationVersionNumber10_2_4    462.00
#define NSFoundationVersionNumber10_2_5    462.00
#define NSFoundationVersionNumber10_2_6    462.00
#define NSFoundationVersionNumber10_2_7    462.70
#define NSFoundationVersionNumber10_2_8    462.70
#define NSFoundationVersionNumber10_3    500.00
#define NSFoundationVersionNumber10_3_1    500.00
#define NSFoundationVersionNumber10_3_2    500.30
#define NSFoundationVersionNumber10_3_3    500.54
#define NSFoundationVersionNumber10_3_4    500.56
#define NSFoundationVersionNumber10_3_5    500.56
#define NSFoundationVersionNumber10_3_6    500.56
#define NSFoundationVersionNumber10_3_7    500.56
#define NSFoundationVersionNumber10_3_8    500.56
#define NSFoundationVersionNumber10_3_9    500.58
#define NSFoundationVersionNumber10_4    567.00
#define NSFoundationVersionNumber10_4_1    567.00
#define NSFoundationVersionNumber10_4_2    567.12
#define NSFoundationVersionNumber10_4_3    567.21
#define NSFoundationVersionNumber10_4_4_Intel    567.23
#define NSFoundationVersionNumber10_4_4_PowerPC    567.21
#define NSFoundationVersionNumber10_4_5    567.25
#define NSFoundationVersionNumber10_4_6    567.26
#define NSFoundationVersionNumber10_4_7    567.27
#define NSFoundationVersionNumber10_4_8    567.28
#define NSFoundationVersionNumber10_4_9    567.29
#define NSFoundationVersionNumber10_4_10    567.29
#define NSFoundationVersionNumber10_4_11    567.36
```

Constants

`NSFoundationVersionNumber10_0`

Foundation version released in Mac OS X version 10.0.

Available in Mac OS X v10.1 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_1`

Foundation version released in Mac OS X version 10.1.

Available in Mac OS X v10.2 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_1_1`

Foundation version released in Mac OS X version 10.1.1.

Available in Mac OS X v10.5 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_1_2`
Foundation version released in Mac OS X version 10.1.2.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_1_3`
Foundation version released in Mac OS X version 10.1.3.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_1_4`
Foundation version released in Mac OS X version 10.1.4.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_2`
Foundation version released in Mac OS X version 10.2.
Available in Mac OS X v10.3 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_2_1`
Foundation version released in Mac OS X version 10.2.1.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_2_2`
Foundation version released in Mac OS X version 10.2.2.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_2_3`
Foundation version released in Mac OS X version 10.2.3.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_2_4`
Foundation version released in Mac OS X version 10.2.4.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_2_5`
Foundation version released in Mac OS X version 10.2.5.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_2_6`
Foundation version released in Mac OS X version 10.2.6.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_2_7`
Foundation version released in Mac OS X version 10.2.7.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_2_8`
Foundation version released in Mac OS X version 10.2.8.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_3`
Foundation version released in Mac OS X version 10.3.
Available in Mac OS X v10.4 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_3_1`
Foundation version released in Mac OS X version 10.3.1.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_3_2`
Foundation version released in Mac OS X version 10.3.2.
Available in Mac OS X v10.4 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_3_3`
Foundation version released in Mac OS X version 10.3.3.
Available in Mac OS X v10.4 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_3_4`
Foundation version released in Mac OS X version 10.3.4.
Available in Mac OS X v10.4 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_3_5`
Foundation version released in Mac OS X version 10.3.5.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_3_6`
Foundation version released in Mac OS X version 10.3.6.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_3_7`
Foundation version released in Mac OS X version 10.3.7.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_3_8`
Foundation version released in Mac OS X version 10.3.8.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_3_9`
Foundation version released in Mac OS X version 10.3.9.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4`
Foundation version released in Mac OS X version 10.4.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_1`
Foundation version released in Mac OS X version 10.4.1.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_2`
Foundation version released in Mac OS X version 10.4.2.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_3`
Foundation version released in Mac OS X version 10.4.3.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_4_Intel`
Foundation version released in Mac OS X version 10.4.4 for Intel.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_4_PowerPC`
Foundation version released in Mac OS X version 10.4.4 for PowerPC.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_5`
Foundation version released in Mac OS X version 10.4.5.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_6`
Foundation version released in Mac OS X version 10.4.6.
Available in Mac OS X v10.5 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_7`
: Foundation version released in Mac OS X version 10.4.7.

Available in Mac OS X v10.5 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_8`
: Foundation version released in Mac OS X version 10.4.8.

Available in Mac OS X v10.5 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_9`
: Foundation version released in Mac OS X version 10.4.9.

Available in Mac OS X v10.5 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_10`
: Foundation version released in Mac OS X version 10.4.10.

Available in Mac OS X v10.5 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_11`
: Foundation version released in Mac OS X version 10.4.11.

Available in Mac OS X v10.5 and later.

Declared in `NSObjCRuntime.h`.

Declared In
`NSObjCRuntime.h`

Document Revision History

This table describes the changes to *Foundation Framework Reference*.

Date	Notes
2008-06-27	Updated for iPhone OS.
2007-10-31	Updated for Mac OS X v10.5. Updated framework illustrations.
2007-04-16	Updated for Mac OS X v10.5.
2006-05-23	First publication of this content as a collection of separate documents.

Document Revision History

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z