Foundation Reference for Java

(Legacy)

Cocoa > Java

2006-07-24

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

Simultaneously published in the United States and Canada.

Contents

Figures

The Foundation Framework

Package: com.apple.cocoa.foundation

Introduction

Important: The Java API for the Foundation framework is deprecated in Mac OS X version 10.4 and later. You should use the Objective-C API, documented in *Foundation Framework Reference*, to develop Cocoa applications.

The Foundation framework defines a base layer of Java classes. In addition to providing a set of useful primitive object classes, it introduces several paradigms that define functionality not covered by the Java 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.

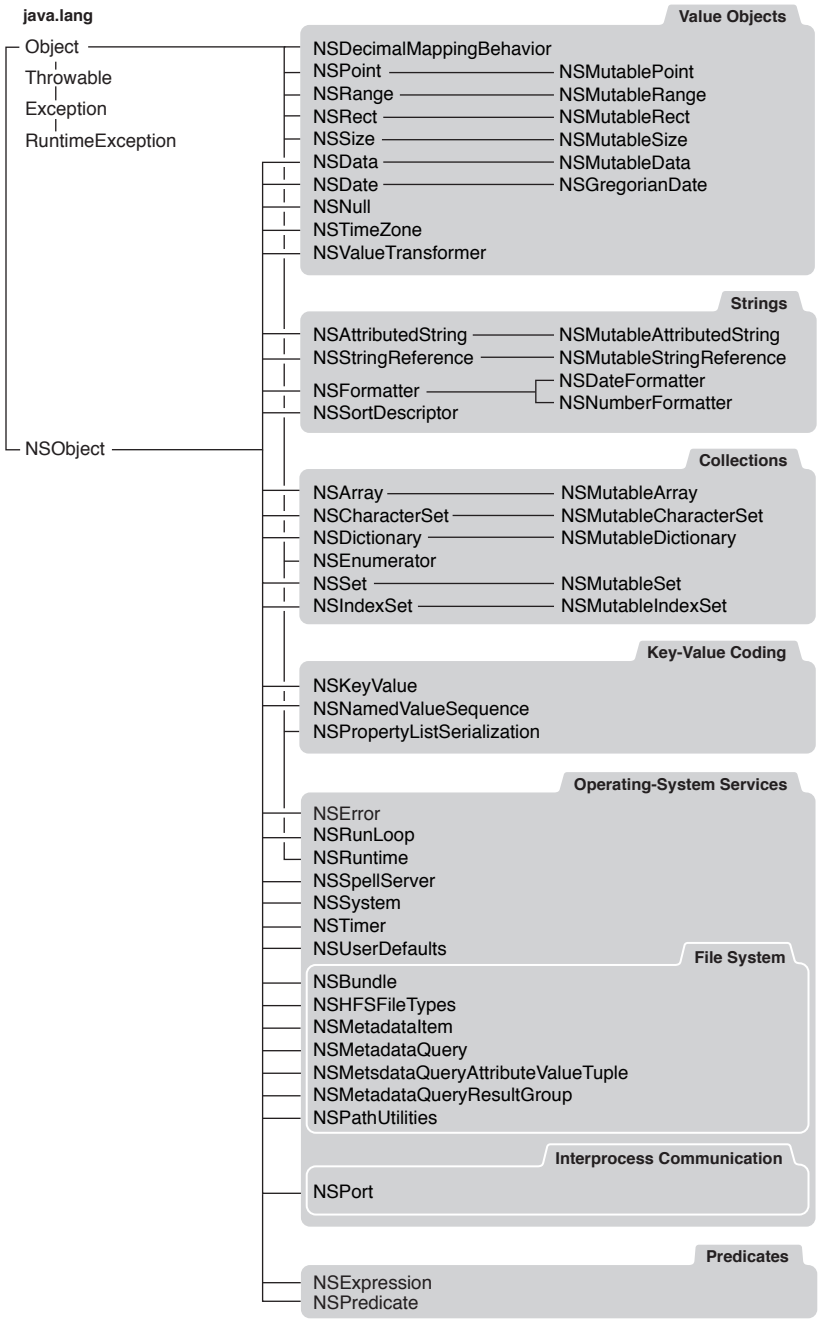
The Foundation framework includes the root object class, classes representing basic data types such as strings and byte arrays, collection classes for storing other objects, classes representing system information such as dates, and classes representing communication ports. See Figure I-1 (page 26) for 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.

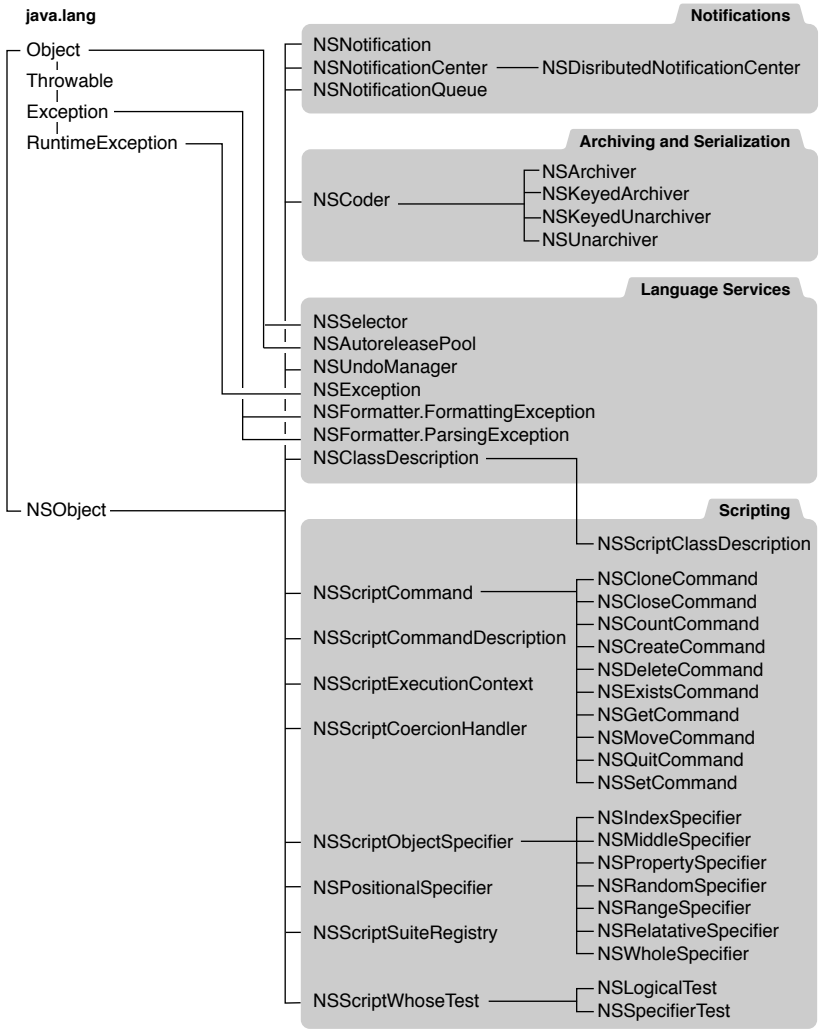
Foundation Framework Classes

The Foundation class hierarchy is rooted in the Foundation framework's NSObject class (see Figure I-1 (page 26)). The remainder of the Foundation framework consists of several related groups of classes as well as a few individual classes. NSStringReference and NSMutableStringReference, for example, act as brokers for instances of various 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.

Figure I-1 The Foundation framework class hierarchy



Java Foundation Continued



Many of these classes have closely related functionality:

- Data storage. NSData provides object-oriented storage for arrays of bytes. NSArray, NSDictionary, and NSSet provide storage for objects of any class.
- Text and strings. NSCharacterSet represents various groupings of characters that are used by the String and NSScanner classes. An NSScanner object is used to scan numbers and words from a String object.
- Dates and times. The NSDate and NSTimeZone classes store times and dates. They offer methods for calculating date and time differences, 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 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.

Classes

NSAppleEventDescriptor

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Scriptable Applications Programming Guide for Cocoa

Overview

Important: The information in this document is obsolete and should not be used for new development.

A descriptor is the basic building block for Apple events—every Apple event is a descriptor, where descriptor is a type of data structure. 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.

Cocoa supplies built-in scripting support that converts received Apple events into script commands that operate on application objects. As a result, most Cocoa applications don't need to work directly with Apple event descriptors. However, those applications that do need to construct Apple events or extract information from them can use NSAppleEventDescriptor. The most common reason to construct an Apple event is to supply information in a return event. In addition, if you execute an AppleScript script using the NSAppleScript class, you get an NSAppleEventDescriptor as the return value, from which you extract the necessary information.

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. 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.

Tasks

Constructors

`NSAppleEventDescriptor` (page 34)
Creates an empty NSAppleEventDescriptor.

Creating an Event Descriptor

`descriptorWithBoolean` (page 35)
Creates and returns a newly allocated NSAppleEventDescriptor with Apple event type `typeBoolean` and value specified by *`boolean`*.

`descriptorWithEnumCode` (page 35)
Creates and returns a newly allocated NSAppleEventDescriptor with Apple event type `typeEnumerated` and value specified by *`enumerator`*.

`descriptorWithInt32` (page 35)
Creates and returns a newly allocated NSAppleEventDescriptor with Apple event type `typeSInt32` and value specified by *`signedInt`*.

`descriptorWithString` (page 35)
Creates and returns a newly allocated NSAppleEventDescriptor with Apple event type `typeUnicodeText` and value specified by *`string`*.

`descriptorWithTypeCode` (page 35)
Creates and returns a newly allocated NSAppleEventDescriptor with Apple event type `typeType` and value specified by *`typeCode`*.

`listDescriptor` (page 36)
Creates and returns an instance of NSAppleEventDescriptor initialized as an empty list descriptor.

`nullDescriptor` (page 36)
Creates and returns an instance of NSAppleEventDescriptor with no parameter or attribute values set.

`recordDescriptor` (page 36)
Creates and returns a descriptor for an Apple event record whose data has yet to be set.

Getting Information About an Event Descriptor

`data` (page 37)
Returns the receiving descriptor's data as an NSData object.

`descriptorType` (page 38)
Returns the descriptor type for the receiving descriptor.

`coerceToDescriptorType` (page 37)
Returns an instance of NSAppleEventDescriptor coerced to the type specified by *`descType`*.

`numberOfItems` (page 39)
Returns the number of descriptors in the receiving descriptor list.

booleanValue (page 37)
Return the contents of the descriptor, after first coercing it to typeBoolean.

enumCodeValue (page 38)
Return the contents of the descriptor, after first coercing it to typeEnumerated.

int32Value (page 39)
Return the contents of the descriptor, after first coercing it to typeSInt32.

stringValue (page 41)
Return the contents of the descriptor, after first coercing it to typeUnicodeText.

typeCodeValue (page 41)
Return the contents of the descriptor, after first coercing it to typeType.

Working with List Descriptors

descriptorAtIndex (page 37)
Returns an instance of NSAppleEventDescriptor from the position specified by *anIndex*.

insertDescriptor (page 38)
Inserts the NSAppleEventDescriptor specified by *descriptor* at the position specified by *anIndex*.

removeDescriptorAtIndex (page 39)
Removes the receiver's descriptor at the position specified by *anIndex*.

Working with Record Descriptors

descriptorForKeyword (page 37)
Returns an instance of NSAppleEventDescriptor for the receiver's descriptor specified by *keyword*.

keywordForDescriptorAtIndex (page 39)
Returns the keyword for the descriptor at the position specified by *anIndex*.

removeDescriptorWithKeyword (page 40)
Removes the descriptor in the receiver identified by *keyword*.

setDescriptor (page 40)
Adds *descriptor* to the receiver identified by *keyword*.

Working with Apple Event Descriptors

attributeDescriptorForKeyword (page 36)
Returns an instance of NSAppleEventDescriptor for the attribute specified by *keyword*.

setAttributeDescriptor (page 40)
Adds *descriptor* to the receiver as an attribute identified by *keyword*.

eventClass (page 38)
Returns the event class for the receiving descriptor.

eventID (page 38)
Returns the event ID for the receiving descriptor.

paramDescriptorForKeyword (page 39)
Returns a descriptor for the receiver's Apple event parameter specified by *keyword*.

`setParamDescriptor` (page 41)
Adds *descriptor* to the receiver as an Apple event parameter identified by *keyword*.

`removeParamDescriptorWithKeyword` (page 40)
Removes the receiver's parameter descriptor identified by *keyword*.

`returnID` (page 40)
Returns the receiver's return ID (the ID for a reply Apple event).

`transactionID` (page 41)
Returns the receiver's transaction ID, if any.

Constructors

NSAppleEventDescriptor

Creates an empty NSAppleEventDescriptor.

```
NSAppleEventDescriptor()
```

Creates an NSAppleEventDescriptor object with descriptor type specified by *descType* and data specified by *data*.

```
NSAppleEventDescriptor(int descType, NSData data)
```

Creates a new NSAppleEventDescriptor object for an Apple event.

```
NSAppleEventDescriptor(int eventClass, int eventID, NSAppleEventDescriptor
    addressDescriptor, int returnID, int transactionID)
```

Discussion
Returns `null` if an error occurs. The event class and event ID for the returned descriptor are set to the values in the *eventClass* and *eventID* parameters. The *addressDescriptor* parameter supplies an Apple event descriptor that identifies the target application for the Apple event.

The *returnID* parameter specifies the return ID for the created Apple event. If you pass a value of `kAutoGenerateReturnID` (–1), 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.

The *transactionID* parameter specifies the transaction ID for the created Apple event. 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` (0) if the Apple event is not one of a series of interdependent Apple events.

The constants `kAutoGenerateReturnID` and `kAnyTransactionID` are defined in the header `AEDataModel.h` in `AE.framework`, a subframework of `ApplicationServices.framework`.

Static Methods

descriptorWithBoolean

Creates and returns a newly allocated NSAppleEventDescriptor with Apple event type `typeBoolean` and value specified by *`boolean`*.

```
public static NSAppleEventDescriptor descriptorWithBoolean(boolean boolean)
```

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

descriptorWithEnumCode

Creates and returns a newly allocated NSAppleEventDescriptor with Apple event type `typeEnumerated` and value specified by *`enumerator`*.

```
public static NSAppleEventDescriptor descriptorWithEnumCode(int enumerator)
```

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

descriptorWithInt32

Creates and returns a newly allocated NSAppleEventDescriptor with Apple event type `typeSInt32` and value specified by *`signedInt`*.

```
public static NSAppleEventDescriptor descriptorWithInt32(int signedInt)
```

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

descriptorWithString

Creates and returns a newly allocated NSAppleEventDescriptor with Apple event type `typeUnicodeText` and value specified by *`string`*.

```
public static NSAppleEventDescriptor descriptorWithString(String string)
```

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

descriptorWithTypeCode

Creates and returns a newly allocated NSAppleEventDescriptor with Apple event type `typeType` and value specified by *`typeCode`*.

```
public static NSAppleEventDescriptor descriptorWithTypeCode(int typeCode)
```

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

listDescriptor

Creates and returns an instance of NSAppleEventDescriptor initialized as an empty list descriptor.

```
public static NSAppleEventDescriptor listDescriptor()
```

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` (page 38) or remove them with `removeDescriptorAtIndex` (page 39).

nullDescriptor

Creates and returns an instance of NSAppleEventDescriptor with no parameter or attribute values set.

```
public static NSAppleEventDescriptor nullDescriptor()
```

Discussion
This method isn't typically called, as most NSAppleEventDescriptor instance methods can't be safely called on the returned descriptor.

recordDescriptor

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

```
public static NSAppleEventDescriptor recordDescriptor()
```

Discussion
A record descriptor 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` (page 40), `setDescriptor` (page 40), and `setParamDescriptor` (page 41).

Instance Methods

attributeDescriptorForKeyword

Returns an instance of NSAppleEventDescriptor for the attribute specified by *keyword*.

```
public NSAppleEventDescriptor attributeDescriptorForKeyword(int keyword)
```

Discussion
Returns `null` if any error occurs.

booleanValue

Return the contents of the descriptor, after first coercing it to `typeBoolean`.

```
public boolean booleanValue()
```

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

coerceToDescriptorType

Returns an instance of NSAppleEventDescriptor coerced to the type specified by *descType*.

```
public NSAppleEventDescriptor coerceToDescriptorType(int descType)
```

Discussion
Returns `null` if the coercion fails.

data

Returns the receiving descriptor's data as an NSData object.

```
public NSData data()
```

Discussion
Returns `null` if an error occurs.

descriptorAtIndex

Returns an instance of NSAppleEventDescriptor from the position specified by *anIndex*.

```
public NSAppleEventDescriptor descriptorAtIndex(int anIndex)
```

Discussion
NSAppleEventDescriptor indices are one-based. Returns `null` if an error occurs.

See Also
`insertDescriptor` (page 38)
`removeDescriptorAtIndex` (page 39)

descriptorForKeyword

Returns an instance of NSAppleEventDescriptor for the receiver's descriptor specified by *keyword*.

```
public NSAppleEventDescriptor descriptorForKeyword(int keyword)
```

Discussion
Returns `null` if an error occurs.

descriptorType

Returns the descriptor type for the receiving descriptor.

```
public int descriptorType()
```

enumCodeValue

Return the contents of the descriptor, after first coercing it to `typeEnumerated`.

```
public int enumCodeValue()
```

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

eventClass

Returns the event class for the receiving descriptor.

```
public int eventClass()
```

Discussion
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 the header `AERegistry.h` in `AE.framework`, a subframework of `ApplicationServices.framework`).

eventID

Returns the event ID for the receiving descriptor.

```
public int eventID()
```

Discussion
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 the header `AERegistry.h` in `AE.framework`, a subframework of `ApplicationServices.framework`).

insertDescriptor

Inserts the NSAppleEventDescriptor specified by *`descriptor`* at the position specified by *`anIndex`*.

```
public void insertDescriptor(NSAppleEventDescriptor descriptor, int anIndex)
```

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

See Also
`descriptorAtIndex` (page 37)

removeDescriptorAtIndex (page 39)

int32Value

Return the contents of the descriptor, after first coercing it to typeSInt32.

```
public int int32Value()
```

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

keywordForDescriptorAtIndex

Returns the keyword for the descriptor at the position specified by *anIndex*.

```
public int keywordForDescriptorAtIndex(int anIndex)
```

Discussion
NSAppleEventDescriptor indices are one-based. Returns the value 0 if an error occurs.

numberOfItems

Returns the number of descriptors in the receiving descriptor list.

```
public int numberOfItems()
```

Discussion
Returns the value 0 if an error occurs.

paramDescriptorForKeyword

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

```
public NSAppleEventDescriptor paramDescriptorForKeyword(int keyword)
```

Discussion
The receiver must be an Apple event. Returns null if an error occurs.

removeDescriptorAtIndex

Removes the receiver's descriptor at the position specified by *anIndex*.

```
public void removeDescriptorAtIndex(int anIndex)
```

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

See Also
descriptorAtIndex (page 37)

`insertDescriptor` (page 38)

removeDescriptorWithKeyword

Removes the descriptor in the receiver identified by *keyword*.

```
public void removeDescriptorWithKeyword(int keyword)
```

Discussion
The receiver must be an Apple event or Apple event record.

removeParamDescriptorWithKeyword

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

```
public void removeParamDescriptorWithKeyword(int keyword)
```

Discussion
The receiver must be an Apple event or Apple event record.

returnID

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

```
public int returnID()
```

Discussion
The receiver must be an Apple event. Returns the value 0 if an error occurs.

setAttributeDescriptor

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

```
public void setAttributeDescriptor(NSAppleEventDescriptor descriptor, int keyword)
```

Discussion
The receiver must be an Apple event.

setDescriptor

Adds *descriptor* to the receiver identified by *keyword*.

```
public void setDescriptor(NSAppleEventDescriptor descriptor, int keyword)
```

Discussion
The receiver must be an Apple event or Apple event record.

setParamDescriptor

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

```
public void setParamDescriptor(NSAppleEventDescriptor descriptor, int keyword)
```

Discussion
The receiver must be an Apple event or Apple event record.

stringValue

Return the contents of the descriptor, after first coercing it to `typeUnicodeText`.

```
public String stringValue()
```

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

transactionID

Returns the receiver's transaction ID, if any.

```
public int transactionID()
```

Discussion
For more information on transactions, see the description for `NSAppleEventDescriptor` (page 34).

typeCodeValue

Return the contents of the descriptor, after first coercing it to `typeType`.

```
public int typeCodeValue()
```

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

NSAppleScript

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Availability	Available in Mac OS X v10.2 and later.
Companion guide	Cocoa Scripting Guide

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 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 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 NSAppleScript object, you supply the script as a string. Should an error occur when compiling or executing the script, several of the methods fill in a dictionary containing error information. The keys for obtaining error information are described in the "Constants" (page 46) section.

Tasks

Constructors

`NSAppleScript` (page 44)
Creates an empty NSAppleScript object.

Getting Information About a Script

`isCompiled` (page 45)
Returns `true` if the receiver is already compiled, `false` otherwise.

`source` (page 46)
Returns the source code of the receiver if it is available, `null` otherwise.

`richTextSource` (page 46)
Returns the syntax-highlighted source code of the receiver if the receiver has been compiled and its source code is available.

Compiling and Executing a Script

`compile` (page 45)
Compiles the receiver, if it is not already compiled.

`execute` (page 45)
Executes the receiver, compiling it first if it is not already compiled.

`executeAppleEvent` (page 45)
Executes an Apple event in the context of the receiver, as a means of allowing the application to invoke a handler in the script; compiles the receiver first if it is not already compiled.

Constructors

NSAppleScript

Creates an empty NSAppleScript object.

```
public NSApplescript()
```

Discussion
Use the other constructor to create an object with associated AppleScript source code.

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

Creates an NSAppleScript object from the AppleScript source code contained in *source*.

```
public NSAppleScript(String source)
```

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

Instance Methods

compile

Compiles the receiver, if it is not already compiled.

```
public boolean compile(NSMutableDictionary errorInfo)
```

Discussion
Returns `true` for success or if the script was already compiled, `false` and it fills in the error information dictionary otherwise.

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

execute

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

```
public NSAppleEventDescriptor execute(NSMutableDictionary errorInfo)
```

Discussion
Returns the result of executing the script, or `null` and it fills in the error information dictionary for failure.

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

executeAppleEvent

Executes an Apple event in the context of the receiver, as a means of allowing the application to invoke a handler in the script; compiles the receiver first if it is not already compiled.

```
public NSAppleEventDescriptor executeAppleEvent(NSAppleEventDescriptor event,
    NSMutableDictionary errorInfo)
```

Discussion
Returns the result of executing the event, or `null` and it fills in the error information dictionary for failure. You cannot use this method to send Apple events to other applications.

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

isCompiled

Returns `true` if the receiver is already compiled, `false` otherwise.

```
public boolean isCompiled()
```

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

richTextSource

Returns the syntax-highlighted source code of the receiver if the receiver has been compiled and its source code is available.

```
public NSAttributedString richTextSource()
```

Discussion
Returns `null` otherwise.

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

source

Returns the source code of the receiver if it is available, `null` otherwise.

```
public String source()
```

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

Constants

If the result of `compile` (page 45), `execute` (page 45), or `executeAppleEvent` (page 45), signals failure (`false`, `null`, or `null`, respectively), the method puts error information into the mutable dictionary passed to it. The error info dictionary may contain entries that use any combination of the following keys, including no entries at all.

Constant	Description
`AppleScriptErrorMessage`	A String that supplies a detailed description of the error condition.
`AppleScriptErrorNumber`	The error number.
`AppleScriptErrorAppName`	A String that specifies the name of the application that generated the error.
`AppleScriptErrorBriefMessage`	A String that provides a brief description of the error.
`AppleScriptErrorRange`	An NSRange that specifies a range.

NSArchiver

Inherits from	NSCoder : NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Archives and Serializations Programming Guide for Cocoa

Class at a Glance

An NSArchiver encodes objects into a format that can be written to a file. The archiving process traverses a set of interconnected objects, making sure to encode each one only once.

Principal Attributes

- An NSMutableData object containing the encoded data

NSArchiver Constructors
: Returns an archiver.

Commonly Used Methods

`archiveRootObjectToFile` (page 50)
: Archives a graph of objects to a file.

`archivedDataWithRootObject` (page 49)
: Archives a graph of objects into an NSMutableData object.

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 set of objects, the class information and instance variables for each object are written to the archive. NSArchiver's companion class, NSUnarchiver (page 623), decodes the data in an archive and creates a set 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.

Tasks

Constructors

`NSArchiver` (page 49)
Creates an empty archiver.

Archiving Data

`archivedDataWithRootObject` (page 49)
Returns a data object containing the encoded form of the object graph whose root object is *rootObject*.

`archiveRootObjectToFile` (page 50)
Creates a temporary instance of NSArchiver and archives *rootObject* by encoding it into a data object and writing the resulting data object to the file *path*.

`classNameGloballyEncodedForTrueClassName` (page 50)
Returns the class name globally used to archive instances of the class *trueName*.

`globallyEncodeClassNameIntoClassName` (page 50)
Encodes a substitute name globally used for the class named *trueName*.

`encodeRootObject` (page 53)
Archives *rootObject* along with all the objects to which it is connected.

`encodeConditionalObject` (page 52)
Archives *object* conditionally.

`encodeByte` (page 51)
Encodes *aByte*.

`encodeChar` (page 51)
Encodes *aChar*.

`encodeDataObject` (page 52)
Encodes *aData*.

`encodeDouble` (page 52)
Encodes *aDouble*.

`encodeFloat` (page 52)
Encodes *aFloat*.

`encodeInt` (page 53)
Encodes *anInt*.

`encodeLong` (page 53)
Encodes *aLong*.

`encodeObject` (page 53)
Encodes *anObject*.

`encodeShort` (page 53)
Encodes *aShort*.

`versionForClassName` (page 54)
Returns the version number for the current implementation of the class named *`className`* or `NSArray.NotFound` if no class named *`className`* exists.

Getting the Archived Data

`data` (page 51)
Returns the archived data.

Substituting Classes or Objects

`classNameEncodedForTrueClassName` (page 51)
Returns the class name used to archive instances of the class *`trueName`*.

`encodeClassNameIntoClassName` (page 51)
Encodes a substitute name for the class named *`trueName`*.

`replaceObject` (page 54)
Causes the NSArchiver to treat subsequent requests to encode *`object`* as though they were requests to encode *`newObject`*.

Constructors

NSArchiver

Creates an empty archiver.

```
public NSArchiver()
```

Discussion
Use the other constructor, or the static methods `archiveRootObjectToFile` (page 50) or `archivedDataWithRootObject` (page 49), instead.

Creates an archiver, encoding stream and version information into *`data`*.

```
public NSArchiver(NSMutableData data)
```

Discussion
Throws an `InvalidArgumentException` if *`data`* is `null`.

Static Methods

archivedDataWithRootObject

Returns a data object containing the encoded form of the object graph whose root object is *`rootObject`*.

```
public static NSData archivedDataWithRootObject(Object rootObject)
```

Discussion
This method invokes `encodeRootObject` (page 53) to create a temporary archiver that encodes the object graph.

See Also
`encodeRootObject` (page 53)

archiveRootObjectToFile

Creates a temporary instance of NSArchiver and archives *rootObject* by encoding it into a data object and writing the resulting data object to the file *path*.

```
public static boolean archiveRootObjectToFile(Object rootObject, String path)
```

Discussion
This convenience method invokes `archivedDataWithRootObject` (page 49) to get the encoded data. Returns `true` upon success.

The archived data should be retrieved from the archive by an NSUnarchiver (page 623) object.

See Also
`archivedDataWithRootObject` (page 49)

classNameGloballyEncodedForTrueClassName

Returns the class name globally used to archive instances of the class *trueName*.

```
public static String classNameGloballyEncodedForTrueClassName(String trueName)
```

See Also
`globallyEncodeClassNameIntoClassName` (page 50)
`classNameEncodedForTrueClassName` (page 51)

globallyEncodeClassNameIntoClassName

Encodes a substitute name globally used for the class named *trueName*.

```
public static void globallyEncodeClassNameIntoClassName(String trueName, String
    inArchiveName)
```

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. Instead, invoke it before `encodeRootObject` (page 53).

See Also
`classNameGloballyEncodedForTrueClassName` (page 50)
`encodeClassNameIntoClassName` (page 51)

Instance Methods

classNameEncodedForTrueClassName

Returns the class name used to archive instances of the class *trueName*.

```
public String classNameEncodedForTrueClassName(String trueName)
```

See Also
encodeClassNameIntoClassName (page 51)
classNameGloballyEncodedForTrueClassName (page 50)

data

Returns the archived data.

```
public NSMutableData data()
```

Discussion
The returned data object is the same one specified as the argument to the constructor. 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 53) has returned. In other words, although it is possible for a class to invoke this method from within an encoding method, that method must not alter the data.

encodeByte

Encodes *aByte*.

```
public void encodeByte(byte aByte)
```

Discussion
This method must be matched by a subsequent decodeByte (page 627) message.

encodeChar

Encodes *aChar*.

```
public void encodeChar(char aChar)
```

Discussion
This method must be matched by a subsequent decodeChar (page 627) message.

encodeClassNameIntoClassName

Encodes a substitute name for the class named *trueName*.

```
public void encodeClassNameIntoClassName(String trueName, String inArchiveName)
```

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 . Instead, invoke it before `encodeRootObject` (page 53).

See Also
`classNameEncodedForTrueClassName` (page 51)
`globallyEncodeClassNameIntoClassName` (page 50)

encodeConditionalObject

Archives *object* conditionally.

```
public void encodeConditionalObject(Object object)
```

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" for more information.)

If *object* is `null`, the NSArchiver encodes it unconditionally as `null`. This method throws an `InvalidArgumentException` if no root object has been encoded.

encodeDataObject

Encodes *aData*.

```
public void encodeDataObject(NSData aData)
```

Discussion
This method must be matched by a subsequent `decodeDataObject` (page 627) message.

encodeDouble

Encodes *aDouble*.

```
public void encodeDouble(double aDouble)
```

Discussion
This method must be matched by a subsequent `decodeDouble` (page 627) message.

encodeFloat

Encodes *aFloat*.

```
public void encodeFloat(float aFloat)
```

Discussion
This method must be matched by a subsequent `decodeFloat` (page 628) message.

encodeInt

Encodes *anInt*.

```
public void encodeInt(int anInt)
```

Discussion
This method must be matched by a subsequent `decodeInt` (page 628) message.

encodeLong

Encodes *aLong*.

```
public void encodeLong(long aLong)
```

Discussion
This method must be matched by a subsequent `decodeLong` (page 628) message.

encodeObject

Encodes *anObject*.

```
public void encodeObject(Object anObject)
```

Discussion
This method must be matched by a subsequent `decodeObject` (page 628) message.

encodeRootObject

Archives *rootObject* along with all the objects to which it is connected.

```
public void encodeRootObject(Object rootObject)
```

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" for more information.)

This message must not be sent more than once to a given NSArchiver; an `InvalidArgumentException` is thrown if a root object has already been encoded. Therefore, don't attempt to reuse an NSArchiver; instead, create a new one. To encode multiple object graphs, use distinct NSArchivers.

encodeShort

Encodes *aShort*.

```
public void encodeShort(short aShort)
```

Discussion
This method must be matched by a subsequent `decodeShort` (page 628) message.

replaceObject

Causes the NSArchiver to treat subsequent requests to encode *object* as though they were requests to encode *newObject*.

```
public void replaceObject(Object object, Object newObject)
```

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

versionForClassName

Returns the version number for the current implementation of the class named *className* or `NSArray.NotFound` if no class named *className* exists.

```
public int versionForClassName(String className)
```

Discussion
The class version number of each encoded object is written to the archive so that newer versions of the class can detect and properly decode older archived versions.

NSArray

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.foundation
Companion guides	Collections Programming Topics for Cocoa Key-Value Coding Programming Guide Property List Programming Guide Predicate Programming Guide

Class at a Glance

An NSArray stores an immutable array of objects. The mutable subclass of NSArray is NSMutableArray (page 293).

Principal Attributes

- A count of the number of objects in the array
- The list of objects contained in the array

NSArray Constructors
: Returns an array.

Commonly Used Methods

`count` (page 61)
: Returns the number of objects currently in the array.

`objectAtIndex` (page 64)
: Returns the object located at the specified index.

Overview

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

NSArray's two primitive methods—`count` (page 61) and `objectAtIndex` (page 64)—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 64) and `reverseObjectEnumerator` (page 65) 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 in both the Java API and Foundation, such as java.util.Hashtable or NSDictionary. See the `objectEnumerator` method description for a code excerpt that shows how to use these methods to access the elements of an array.

NSArray provides methods for querying the elements of the array. The `indexOfObject` (page 63)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 `equals` (page 424) message. Another method, `indexOfIdenticalObject` (page 62), is provided for the less common case of determining whether a specific object is present in the array. The `indexOfIdenticalObject` method tests each element in the array to see whether its `id` matches that of the argument.

NSArray's `filteredArrayUsingPredicate` (page 62) method allows you to create a new array from an existing array filtered using a predicate (see Predicates Programming Guide).

To act on the array as a whole, a variety of other methods are defined. You can extract a subset of the array (`subarrayWithRange` (page 66)) or concatenate the elements of an array of Strings into a single string (`componentsJoinedByString` (page 61)). In addition, you can compare two arrays using the `isEqualToArray` (page 63) and `firstObjectCommonWithArray` (page 62) methods. Finally, you can create new arrays that contain the objects in an existing array and one or more additional objects with `arrayByAddingObject` (page 60) and `arrayByAddingObjectsFromArray` (page 61).

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 61) and `objectAtIndex` (page 64). 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 constructor for your subclass that is suited to the backing store that the subclass is managing. The NSArray class adopts the NSCopying, NSMutableCopying, and NSCoding interfaces; if you want instances of your own custom subclass created from copying or coding, override the methods in these interfaces.

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. 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 the corresponding Core Foundation type, CFArray. 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.

Tasks

Constructors

`NSArray` (page 59)

Querying the Array

`containsObject` (page 61)
: Returns `true` if *`anObject`* is present in the array.

`count` (page 61)
: Returns the number of objects currently in the array.

`getObjects` (page 62)
: Copies all objects contained in the receiver to *`aBuffer`*.

`indexOfObject` (page 63)
: Searches all objects in the receiver for *anObject* and returns the lowest index whose corresponding array value is equal to *anObject*.

`indexOfIdenticalObject` (page 62)
: This method has been deprecated.

`lastObject` (page 63)
: Returns the object in the array with the highest index value.

`objectAtIndex` (page 64)
: Returns the object located at *index*.

`objectsAtIndexes` (page 64)
: Returns an array containing the objects in the receiver at the specified indexes.

`objectEnumerator` (page 64)

`reverseObjectEnumerator` (page 65)
: Returns 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.

Comparing Arrays

`firstObjectCommonWithArray` (page 62)
: Returns the first object contained in the receiver that's equal to an object in *otherArray*.

`isEqualToArray` (page 63)
: Compares the receiving array to *otherArray*.

Deriving New Arrays

`arrayByAddingObject` (page 60)
: Returns a new array that is a copy of the receiver with *anObject* added to the end.

`arrayByAddingObjectsFromArray` (page 61)
: Returns a new array that is a copy of the receiver with the objects contained in *otherArray* added to the end.

`filteredArrayUsingPredicate` (page 62)
: Evaluates the *predicate* against the receiver's content and returns a new array containing the objects that match.

`subarrayWithRange` (page 66)
: Returns a new array containing the receiver's elements that fall within the limits specified by *range*.

Sorting Arrays

`sortedArrayUsingDescriptors` (page 65)
: Returns a copy of the receiver sorted as specified by *sortDescriptors*.

`sortedArrayUsingSelector` (page 65)
: Returns an array that lists the receiver's elements in ascending order, as determined by the comparison method specified by the selector *selector*.

Working with String Elements

`componentsJoinedByString` (page 61)
Constructs and returns a String that is the result of interposing *separator* between the elements of the receiver's array.

`componentsSeparatedByString` (page 59)
Returns an NSArray containing substrings from *aString* that have been divided by *separator*.

Constructors

NSArray

```
public NSArray()
```

Discussion
Creates an empty array. This method is used by mutable subclasses of `NSArray`.

```
public NSArray(Object anObject)
```

Discussion
Creates an array containing the single element *anObject*. After an immutable array has been initialized in this way, it can't be modified.

```
public NSArray(Object[] objects)
```

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

```
public NSArray(NSArray anArray)
```

Discussion
Creates an array containing the objects in *anArray*. After an immutable array has been initialized in this way, it can't be modified.

Static Methods

componentsSeparatedByString

Returns an NSArray containing substrings from *aString* that have been divided by *separator*.

```
public static NSArray componentsSeparatedByString(String aString, String separator)
```

Discussion
The substrings in the array appear in the order they did in *aString*. If *aString* begins or ends with the separator, the first or last substring, respectively, is empty. For example, this code excerpt:

```
NSStringReference list = "wrenches, hammers, saws";
```

```
NSArray listItems = [list.componentsSeparatedByString (", ")];
```

produces an array with these contents:

Index	Substring
0	wrenches
1	hammers
2	saws

If *list* begins with a comma and space—for example, ", wrenches, hammers, saws"—the array has these contents:

Index	Substring
0	(empty string)
1	wrenches
2	hammers
3	saws

If *list* has no separators—for example, "wrenches"—the array contains the string itself, in this case "wrenches".

See Also
`componentsJoinedByString` (page 61)
`componentsSeparatedByString` (page 597) (NSStringReference)

Instance Methods

arrayByAddingObject

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

```
public NSArray arrayByAddingObject(Object anObject)
```

Discussion
If *anObject* is `null`, an `InvalidArgumentException` is thrown.

See Also
`addObject` (page 296) (NSMutableArray)

arrayByAddingObjectsFromArray

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

```
public NSArray arrayByAddingObjectsFromArray(NSArray otherArray)
```

See Also
addObjectsFromArray (page 296) (NSMutableArray)

componentsJoinedByString

Constructs and returns a String that is the result of interposing *separator* between the elements of the receiver's array.

```
public String componentsJoinedByString(String separator)
```

Discussion
For example, this code excerpt writes the path System/Library to the console:

```
NSArray pathArray = new NSArray(new Object[] {"System",
    "Library"});
System.out.println("The path is "+
    pathArray.componentsJoinedByString("/") + ".");
```

Each element in the receiver's array must handle either description, or if it is not implemented, toString (page 426). If the receiver has no elements, a String representing an empty string is returned.

See Also
componentsSeparatedByString (page 597) (NSStringReference)

containsObject

Returns true if *anObject* is present in the array.

```
public boolean containsObject(Object anObject)
```

Discussion
This method determines whether an object is present in the array by sending an equals (page 424) message to each of the array's objects (and passing *anObject* as the parameter to each equals message).

See Also
indexOfObject (page 63)
indexOfIdenticalObject (page 62)

count

Returns the number of objects currently in the array.

```
public int count()
```

See Also
objectAtIndex (page 64)

filteredArrayUsingPredicate

Evaluates the *predicate* against the receiver's content and returns a new array containing the objects that match.

```
public NSArray filteredArrayUsingPredicate(NSPredicate predicate)
```

Discussion
For more details, see Predicates Programming Guide.

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

firstObjectCommonWithArray

Returns the first object contained in the receiver that's equal to an object in *otherArray*.

```
public Object firstObjectCommonWithArray(NSArray otherArray)
```

Discussion
If no such object is found, this method returns `null`. This method uses `equals` (page 424) to check for object equality.

See Also
containsObject (page 61)

getObjects

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

```
public void getObjects(Object[] aBuffer)
```

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

```
public void getObjects(Object[] aBuffer, NSRange aRange)
```

indexOfIdenticalObject

This method has been deprecated.

```
public int indexOfIdenticalObject(Object anObject)
```

This method has been deprecated.

```
public int indexOfIdenticalObject(Object anObject, NSRange aRange)
```

See Also
containsObject (page 61)

indexOfObject (page 63)

indexOfObject

Searches all objects in the receiver for *anObject* and returns the lowest index whose corresponding array value is equal to *anObject*.

```
public int indexOfObject(Object anObject)
```

Discussion
Objects are considered equal if equals (page 424) returns true. If none of the specified objects is equal to *anObject*, returns NSArray.NotFound.

Searches the specified range within the receiver for *anObject* and returns the lowest index whose corresponding array value is equal to *anObject*.

```
public int indexOfObject(Object anObject, NSRange aRange)
```

Discussion
Objects are considered equal if equals (page 424) returns true. If none of the specified objects is equal to *anObject*, returns NSArray.NotFound.

See Also
containsObject (page 61)
indexOfIdenticalObject (page 62)

isEqualToArray

Compares the receiving array to *otherArray*.

```
public boolean isEqualToArray(NSArray otherArray)
```

Discussion
If the contents of *otherArray* are equal to the contents of the receiver, this method returns true. If not, it returns false.

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 equals (page 424) test.

lastObject

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

```
public Object lastObject()
```

Discussion
If the array is empty, lastObject returns null.

See Also
removeLastObject (page 298) (NSMutableArray)

objectAtIndex

Returns the object located at *index*.

```
public Object objectAtIndex(int 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`), a `RangeException` is thrown.

See Also
`count` (page 61)

objectEnumerator

```
public java.util.Enumeration objectEnumerator()
```

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:

```
java.util.Enumeration enumerator = myArray.objectEnumerator();

while (enumerator.hasMoreElements()) {{
    Object anObject = enumerator.nextElement();
    /* code to act on each element */
}
```

When this method is used with mutable subclasses of NSArray, your code shouldn't modify the array during enumeration.

See Also
`reverseObjectEnumerator` (page 65)
`nextElement` (page 167) (NSEnumerator)

objectsAtIndexes

Returns an array containing the objects in the receiver at the specified indexes.

```
public NSArray objectsAtIndexes(NSIndexSet indexes)
```

Discussion
Throws an exception if any location in *indexes* exceeds the bounds of the receiver.

Availability
Available in Mac OS X version 10.4 and later.

See Also
`count` (page 61)
`objectAtIndex` (page 64)

reverseObjectEnumerator

Returns 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.

```
public java.util.Enumeration reverseObjectEnumerator()
```

Discussion
Your code shouldn't modify the array during enumeration.

See Also
`objectEnumerator` (page 64)
`nextElement` (page 167) (NSEnumerator)

sortedArrayUsingDescriptors

Returns a copy of the receiver sorted as specified by *`sortDescriptors`*.

```
public NSArray sortedArrayUsingDescriptors(NSArray sortDescriptors)
```

Discussion
The first descriptor specifies the primary key value 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 (page 581) for additional information.

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

See Also
`sortedArrayUsingSelector` (page 65)

sortedArrayUsingSelector

Returns an array that lists the receiver's elements in ascending order, as determined by the comparison method specified by the selector *`selector`*.

```
public NSArray sortedArrayUsingSelector(NSSelector selector)
```

Discussion
The new array contains references to the receiver's elements, not copies of them. The retain count is incremented for each element in the receiving array.

The *`selector`* message is sent to each object in the array and has as its single argument another object in the array. The *`selector`* method is used to compare two elements at a time and should return `OrderedAscending` if the receiver is smaller than the argument, `OrderedDescending` if the receiver is larger than the argument, and `OrderedSame` if they are equal.

See Also
`sortedArrayUsingDescriptors` (page 65)

subarrayWithRange

Returns a new array containing the receiver's elements that fall within the limits specified by *range*.

```
public NSArray subarrayWithRange(NSRange range)
```

Discussion
If *range* isn't within the receiver's range of elements, a `RangeException` is thrown.

For example, the following code example creates an array containing the elements found in the first half of *wholeArray* (assuming *wholeArray* exists).

```
NSRange theRange = new NSRange(0, wholeArray.count()/2);
NSArray halfArray = wholeArray.subarrayWithRange(theRange);
```

Constants

NSArray provides the following constant as a convenience; you can use it to compare to values returned by some NSArray methods:

Constant	Description
`NotFound`	Returned when an object is not found in an NSArray.

NSAttributedString

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.foundation
Companion guide	Attributed Strings Programming Guide

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 classes NSAttributedString and NSMutableAttributedString declare the programmatic interface for read-only attributed strings and modifiable attributed strings, respectively. Methods supporting the drawing of NSAttributedStrings are found in the Application Kit class NSGraphics. The Application Kit 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.

The mutable subclass of NSAttributedString is NSMutableAttributedString (page 303).

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.

Tasks

Constructors

`NSAttributedString` (page 69)
Creates an empty NSAttributedString.

Retrieving Character Information

`stringReference` (page 74)
Returns the character contents of the receiver as an NSStringReference object.

`length` (page 72)
Returns the length of the receiver's string object.

Retrieving Attribute Information

`attributesAtIndex` (page 71)
Returns the attributes for the character at *`index`*.

`attributeAtIndex` (page 70)
Returns the value for the attribute named *`attributeName`* of the character at *`index`*, or `null` if there is no such attribute.

Comparing Attributed Strings

`isEqualToAttributedString` (page 72)
Returns `true` if the receiver is equal to *`otherString`*.

Extracting a Substring

`attributedSubstringWithRange` (page 71)
Returns an NSAttributedString object consisting of the characters and attributes within *`aRange`* in the receiver.

Retrieving General Information

`containsAttachments` (page 71)
Returns `true` if the receiver contains any attachment attributes, `false` otherwise.

`doubleClickAtIndex` (page 72)
Returns the range of characters that form a word (or other linguistic unit) surrounding *`index`*, taking language characteristics into account.

`fontAttributesInRange` (page 72)
Returns the font attributes in effect for the character at *`aRange`*`.location`.

`lineBreakBeforeIndex` (page 73)
Returns the index of the closest character before *`index`* and within *`aRange`* that can be placed on a new line when laying out text.

`lineBreakByHyphenatingBeforeIndex` (page 73)
Returns the index of the closest character before *`index`* and within *`aRange`* that can be placed on a new line by hyphenating.

`nextWordFromIndex` (page 73)
Returns the index of the first character of the word after or before *`index`*.

`rulerAttributesInRange` (page 74)
Returns the ruler (paragraph) attributes in effect for the characters within *`aRange`*.

Generating Data

`RTFDFileWrapperFromRange` (page 74)
Returns an object that contains an RTFD document corresponding to the characters and attributes within *aRange*.

`RTFFromRange` (page 74)
Returns an NSData object that contains an RTF stream corresponding to the characters and attributes within *aRange*, omitting all attachment attributes.

`docFormatFromRange` (page 71)
Returns an NSData object that contains a Microsoft Word–format stream corresponding to the characters and attributes within the specified range.

Constructors

NSAttributedString

Creates an empty NSAttributedString.

```
public NSAttributedString()
```

Creates an NSAttributedString with the characters of *aString* and no attribute information.

```
public NSAttributedString(String aString)
```

Creates an NSAttributedString with the characters of *aString* and the attributes of *attributes*.

```
public NSAttributedString(String aString, NSDictionary attributes)
```

Creates an NSAttributedString with the characters and attributes of *attributedString*.

```
public NSAttributedString(NSAttributedString attributedString)
```

Creates an NSAttributedString with the contents of *aURL*, returning document properties, which are described in "Constants" (page 75), in *attributes*.

```
public NSAttributedString(java.net.URL aURL, NSMutableDictionary attributes)
```

Creates an NSAttributedString with the contents of *aData*, returning document properties, which are described in "Constants" (page 75), in *attributes*.

```
public NSAttributedString(NSData aData, NSMutableDictionary attributes)
```

Creates an NSAttributedString from *wrapper*, an NSFileWrapper object containing an RTFD document.

```
public NSAttributedString(NSFileWrapper wrapper, NSMutableDictionary attributes)
```

Discussion
Also returns in *attributes* a dictionary containing document-level attributes described in "Constants" (page 75). Returns `null` if *wrapper* can't be interpreted as an RTFD document.

Creates an NSAttributedString from the HTML contained in *data* and base URL *aURL*.

```
public NSAttributedString(NSData data, java.net.URL aURL, NSMutableDictionary
    attributes)
```

Discussion
Also returns in *attributes* a dictionary containing document-level attributes described in "Constants" (page 75). Returns `null` if the file at *aURL* can't be decoded.

Creates an NSAttributedString from HTML contained in *data*.

```
public NSAttributedString(NSData data, NSDictionary options, NSMutableDictionary
    attributes)
```

Discussion
options can contain one of the values described in NSMutableAttributedString's `readFromURL` (page 310) method. Also returns in *attributes* a dictionary containing document-level attributes described in "Constants" (page 75). Returns `null` if *data* can't be decoded.

Instance Methods

attributeAtIndex

Returns the value for the attribute named *attributeName* of the character at *index*, or `null` if there is no such attribute.

```
public Object attributeAtIndex(String attributeName, int index, NSMutableRange
    aRange)
```

Discussion
If the named attribute exists at *index* and *aRange* is non-`null`, it's filled with a range over which the named attribute's value applies. If the named attribute doesn't exist at *index* and *aRange* is non-`null`, *aRange* is filled instead with the range over which the attribute doesn't exist. This range isn't necessarily the maximum range covered by *attributeName*, and its extent is implementation-dependent.

Throws an exception if *index* lies beyond the end of the receiver's characters.

Returns the value for the attribute named *attributeName* of the character at *index*, or `null` if there is no such attribute.

```
public Object attributeAtIndex(String attributeName, int index, NSMutableRange
    aRange, NSRange rangeLimit)
```

Discussion
If the named attribute exists at *index* and *aRange* is non-`null`, it's filled with 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 doesn't exist at *index* and *aRange* is non-`null`, *aRange* is filled instead with the full range over which the attribute doesn't exist, clipped to *rangeLimit*.

Throws an exception if *index* or any part of *rangeLimit* lies beyond the end of the receiver's characters.

See Also
`attributesAtIndex` (page 71)

attributedSubstringWithRange

Returns an NSAttributedString object consisting of the characters and attributes within *aRange* in the receiver.

```
public NSAttributedString attributedSubstringWithRange(NSRange aRange)
```

Discussion
Throws a `RangeException` 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.

attributesAtIndex

Returns the attributes for the character at *index*.

```
public NSDictionary attributesAtIndex(int index, NSMutableRange aRange)
```

Discussion
If *aRange* is non-`null` it's filled with 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.

Throws an exception if *index* lies beyond the end of the receiver's characters.

Returns the attributes for the character at *index*.

```
public NSDictionary attributesAtIndex(int index, NSMutableRange aRange, NSRange
    rangeLimit)
```

Discussion
If *aRange* is non-`null`, it's filled with the maximum range over which the attributes and values are the same as those at *index*, clipped to *rangeLimit*.

Throws an exception if *index* or any part of *rangeLimit* lies beyond the end of the receiver's characters.

See Also
`attributeAtIndex` (page 70)

containsAttachments

Returns `true` if the receiver contains any attachment attributes, `false` otherwise.

```
public boolean containsAttachments()
```

Discussion
This method checks only for attachment attributes, not for `NSAttachmentCharacter`.

docFormatFromRange

Returns an NSData object that contains a Microsoft Word–format stream corresponding to the characters and attributes within the specified range.

```
public NSData docFormatFromRange(NSRange range, NSDictionary dict)
```

Discussion
The range is passed in the *range* parameter. Also writes the document-level attributes in *dict*, as explained in "Constants" (page 75). If there are no document-level attributes, *dict* can be `null`. Throws an `NSRangeException` if any part of *range* lies beyond the end of the receiver's characters.

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

doubleClickAtIndex

Returns the range of characters that form a word (or other linguistic unit) surrounding *index*, taking language characteristics into account.

```
public NSRange doubleClickAtIndex(int index)
```

Discussion
Throws a `RangeException` if *index* lies beyond the end of the receiver's characters.

See Also
`nextWordFromIndex` (page 73)

fontAttributesInRange

Returns the font attributes in effect for the character at *aRange*.`location`.

```
public NSDictionary fontAttributesInRange(NSRange aRange)
```

Discussion
Use this method to obtain font attributes that are to be copied or pasted with "copy font" operations. Throws a `RangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

See Also
`rulerAttributesInRange` (page 74)

isEqualToAttributedString

Returns `true` if the receiver is equal to *otherString*.

```
public boolean isEqualToAttributedString(NSAttributedString otherString)
```

Discussion
Attributed strings must match in both characters and attributes to be equal.

length

Returns the length of the receiver's string object.

```
public int length()
```

See Also
`length` (page 600) (NSStringReference)

lineBreakBeforeIndex

Returns the index of the closest character before *index* and within *aRange* that can be placed on a new line when laying out text.

```
public int lineBreakBeforeIndex(int index, NSRange aRange)
```

Discussion
In other words, finds the appropriate line break when the character at *index* won't fit on the same line as the character at the beginning of *aRange*. Returns `NSArray.NotFound` if no line break is possible before *index*.Throws a `RangeException` if *index* or any part of *aRange* lies beyond the end of the receiver's characters.

See Also
`lineBreakByHyphenatingBeforeIndex` (page 73)

lineBreakByHyphenatingBeforeIndex

Returns the index of the closest character before *index* and within *aRange* that can be placed on a new line by hyphenating.

```
public int lineBreakByHyphenatingBeforeIndex(int location, NSRange aRange)
```

Discussion
In other words, during text layout, finds the appropriate line break by hyphenation (the character index at which the hyphen glyph should be inserted) when the character at *index* won't fit on the same line as the character at the beginning of *aRange*. Returns `NSArray.NotFound` if no line break by hyphenation is possible before *index*.Throws a `RangeException` if *index* or any part of *aRange* lies beyond the end of the receiver's characters.

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

See Also
`lineBreakBeforeIndex` (page 73)

nextWordFromIndex

Returns the index of the first character of the word after or before *index*.

```
public int nextWordFromIndex(int index, boolean flag)
```

Discussion
If *flag* is `true`, this is the first character after *index* that begins a word; if *flag* is `false`, it's the first character before *index* that begins a word, whether *index* is located within a word or not. If *index* lies at either end of the string and the search direction would progress past that end, it's returned unchanged. This method is intended for moving the insertion point during editing, not for linguistic analysis or parsing of text.Throws a `RangeException` if *index* lies beyond the end of the receiver's characters.

See Also
`lineBreakBeforeIndex` (page 73)

RTFDFileWrapperFromRange

Returns an object that contains an RTFD document corresponding to the characters and attributes within *aRange*.

```
public Object RTFDFileWrapperFromRange(NSRange aRange, NSDictionary docAttributes)
```

Discussion
The file wrapper also includes the document-level attributes in *docAttributes*. If there are no document-level attributes, *docAttributes* can be `null`. Throws a `RangeException` if any part of *aRange* lies beyond the end of the receiver's characters.You can save the file wrapper using NSFileWrapper's `writeToFile`.

See Also
`RTFFromRange` (page 74)

RTFFromRange

Returns an NSData object that contains an RTF stream corresponding to the characters and attributes within *aRange*, omitting all attachment attributes.

```
public NSData RTFFromRange(NSRange aRange, NSDictionary docAttributes)
```

Discussion
Also writes the document-level attributes in *docAttributes*. If there are no document-level attributes, *docAttributes* can be `null`. Throws a `RangeException` if any part of *aRange* lies beyond the end of the receiver's characters.When writing data to the pasteboard, you can use the NSData object as the first argument to NSPasteboard's `setDataForType`, with a second argument of `NSPasteboard.RTFPboardType`.Although this method strips attachments, it leaves the attachment characters in the text itself. NSText's `RTFFromRange`, on the other hand, does strip attachment characters when extracting RTF.

See Also
`RTFDFileWrapperFromRange` (page 74)

rulerAttributesInRange

Returns the ruler (paragraph) attributes in effect for the characters within *aRange*.

```
public NSDictionary rulerAttributesInRange(NSRange aRange)
```

Discussion
The only ruler attribute currently defined is that named by `NSParagraphStyleAttributeName`. Use this method to obtain attributes that are to be copied or pasted with "copy ruler" operations. Throws a `RangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

See Also
`fontAttributesInRange` (page 72)

stringReference

Returns the character contents of the receiver as an NSStringReference object.

```
public NSStringReference stringReference()
```

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.

Constants

NSAttributedString provides the following attribute name constants:

Attribute Identifier	Value Class	Default Value
`AttachmentAttributeName`	NSTextAttachment	None (no attachment)
`BackgroundColorAttributeName`	NSColor	None (no background drawn)
`BaselineOffsetAttributeName`	float	0.0
`CursorAttributeName`	NSCursor	IBeamCursor
`ExpansionAttributeName`	float	0 (no expansion)
`FontAttributeName`	NSFont	Helvetica 12-point
`ForegroundColorAttributeName`	NSColor	Black
`KernAttributeName`	float	0.0
`LigatureAttributeName`	int	1 (standard ligatures)
`ObliquenessAttributeName`	float	0 (no skew)
`ParagraphStyleAttributeName`	NSParagraphStyle	Object returned by NSParagraphStyle's `defaultParagraphStyle` method
`ShadowAttributeName`	NSShadow	`null` (no shadow)
`StrikethroughColorAttributeName`	NSColor	`null` (same as foreground color)
`StrikethroughStyleAttributeName`	int	0 (no strikethrough)
`StrokeColorAttributeName`	NSColor	`null` (same as foreground color)
`StrokeWidthAttributeName`	float	0 (no stroke)
`SuperscriptAttributeName`	int	0
`ToolTipAttributeName`	String	`null` (no tooltip)

Attribute Identifier	Value Class	Default Value
`UnderlineColorAttributeName`	NSColor	`null` (same as foreground color)
`UnderlineStyleAttributeName`	int	None (no underline)

NSAttributedString provides the following constants to use when working with underlines:

Constant	Description
`UnderlineByWordMask`	Underline skips whitespace characters.
`UnderlineStyleNone`	No underline.
`UnderlineStyleSingle`	Single underline drawn below the characters.
`UnderlineStyleThick`	Thick underline drawn below the characters.
`UnderlineStyleDouble`	Double underline drawn below the characters.
`UnderlinePatternSolid`	Draw a solid underline.
`UnderlinePatternDot`	Draw an underline using a pattern of dots.
`UnderlinePatternDash`	Draw an underline using a pattern of dashes.
`UnderlinePatternDashDot`	Draw an underline using a pattern of alternating dashes and dots.
`UnderlinePatternDashDotDot`	Draw an underline using a pattern of a dash followed by two dots.

The following constants previously used for underline style were deprecated in Mac OS X v10.3:

```
NoUnderlineStyle
SingleUnderlineStyle
UnderlineStrikethroughMask
```

The constructors can return a dictionary with the following document-wide attributes:

Constant	Description
`PaperSize`	NSSize.
`LeftMargin`	float, in points.
`RightMargin`	float, in points.
`TopMargin`	float, in points.
`BottomMargin`	float, in points.
`HyphenationFactor`	float.
`DocumentType`	How the document was interpreted; one of the values below.

Constant	Description
`CharacterEncoding`	For plain text files only; int specifying NSStringEncoding used to interpret the file.
`ViewSize`	NSSize.
`ViewZoom`	float; 100 == 100% zoom.
`ViewMode`	int; 0 = normal; 1 = page layout (use value of `PaperSize`).
`CocoaRTFVersion`	If RTF file, stores the version of Cocoa the file was created with. Number containing int. Absence of this value indicates RTF file not created by Cocoa or its predecessors. 100 is Mac OS X; lower values are pre–Mac OS X.
`Converted`	int. Indicates whether the file was converted by a filter service. If missing, 0, or negative, the file was originally in the format specified by document type. If 1 or more, it was converted to this type by a filter service.

The following values can be returned for the `DocumentType` key in the document attributes dictionary:

DocumentType
`NSHTMLTextDocumentType`
`NSMacSimpleTextDocumentType`
`NSPlainTextDocumentType`
`NSRTFDTextDocumentType`
`NSRTFTextDocumentType`
`NSDocFormatTextDocumentType`
`NSWordMLTextDocumentType`

NSAutoreleasePool

Inherits from	Object
Package:	com.apple.cocoa.foundation
Companion guide	Memory Management Programming Guide for Cocoa

Overview

An autorelease pool is used to manage Foundation's autorelease mechanism for Objective-C objects. NSAutoreleasePool provides Java applications access to autorelease pools. Typically it is not necessary for Java applications to use NSAutoreleasePools since Java manages garbage collection. However, some situations require an autorelease pool; for instance, if you start off a thread that calls Cocoa, there won't be a top-level pool.

You know you need an autorelease pool when you see "no pool in place - just leaking" warnings coming from your Cocoa Java application. Just wrap pools around the places where you have top-level Java threads calling into Cocoa. Use code like the following to accomplish this task:

```
int myPool = NSAutoreleasePool.push();

// Your code here

NSAutoreleasePool.pop(myPool);
```

Tasks

Creating a Pool

`push` (page 80)
Creates an NSAutoreleasePool and returns an identifier to it.

Freeing a Pool

`pop` (page 80)
Indicates that you are finished using the NSAutoreleasePool identified by *`pool`*.

Static Methods

pop

Indicates that you are finished using the NSAutoreleasePool identified by *pool*.

```
public static void pop(int pool)
```

See Also
push (page 80)

push

Creates an NSAutoreleasePool and returns an identifier to it.

```
public static int push()
```

See Also
pop (page 80)

NSBundle

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guides	Bundle Programming Guide Resource Programming Guide

Overview

An NSBundle represents a location in the file system that groups code and resources that can be used in a program. NSBundles locate program resources, dynamically load executable code, and assist in localization. You build a bundle in Xcode using one of these project types: Application, Framework, Loadable Bundle, Palette.

Tasks

Constructors

`NSBundle` (page 84)
Creates an empty NSBundle.

Getting an NSBundle

`bundleForClass` (page 84)
Returns the NSBundle that dynamically loaded *`aClass`* (a loadable bundle), the NSBundle 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.

`bundleWithIdentifier` (page 85)
Returns the previously created NSBundle instance that has the bundle identifier *`identifier`*.

`bundleWithPath` (page 85)
Returns an NSBundle that corresponds to the specified directory *`fullPath`* or `null` if *`fullPath`* does not identify an accessible bundle directory.

`mainBundle` (page 86)
Returns an NSBundle that corresponds to the directory where the application executable is located or `null` if this executable is not located in an accessible bundle directory.

`allBundles` (page 84)
Returns an array of all the application's nonframework bundles.

`allFrameworks` (page 84)
Returns an array of all of the application's bundles that represent frameworks.

Getting a Bundled Class

`principalClass` (page 91)
Returns the receiver's principal class after ensuring that the code containing the definition of that class is dynamically loaded.

Finding a Resource

`pathForResource` (page 90)
Returns the full pathname for the resource identified by *`name`* with the specified file *`extension`*.

`pathsForResources` (page 91)
Returns an array containing the pathnames for all bundle resources having the specified filename *`extension`* and residing in the resource subdirectory specified by *`subpath`*; returns an empty array if no matching resource files are found.

`resourcePath` (page 92)
Returns the full pathname of the receiving bundle's subdirectory containing resources.

Getting the Bundle Directory

`bundlePath` (page 87)
Returns the full pathname of the receiver's bundle directory.

Getting Bundle Information

`builtInPlugInsPath` (page 87)
Returns the full pathname of the receiving bundle's subdirectory containing plug-ins.

`bundleIdentifier` (page 87)
Returns the receiver's bundle identifier, which is defined by the `CFBundleIdentifier` key in the bundle's information property list.

`executablePath` (page 87)
Returns the full pathname of the receiving bundle's executable file.

`infoDictionary` (page 87)
Returns a dictionary that contains information about the receiver.

`objectForInfoDictionaryKey` (page 89)
Returns the value associated with *`key`* in the bundle's property list (`Info.plist`).

`pathForAuxiliaryExecutable` (page 89)
Returns the full pathname of the executable *`executableName`* in the receiver's bundle.

`privateFrameworksPath` (page 92)
Returns the full pathname of the receiving bundle's subdirectory containing private frameworks.

`sharedFrameworksPath` (page 92)
Returns the full pathname of the receiving bundle's subdirectory containing shared frameworks.

`sharedSupportPath` (page 93)
Returns the full pathname of the receiving bundle's subdirectory containing shared support files.

Managing Localized Resources

`localizedString` (page 85)
Returns the localized version of the string designated by *`key`* in the `Localizable.strings` file in the application's main bundle.

`localizedStringForKey` (page 89)
Returns a localized version of the string designated by *`key`* in table *`tableName`*.

Loading a Bundle's Code

`load` (page 88)
Dynamically loads the bundle's executable code into a running program, if the code has not already been loaded.

`isLoaded` (page 88)
Obtains information about the load status of a bundle.

Managing Localizations

`preferredLocalizations` (page 86)
Returns the one or more localizations from the list *`localizationsArray`* that NSBundle prefers to use to locate resources based on the user's preferences.

`localizations` (page 88)
Returns a list of all the localizations contained within the receiver's bundle.

`developmentLocalization` (page 87)
Returns the localization used to create the bundle.

`preferredLocalizations` (page 91)
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 88)
Returns a dictionary with the keys from the bundle's localized property list (`InfoPList.strings`).

Constructors

NSBundle

Creates an empty NSBundle.

```
public NSBundle()
```

Do not use this constructor. Use `bundleWithPath` (page 85) instead.

```
public NSBundle(String fullPath)
```

Static Methods

allBundles

Returns an array of all the application's nonframework bundles.

```
public static NSArray allBundles()
```

Discussion
This array includes the main bundle and all bundles that have been dynamically created but doesn't contain any bundles that represent frameworks.

allFrameworks

Returns an array of all of the application's bundles that represent frameworks.

```
public static NSArray allFrameworks()
```

Discussion
This array includes frameworks that are linked into an application when the application is built and bundles for frameworks that have been dynamically created. Only frameworks with one or more Objective-C classes in them are included.

bundleForClass

Returns the NSBundle that dynamically loaded *aClass* (a loadable bundle), the NSBundle 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.

```
public static NSBundle bundleForClass(Class aClass)
```

See Also
`mainBundle` (page 86)
`bundleWithPath` (page 85)

bundleWithIdentifier

Returns the previously created NSBundle instance that has the bundle identifier *identifier*.

```
public static NSBundle bundleWithIdentifier(String identifier)
```

Discussion
The instance must currently exist. Returns `null` if the requested bundle is not found.

bundleWithPath

Returns an NSBundle that corresponds to the specified directory *fullPath* or `null` if *fullPath* does not identify an accessible bundle directory.

```
public static NSBundle bundleWithPath(String fullPath)
```

Discussion
This method creates and initializes the returned object if there is no existing NSBundle associated with *fullPath*, in which case it returns the existing object. *fullPath* must be a full pathname for a directory; if it contains any symbolic links, they must be resolvable. If the directory doesn't exist or the user doesn't have access to it, this method returns `null`.

See Also
`mainBundle` (page 86)
`bundleForClass` (page 84)

localizedString

Returns the localized version of the string designated by *key* in the `Localizable.strings` file in the application's main bundle.

```
public static String localizedString(String key)
```

Discussion
Returns *key* if it is not found in the file.

Returns the localized version of the string designated by *key* in the `Localizable.strings` file in the application's main bundle.

```
public static String localizedString(String key, String comment)
```

Discussion
Returns *key* if it is not found in the file.

Returns the localized version of the string designated by *key* in table *tableName* in the application's main bundle.

```
public static String localizedString(String key, String tableName, String comment)
```

Discussion
The argument *tableName* specifies the `.strings` file to search. Returns *key* if it is not found in the file.

Returns the localized version of the string designated by *key* in table *tableName* in *bundle*.

```
public static String localizedString(String key, String tableName, NSBundle bundle,
     String comment)
```

Discussion
The argument *tableName* specifies the `.strings` file to search. Returns *key* if it is not found in the file.

The *comment* arguments are not used by these methods. They are used only by the `genstrings` command-line tool, which creates `.strings` files by extracting the arguments of `localizedString` invocations in your source code. Use *comment* to document the purpose of each key in the `.strings` file.

See Also
`localizedStringForKey` (page 89)

mainBundle

Returns an NSBundle that corresponds to the directory where the application executable is located or `null` if this executable is not located in an accessible bundle directory.

```
public static NSBundle mainBundle()
```

Discussion
This method allocates and initializes the returned NSBundle if it doesn't already exist.

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.

See Also
`bundleForClass` (page 84)
`bundleWithPath` (page 85)

preferredLocalizations

Returns the one or more localizations from the list *localizationsArray* that NSBundle prefers to use to locate resources based on the user's preferences.

```
public static NSArray preferredLocalizations(NSArray localizationsArray)
```

Returns the localizations that NSBundle would prefer, given the specified bundle and user preference localizations.

```
public static NSArray preferredLocalizations(NSArray localizationsArray, NSArray
    preferencesArray)
```

Discussion
Use the argument *localizationsArray* to specify the supported localizations of the bundle and use *preferencesArray* to specify the user's localization preferences. If you specify `null` for *preferencesArray*, this method uses the current 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.

Instance Methods

builtInPlugInsPath

Returns the full pathname of the receiving bundle's subdirectory containing plug-ins.

```
public String builtInPlugInsPath()
```

bundleIdentifier

Returns the receiver's bundle identifier, which is defined by the `CFBundleIdentifier` key in the bundle's information property list.

```
public String bundleIdentifier()
```

See Also
`infoDictionary` (page 87)

bundlePath

Returns the full pathname of the receiver's bundle directory.

```
public String bundlePath()
```

developmentLocalization

Returns the localization used to create the bundle.

```
public String developmentLocalization()
```

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.

executablePath

Returns the full pathname of the receiving bundle's executable file.

```
public String executablePath()
```

infoDictionary

Returns a dictionary that contains information about the receiver.

```
public NSDictionary infoDictionary()
```

Discussion
This information is extracted from the property list (`Info.plist`) associated with the bundle. The returned dictionary is empty if no `Info.plist` can be found. Common keys for accessing the values of the dictionary are `CFBundleIdentifier`, `NSMainNibFile`, and `NSPrincipalClass`.

See Also
`principalClass` (page 91)

isLoaded

Obtains information about the load status of a bundle.

```
public boolean isLoaded()
```

Discussion
Returns `true` if the bundle's code is currently loaded; otherwise, returns `false`.

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

See Also
`load` (page 88)

load

Dynamically loads the bundle's executable code into a running program, if the code has not already been loaded.

```
public boolean load()
```

Discussion
A bundle attempts to load its code—if it has any—only once. Returns `true` if the method successfully loads the bundle's code or if the code has already been loaded. Returns `false` if the method fails to load the code. You don't need to load a bundle's executable code to search the bundle's resources.

See Also
`isLoaded` (page 88)
`principalClass` (page 91)

localizations

Returns a list of all the localizations contained within the receiver's bundle.

```
public NSArray localizations()
```

localizedInfoDictionary

Returns a dictionary with the keys from the bundle's localized property list (`InfoPList.strings`).

```
public NSDictionary localizedInfoDictionary()
```

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.

localizedStringForKey

Returns a localized version of the string designated by *key* in table *tableName*.

```
public String localizedStringForKey(String key, String value, String tableName)
```

Discussion
The argument *tableName* specifies the receiver's string table to search. If *tableName* is `null` or is an empty string, the method attempts to use the table in `Localizable.strings`. The *value* argument specifies the value to return if *key* is `null` or if a localized string for *key* can't be found in the table. If *value* is `null` or an empty string, and a localized string is not found in the table, the method returns *key*. If *key* and *value* are both `null`, the method returns the empty string. 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` (page 89) to log a message when the method can't find a localized string. If you set this default to `true` (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.

See Also
`pathForResource` (page 90)
`pathsForResources` (page 91)

objectForInfoDictionaryKey

Returns the value associated with *key* in the bundle's property list (`Info.plist`).

```
public Object objectForInfoDictionaryKey(String key)
```

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.

pathForAuxiliaryExecutable

Returns the full pathname of the executable *executableName* in the receiver's bundle.

```
public String pathForAuxiliaryExecutable(String executableName)
```

pathForResource

Returns the full pathname for the resource identified by *name* with the specified file *extension*.

```
public String pathForResource(String name, String extension)
```

Discussion
If *extension* is an empty string or `null`, 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".

Returns the full pathname for the resource identified by *name*, with the specified filename *extension*, and residing in the specific resource subdirectory specified by *subpath*; returns `null` if no matching resource file exists in the bundle.

```
public String pathForResource(String name, String extension, String subpath)
```

Discussion
If *extension* is an empty string or `null`, the returned pathname is the first one encountered where the file name exactly matches *name*.

The argument *subpath* specifies the name of a specific subdirectory to search within the current bundle's general resource directory hierarchy. If *subpath* is `null`, 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".

Returns the full pathname for the resource identified by *name*, with the specified filename *extension*, residing in the resource subdirectory specified by *subpath*, and limited to global resources and those associated with *localizationName*.

```
public String pathForResource(String name, String extension, String subpath, String
    localizationName)
```

Discussion
This method is equivalent to the three parameter version except that only nonlocalized resources and those in the language-specific `.lproj` directory specified by *localizationName* are searched.

See Also
`localizedStringForKey` (page 89)
`pathsForResources` (page 91)

pathsForResources

Returns an array containing the pathnames for all bundle resources having the specified filename *extension* and residing in the resource subdirectory specified by *subpath*; returns an empty array if no matching resource files are found.

```
public NSArray pathsForResources(String extension, String subpath)
```

Discussion
This method provides a means for dynamically discovering multiple bundle resources of the same type. If *extension* is an empty string or `null`, 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 `null`, 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".

Returns an array containing pathnames for all bundle resources having the specified filename *extension*, residing in the resource subdirectory specified by *subpath*, and limited to global resources and those associated with *localizationName*.

```
public NSArray pathsForResources(String extension, String subpath, String
    localizationName)
```

Discussion
This method is equivalent to the two parameter version except that only nonlocalized resources and those in the language-specific `.lproj` directory specified by *localizationName* are searched.

See Also
`localizedStringForKey` (page 89)

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.

```
public NSArray preferredLocalizations()
```

See Also
`preferredLocalizations` (page 86)
`localizations` (page 88)

principalClass

Returns the receiver's principal class after ensuring that the code containing the definition of that class is dynamically loaded.

```
public Class principalClass()
```

Discussion
If the receiver encounters errors in loading or if it can't find the executable code file in the bundle directory, it returns `NIL`. 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 87), using "`NSExecutable`" as the key. The NSBundle 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.

See Also
`infoDictionary` (page 87)
`load` (page 88)

privateFrameworksPath

Returns the full pathname of the receiving bundle's subdirectory containing private frameworks.

```
public String privateFrameworksPath()
```

resourcePath

Returns the full pathname of the receiving bundle's subdirectory containing resources.

```
public String resourcePath()
```

See Also
`bundlePath` (page 87)

sharedFrameworksPath

Returns the full pathname of the receiving bundle's subdirectory containing shared frameworks.

```
public String sharedFrameworksPath()
```

sharedSupportPath

Returns the full pathname of the receiving bundle's subdirectory containing shared support files.

```
public String sharedSupportPath()
```

NSCharacterSet

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.foundation
Companion guide	String Programming Guide for Cocoa

Overview

An NSCharacterSet object represents a set of Unicode 3.2 compliant characters. String and NSScanner objects use NSCharacterSets to group characters together for searching operations, so that they can find any of a particular set of characters during a search. The two 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).

The NSCharacterSet class declares the programmatic interface for an object that manages a set of Unicode characters (see the NSStringReference (page 591) class specification for information on Unicode). NSCharacterSet's principal primitive method, `characterIsMember` (page 102), provides the basis for all other instance methods in its interface. A subclass of NSCharacterSet needs only to implement this method for proper behavior. For optimal performance, a subclass should also override `bitmapRepresentation` (page 102), which otherwise works by invoking `characterIsMember` (page 102) for every possible Unicode value.

The mutable subclass of NSCharacterSet is NSMutableCharacterSet (page 315).

Tasks

Constructors

`NSCharacterSet` (page 97)
Creates an empty NSCharacterSet.

Creating a Standard Character Set

`alphanumericCharacterSet` (page 98)
Returns a character set containing the characters in the categories Letters, Marks, and Numbers.

`capitalizedLetterCharacterSet` (page 98)
Returns a character set containing the characters in the category of Titlecase Letters.

`controlCharacterSet` (page 98)
Returns a character set containing the characters in the categories of Control or Format Characters.

`decimalDigitCharacterSet` (page 99)
Returns a character set containing the characters in the category of Decimal Numbers.

`decomposableCharacterSet` (page 99)
Returns 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.

`illegalCharacterSet` (page 99)
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 100)
Returns a character set containing the characters in the categories Letters and Marks.

`lowercaseLetterCharacterSet` (page 100)
Returns a character set containing the characters in the category of Lowercase Letters.

`nonBaseCharacterSet` (page 100)
Returns a character set containing the characters in the category of Marks.

`punctuationCharacterSet` (page 100)
Returns a character set containing the characters in the category of Punctuation.

`symbolCharacterSet` (page 101)
Returns a character set containing the characters in the category of Symbols.

`uppercaseLetterCharacterSet` (page 101)
Returns a character set containing the characters in the categories of Uppercase Letters and Titlecase Letters.

`whitespaceAndNewlineCharacterSet` (page 101)
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 101)
Returns a character set containing only the in-line whitespace characters space (U+0020) and tab (U+0009).

Opening a Character Set File

`characterSetWithContentsOfFile` (page 98)
Returns a character set read from the bitmap representation stored in the file at *`path`*, which must end with the extension `.bitmap`.

Testing Set Membership

`characterIsMember` (page 102)
Returns `true` if *`aCharacter`* is in the receiving character set, `false` if it isn't.

`isSupersetOfSet` (page 103)
Returns `true` if the receiving character set is a superset of *`theOtherSet`*, `false` if it isn't.

Getting a Binary Representation

`bitmapRepresentation` (page 102)
Returns an NSData object encoding the receiving character set in binary format.

Deriving New Character Sets

`characterSetByIntersectingCharacterSet` (page 102)
Returns a character set containing only characters that exist in both the receiver and *`otherSet`*.

`characterSetByInvertingCharacterSet` (page 102)
Returns a character set containing only characters that do not exist in the receiver. Inverting an immutable character set is much more efficient than inverting a mutable character set.

`characterSetBySubtractingCharacterSet` (page 102)
Returns a character set containing all the characters in the receiver except for those in *`otherSet`*.

`characterSetByUnioningCharacterSet` (page 102)
Returns a character set containing all characters that exist in either the receiver or *`otherSet`*.

Constructors

NSCharacterSet

Creates an empty NSCharacterSet.

```
public NSCharacterSet()
```

Creates a character set containing characters determined by the bitmap representation *`aData`*.

```
public NSCharacterSet(NSData aData)
```

Discussion
This capability is useful for creating a character set object with data from a file or other external data source.

Creates a character set containing characters whose Unicode values are given by *`aRange`*.

```
public NSCharacterSet(NSRange aRange)
```

Discussion
`aRange.location` is the value of the first character, and *`aRange.location`* + *`aRange.length`* – 1 is the value of the last. Returns an empty character set if *`aRange.length`* is 0.

Creates a character set containing the characters in *`aString`*.

```
public NSCharacterSet(String aString)
```

Discussion
Returns an empty character set if *aString* is empty.

Static Methods

alphanumericCharacterSet

Returns a character set containing the characters in the categories Letters, Marks, and Numbers.

```
public static NSCharacterSet alphanumericCharacterSet()
```

Discussion
Informally, this set is the set of all characters used as basic units of alphabets, syllabaries, ideographs, and digits.

See Also
`letterCharacterSet` (page 100)
`decimalDigitCharacterSet` (page 99)

capitalizedLetterCharacterSet

Returns a character set containing the characters in the category of Titlecase Letters.

```
public static NSCharacterSet capitalizedLetterCharacterSet()
```

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

See Also
`letterCharacterSet` (page 100)
`uppercaseLetterCharacterSet` (page 101)

characterSetWithContentsOfFile

Returns a character set read from the bitmap representation stored in the file at *path*, which must end with the extension `.bitmap`.

```
public static NSCharacterSet characterSetWithContentsOfFile(String path)
```

Discussion
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.

controlCharacterSet

Returns a character set containing the characters in the categories of Control or Format Characters.

```
public static NSCharacterSet controlCharacterSet()
```

Discussion
These characters are specifically the Unicode values U+0000 to U+001F and U+007F to U+009F.

See Also
`illegalCharacterSet` (page 99)

decimalDigitCharacterSet

Returns a character set containing the characters in the category of Decimal Numbers.

```
public static NSCharacterSet decimalDigitCharacterSet()
```

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.

See Also
`alphanumericCharacterSet` (page 98)

decomposableCharacterSet

Returns 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.

```
public static NSCharacterSet decomposableCharacterSet()
```

Discussion
These characters include compatibility characters as well as precomposed characters.

> **Note:** This character set doesn't currently include the Hangul characters defined in version 2.0 of the Unicode standard.

See Also
`nonBaseCharacterSet` (page 100)

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.

```
public static NSCharacterSet illegalCharacterSet()
```

See Also
`controlCharacterSet` (page 98)

letterCharacterSet

Returns a character set containing the characters in the categories Letters and Marks.

```
public static NSCharacterSet letterCharacterSet()
```

Discussion
Informally, this set is the set of all characters used as letters of alphabets and ideographs.

See Also
`alphanumericCharacterSet` (page 98)
`lowercaseLetterCharacterSet` (page 100)
`uppercaseLetterCharacterSet` (page 101)

lowercaseLetterCharacterSet

Returns a character set containing the characters in the category of Lowercase Letters.

```
public static NSCharacterSet lowercaseLetterCharacterSet()
```

Discussion
Informally, this set is the set of all characters used as lowercase letters in alphabets that make case distinctions.

See Also
`uppercaseLetterCharacterSet` (page 101)
`letterCharacterSet` (page 100)

nonBaseCharacterSet

Returns a character set containing the characters in the category of Marks.

```
public static NSCharacterSet nonBaseCharacterSet()
```

Discussion
This set is also defined as all legal Unicode characters with a nonspacing priority greater than 0. Informally, this set is the set of all characters used as modifiers of base characters.

See Also
`decomposableCharacterSet` (page 99)

punctuationCharacterSet

Returns a character set containing the characters in the category of Punctuation.

```
public static NSCharacterSet punctuationCharacterSet()
```

Discussion
Informally, this set is the set of all nonwhitespace characters used to separate linguistic units in scripts, such as periods, dashes, parentheses, and so on.

symbolCharacterSet

Returns a character set containing the characters in the category of Symbols.

```
public static NSCharacterSet symbolCharacterSet()
```

Discussion
These characters include, for example, the dollar sign ($) and the plus (+) sign.

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

uppercaseLetterCharacterSet

Returns a character set containing the characters in the categories of Uppercase Letters and Titlecase Letters.

```
public static NSCharacterSet uppercaseLetterCharacterSet()
```

Discussion
Informally, this set is the set of all characters used as uppercase letters in alphabets that make case distinctions.

See Also
`capitalizedLetterCharacterSet` (page 98)
`lowercaseLetterCharacterSet` (page 100)
`letterCharacterSet` (page 100)

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).

```
public static NSCharacterSet whitespaceAndNewlineCharacterSet()
```

See Also
`whitespaceCharacterSet` (page 101)

whitespaceCharacterSet

Returns a character set containing only the in-line whitespace characters space (U+0020) and tab (U+0009).

```
public static NSCharacterSet whitespaceCharacterSet()
```

Discussion
This set doesn't contain the newline or carriage return characters.

See Also
`whitespaceAndNewlineCharacterSet` (page 101)

Instance Methods

bitmapRepresentation

Returns an NSData object encoding the receiving character set in binary format.

```
public NSData bitmapRepresentation()
```

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*.

characterIsMember

Returns `true` if *`aCharacter`* is in the receiving character set, `false` if it isn't.

```
public boolean characterIsMember(char aCharacter)
```

characterSetByIntersectingCharacterSet

Returns a character set containing only characters that exist in both the receiver and *`otherSet`*.

```
public NSCharacterSet characterSetByIntersectingCharacterSet(NSCharacterSet otherSet)
```

characterSetByInvertingCharacterSet

Returns a character set containing only characters that do not exist in the receiver. Inverting an immutable character set is much more efficient than inverting a mutable character set.

```
public NSCharacterSet characterSetByInvertingCharacterSet()
```

See Also
`invertCharacterSet` (page 318) (NSMutableCharacterSet)

characterSetBySubtractingCharacterSet

Returns a character set containing all the characters in the receiver except for those in *`otherSet`*.

```
public NSCharacterSet characterSetBySubtractingCharacterSet(NSCharacterSet otherSet)
```

characterSetByUnioningCharacterSet

Returns a character set containing all characters that exist in either the receiver or *`otherSet`*.

```
public NSCharacterSet characterSetByUnioningCharacterSet(NSCharacterSet otherSet)
```

isSupersetOfSet

Returns `true` if the receiving character set is a superset of *`theOtherSet`*, `false` if it isn't.

```
public boolean isSupersetOfSet(NSCharacterSet theOtherSet)
```

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

NSClassDescription

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
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.

Method implementations for all instance methods of NSClassDescription must be provided by concrete subclasses. 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` (page 107).

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

Constructors

`NSClassDescription` (page 106)
Creates an empty NSClassDescription.

Working with Class Descriptions

`classDescriptionForClass` (page 106)
Returns the NSClassDescription for *aClass*.

`invalidateClassDescriptionCache` (page 107)
Removes all NSClassDescriptions from the cache. You should rarely need to invoke this method. Use it whenever a registered NSClassDescription might be replaced by a different version, such as when you have loaded a new provider of NSClassDescriptions, or when you are about to remove a provider of NSClassDescriptions.

`registerClassDescription` (page 107)
Registers an NSClassDescription object for *aClass* in the NSClassDescription cache.

Attribute Keys

`attributeKeys` (page 107)
Overridden by subclasses to return an array of Strings containing the names of immutable values that instances of this class contain.

Relationship Keys

`inverseForRelationshipKey` (page 107)
Overridden by subclasses to return the name of the inverse relationship from the relationship specified by *relationshipKey*. 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 108)
Overridden by subclasses to return the keys for the to-many relationship properties of the receiver.

`toOneRelationshipKeys` (page 108)
Overridden by subclasses to return the keys for the to-one relationship properties of the receiver.

Constructors

NSClassDescription

Creates an empty NSClassDescription.

```
public NSClassDescription()
```

Discussion
You should create instances of concrete subclasses instead of NSClassDescription.

Static Methods

classDescriptionForClass

Returns the NSClassDescription for *aClass*.

```
public static NSClassDescription classDescriptionForClass(Class aClass)
```

Discussion
If a class description for *aClass* is not found, the method posts a `ClassDescriptionNeededForClassNotification` (page 108) on behalf of *aClass*, allowing an observer to register a class description. The method then checks for a class description again. Returns `null` if a class description is still not found.

invalidateClassDescriptionCache

Removes all NSClassDescriptions from the cache. You should rarely need to invoke this method. Use it whenever a registered NSClassDescription might be replaced by a different version, such as when you have loaded a new provider of NSClassDescriptions, or when you are about to remove a provider of NSClassDescriptions.

```
public static void invalidateClassDescriptionCache()
```

registerClassDescription

Registers an NSClassDescription object for *aClass* in the NSClassDescription cache.

```
public static void registerClassDescription(NSClassDescription description, Class
    aClass)
```

Discussion
You should rarely need to directly invoke this method.

Instance Methods

attributeKeys

Overridden by subclasses to return an array of Strings containing the names of immutable values that instances of this class contain.

```
public NSArray attributeKeys()
```

Discussion
For example, a class description that describes Movie objects could return the attribute keys `title`, `dateReleased`, and `rating`.

See Also
`toManyRelationshipKeys` (page 108)
`toOneRelationshipKeys` (page 108)

inverseForRelationshipKey

Overridden by subclasses to return the name of the inverse relationship from the relationship specified by *relationshipKey*. 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.

```
public String inverseForRelationshipKey(String relationshipKey)
```

Discussion
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`.

toManyRelationshipKeys

Overridden by subclasses to return the keys for the to-many relationship properties of the receiver.

```
public NSArray toManyRelationshipKeys()
```

Discussion
To-many relationship properties contain arrays of objects.

See Also
`attributeKeys` (page 107)
`toOneRelationshipKeys` (page 108)

toOneRelationshipKeys

Overridden by subclasses to return the keys for the to-one relationship properties of the receiver.

```
public NSArray toOneRelationshipKeys()
```

Discussion
To-one relationship properties are other objects.

See Also
`attributeKeys` (page 107)
`toManyRelationshipKeys` (page 108)

Notifications

ClassDescriptionNeededForClassNotification

Posted by `classDescriptionForClass` (page 106) when a class description cannot be found for a class.

After the notification is processed, `classDescriptionForClass` (page 106) 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.

NSCloneCommand

Inherits from	NSScriptCommand : NSObject
Package:	com.apple.cocoa.foundation
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 Clone 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:` messages.

Tasks

Constructors

`NSCloneCommand` (page 110)
Returns an NSCloneCommand with no data.

Working with Specifiers

`keySpecifier` (page 110)
Returns a specifier for the object or objects to be cloned.

`setReceiversSpecifier` (page 110)
Sets the receiver's object specifier; when evaluated, the specifier indicates the receiver or receivers of the Clone command.

Constructors

NSCloneCommand

Returns an NSCloneCommand with no data.

```
public NSCloneCommand()
```

Discussion
Do not use this constructor.

Initializes an instance of NSCloneCommand with the command description supplied by *aCommandDescription*.

```
public NSCloneCommand(NSScriptCommandDescription aCommandDescription)
```

Discussion
Note that such an instance has no receiver specifier, arguments, or direct parameter and is not a fully functional command.

Instance Methods

keySpecifier

Returns a specifier for the object or objects to be cloned.

```
public NSScriptObjectSpecifier keySpecifier()
```

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; for example, if you send the specifier a `containerSpecifier` (page 544) message, the result is `null`.

setReceiversSpecifier

Sets the receiver's object specifier; when evaluated, the specifier indicates the receiver or receivers of the Clone command.

```
public void setReceiversSpecifier(NSScriptObjectSpecifier receiversRef)
```

Discussion
This method overrides `setReceiversSpecifier` (page 525) 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`.

NSCloseCommand

Inherits from	NSScriptCommand : NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Cocoa Scripting Guide

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

Constructors

`NSCloseCommand` (page 111)
Returns an NSCloseCommand with no data.

Accessing Save Options

`saveOptions` (page 112)
Returns a constant indicating how to deal with closing any modified documents.

Constructors

NSCloseCommand

Returns an NSCloseCommand with no data.

```
public NSCloseCommand()
```

Discussion
Do not use this constructor.

Initializes an instance of NSCloseCommand with the command description supplied by *aScriptCommandDescription*.

```
public NSCloseCommand(NSScriptCommandDescription aScriptCommandDescription)
```

Discussion
Note that such an instance has no receiver specifier, arguments, or direct parameter and is not a fully functional command.

Instance Methods

saveOptions

Returns a constant indicating how to deal with closing any modified documents.

```
public int saveOptions()
```

Discussion
The default value returned is `SaveOptionsAsk`. See "Constants" (page 112) for a list of possible return values.

Constants

The `saveOptions` (page 112) method returns one of the following constants to indicate how to deal with saving any modified documents:

Constant	Description
`SaveOptionsYes`	Indicates a modified document should be saved on closing without asking the user.
`SaveOptionsNo`	Indicates a modified document should not be saved on closing.
`SaveOptionsAsk`	Indicates the user should be asked before saving any modified documents on closing.

NSCoder

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Archives and Serializations Programming Guide for Cocoa

Overview

The NSCoder abstract class declares the interface used by concrete subclasses to transfer objects and other 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, and NSKeyedUnarchiver. 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, arrays, structures, and strings. It does not handle types whose implementation varies across platforms. 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.

Tasks

Constructors

`NSCoder` (page 116)
Creates an empty NSCoder.

Testing Coder

`allowsKeyedCoding` (page 116)
Returns a Boolean value that indicates whether the receiver supports keyed coding of objects.

`containsValueForKey` (page 116)
Returns a Boolean value that indicates whether an encoded value is available for a string.

Encoding Data

`encodeBoolForKey` (page 121)
Encodes *boolv* and associates it with the string *key*.

`encodeByte` (page 121)
Encodes *aByte*.

`encodeByteForKey` (page 121)
Encodes *bytev* and associates it with the string *key*.

`encodeChar` (page 122)
Encodes *aChar*.

`encodeCharForKey` (page 122)
Encodes *charv* and associates it with the string *key*.

`encodeConditionalObjectForKey` (page 122)
Conditionally encodes a reference to *objv* and associates it with the string *key* only if *objv* has been unconditionally encoded with `encodeObjectForKey` (page 125).

`encodeDataObject` (page 122)
Encodes the NSData object *data*.

`encodeDouble` (page 123)
Encodes *aDouble*.

`encodeDoubleForKey` (page 123)
Encodes *realv* and associates it with the string *key*.

`encodeFloat` (page 123)
Encodes *aFloat*.

`encodeFloatForKey` (page 123)
Encodes *realv* and associates it with the string *key*.

`encodeIntForKey` (page 124)
Encodes *intv* and associates it with the string *key*.

`encodeInt` (page 124)
Encodes *anInt*.

`encodeLong` (page 124)
Encodes *aLong*.

`encodeLongForKey` (page 124)
Encodes *longv* and associates it with the string *key*.

`encodeObject` (page 125)
Encodes *object*.

`encodeObjectForKey` (page 125)
Encodes the object *objv* and associates it with the string *key*.

`encodeShort` (page 125)
Encodes *aShort*.

`encodeShortForKey` (page 125)
Encodes *shortv* and associates it with the string *key*.

Decoding Data

`decodeBoolForKey` (page 117)
Decodes and returns a boolean value that was previously encoded with `encodeBoolForKey` (page 121) and associated with the string *key*.

`decodeByte` (page 117)
Decodes and returns a `byte` value that was previously encoded with `encodeByte` (page 121).

`decodeByteForKey` (page 117)
Decodes and returns a `byte` value that was previously encoded with `encodeByteForKey` (page 121) and associated with the string *key*.

`decodeChar` (page 117)
Decodes and returns a `char` value that was previously encoded with `encodeChar` (page 122).

`decodeCharForKey` (page 118)
Decodes and returns a `char` value that was previously encoded with `encodeCharForKey` (page 122) and associated with the string *key*.

`decodeDataObject` (page 118)
Decodes and returns an NSData object that was previously encoded with `encodeDataObject` (page 122). Subclasses must override this method.

`decodeDouble` (page 118)
Decodes and returns a `double` value that was previously encoded with `encodeDouble` (page 123).

`decodeDoubleForKey` (page 118)
Decodes and returns a `double` value that was previously encoded with either `encodeFloatForKey` (page 123) or `encodeDoubleForKey` (page 123) and associated with the string *key*.

`decodeFloat` (page 119)
Decodes and returns a `float` value that was previously encoded with `encodeFloat` (page 123).

`decodeFloatForKey` (page 119)
Decodes and returns a `float` value that was previously encoded with `encodeFloatForKey` (page 123) or `encodeDoubleForKey` (page 123) and associated with the string *key*.

`decodeInt` (page 119)
Decodes and returns an `int` value that was previously encoded with `encodeInt` (page 124).

`decodeIntForKey` (page 119)
Decodes and returns an `int` value that was previously encoded with `encodeIntForKey` (page 124), `encodeShortForKey` (page 125), or `encodeLongForKey` (page 124) and associated with the string *key*.

`decodeLong` (page 119)
Decodes and returns a `long` value that was previously encoded with `encodeLong` (page 124).

`decodeLongForKey` (page 120)
Decodes and returns a `long` value that was previously encoded with `encodeShortForKey` (page 125), `encodeIntForKey` (page 124), or `encodeLongForKey` (page 124) and associated with the string *key*.

`decodeObject` (page 120)
Decodes an object that was previously encoded with any of the `encode...` methods.

`decodeObjectForKey` (page 120)
Decodes and returns an object that was previously encoded with `encodeObjectForKey` (page 125) or `encodeConditionalObjectForKey` (page 122) and associated with the string *key*.

`decodeShort` (page 120)
Decodes and returns a `short` value that was previously encoded with `encodeShort` (page 125).

`decodeShortForKey` (page 121)
Decodes and returns a `short` value that was previously encoded with `encodeShortForKey` (page 125), `encodeIntForKey` (page 124), or `encodeLongForKey` (page 124) and associated with the string *`key`*.

Getting Version Information

`systemVersion` (page 126)
During encoding, this method should return the system version currently in effect.

`versionForClassName` (page 126)
Returns the version in effect for the class named *`className`* or `NSArray.NotFound` if no class named *`className`* exists.

Constructors

NSCoder

Creates an empty NSCoder.

```
public NSCoder()
```

Discussion
NSCoder is an abstract class, so use one of the concrete subclasses instead.

Instance Methods

allowsKeyedCoding

Returns a Boolean value that indicates whether the receiver supports keyed coding of objects.

```
public boolean allowsKeyedCoding()
```

Discussion
The default implementation returns `false`. Concrete subclasses that support keyed coding, such as NSKeyedArchiver, need to override this method to return `true`.

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

containsValueForKey

Returns a Boolean value that indicates whether an encoded value is available for a string.

```
public boolean containsValueForKey(String 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.

decodeBoolForKey

Decodes and returns a boolean value that was previously encoded with `encodeBoolForKey` (page 121) and associated with the string *key*.

```
public boolean decodeBoolForKey(String key)
```

Discussion
Subclasses must override this method if they perform keyed coding.

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

decodeByte

Decodes and returns a `byte` value that was previously encoded with `encodeByte` (page 121).

```
public byte decodeByte()
```

Discussion
Subclasses must override this method.

decodeByteForKey

Decodes and returns a `byte` value that was previously encoded with `encodeByteForKey` (page 121) and associated with the string *key*.

```
public byte decodeByteForKey(String key)
```

Discussion
Subclasses must override this method if they perform keyed coding.

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

decodeChar

Decodes and returns a `char` value that was previously encoded with `encodeChar` (page 122).

```
public char decodeChar()
```

Discussion
Subclasses must override this method.

decodeCharForKey

Decodes and returns a `char` value that was previously encoded with `encodeCharForKey` (page 122) and associated with the string *key*.

```
public char decodeCharForKey(String key)
```

Discussion
Subclasses must override this method if they perform keyed coding.

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

decodeDataObject

Decodes and returns an NSData object that was previously encoded with `encodeDataObject` (page 122). Subclasses must override this method.

```
public NSData decodeDataObject()
```

Discussion
The implementation of your overriding method must match the implementation of your `encodeDataObject` (page 122) method. For example, a typical `encodeDataObject` (page 122) 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.

decodeDouble

Decodes and returns a `double` value that was previously encoded with `encodeDouble` (page 123).

```
public double decodeDouble()
```

Discussion
Subclasses must override this method.

decodeDoubleForKey

Decodes and returns a `double` value that was previously encoded with either `encodeFloatForKey` (page 123) or `encodeDoubleForKey` (page 123) and associated with the string *key*.

```
public double decodeDoubleForKey(String key)
```

Discussion
Subclasses must override this method if they perform keyed coding.

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

decodeFloat

Decodes and returns a `float` value that was previously encoded with `encodeFloat` (page 123).

```
public float decodeFloat()
```

Discussion
Subclasses must override this method.

decodeFloatForKey

Decodes and returns a `float` value that was previously encoded with `encodeFloatForKey` (page 123) or `encodeDoubleForKey` (page 123) and associated with the string *key*.

```
public float decodeFloatForKey(String 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 throws a `RangeException`. Subclasses must override this method if they perform keyed coding.

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

decodeInt

Decodes and returns an `int` value that was previously encoded with `encodeInt` (page 124).

```
public int decodeInt()
```

Discussion
Subclasses must override this method.

decodeIntForKey

Decodes and returns an `int` value that was previously encoded with `encodeIntForKey` (page 124), `encodeShortForKey` (page 125), or `encodeLongForKey` (page 124) and associated with the string *key*.

```
public int decodeIntForKey(String key);
```

Discussion
If the encoded integer does not fit into the default integer size, the method throws a `RangeException`. Subclasses must override this method if they perform keyed coding.

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

decodeLong

Decodes and returns a `long` value that was previously encoded with `encodeLong` (page 124).

```
public long decodeLong()
```

Discussion
Subclasses must override this method.

decodeLongForKey

Decodes and returns a `long` value that was previously encoded with `encodeShortForKey` (page 125), `encodeIntForKey` (page 124), or `encodeLongForKey` (page 124) and associated with the string *key*.

```
public long decodeLongForKey(String key)
```

Discussion
Subclasses must override this method if they perform keyed coding.

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

decodeObject

Decodes an object that was previously encoded with any of the `encode...` methods.

```
public Object decodeObject()
```

Discussion
Subclasses may need to override this method.

See Also
`encodeObject` (page 125)

decodeObjectForKey

Decodes and returns an object that was previously encoded with `encodeObjectForKey` (page 125) or `encodeConditionalObjectForKey` (page 122) and associated with the string *key*.

```
public Object decodeObjectForKey(String key)
```

Discussion
Subclasses must override this method if they perform keyed coding.

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

decodeShort

Decodes and returns a `short` value that was previously encoded with `encodeShort` (page 125).

```
public short decodeShort()
```

Discussion
Subclasses must override this method.

decodeShortForKey

Decodes and returns a `short` value that was previously encoded with `encodeShortForKey` (page 125), `encodeIntForKey` (page 124), or `encodeLongForKey` (page 124) and associated with the string *key*.

```
public short decodeShortForKey(String key)
```

Discussion
If the encoded integer does not fit into the default integer size, the method throws a `RangeException`. Subclasses must override this method if they perform keyed coding.

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

encodeBoolForKey

Encodes *boolv* and associates it with the string *key*.

```
public void encodeBoolForKey(boolean boolv, String 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 117)

encodeByte

Encodes *aByte*.

```
public void encodeByte(byte aByte)
```

Discussion
Subclasses must override this method.

This method must be matched by a subsequent `decodeByte` (page 117) message.

encodeByteForKey

Encodes *bytev* and associates it with the string *key*.

```
public void encodeByteForKey(byte bytev, String key)
```

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

See Also
`decodeByteForKey` (page 117)

encodeChar

Encodes *aChar*.

```
public void encodeChar(char aChar)
```

Discussion
Subclasses must override this method.

This method must be matched by a subsequent `decodeChar` (page 117) message.

encodeCharForKey

Encodes *charv* and associates it with the string *key*.

```
public void encodeCharForKey(char charv, String key)
```

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

See Also
`decodeCharForKey` (page 118)

encodeConditionalObjectForKey

Conditionally encodes a reference to *objv* and associates it with the string *key* only if *objv* has been unconditionally encoded with `encodeObjectForKey` (page 125).

```
public void encodeConditionalObjectForKey(Object objv, String key)
```

Discussion
Subclasses must override this method if they support keyed coding.

The encoded object is decoded with the `decodeObjectForKey` (page 120) method. If *objv* was never encoded unconditionally, `decodeObjectForKey` (page 120) returns `null` in place of *objv*.

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

encodeDataObject

Encodes the NSData object *data*.

```
public void encodeDataObject(NSData data)
```

Discussion
Subclasses must override this method.

This method must be matched by a subsequent `decodeDataObject` (page 118) message.

See Also
`encodeObject` (page 125)

encodeDouble

Encodes *aDouble*.

```
public void encodeDouble(double aDouble)
```

Discussion
Subclasses must override this method.

This method must be matched by a subsequent `decodeDouble` (page 118) message.

encodeDoubleForKey

Encodes *realv* and associates it with the string *key*.

```
public void encodeDoubleForKey(double realv, String 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 118)
`decodeFloatForKey` (page 119)

encodeFloat

Encodes *aFloat*.

```
public void encodeFloat(float aFloat)
```

Discussion
Subclasses must override this method.

This method must be matched by a subsequent `decodeFloat` (page 119) message.

encodeFloatForKey

Encodes *realv* and associates it with the string *key*.

```
public void encodeFloatForKey(float realv, String 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 119)

decodeDoubleForKey (page 118)

encodeInt

Encodes *anInt*.

```
public void encodeInt(int anInt)
```

Discussion
Subclasses must override this method.

This method must be matched by a subsequent decodeInt (page 119) message.

encodeIntForKey

Encodes *intv* and associates it with the string *key*.

```
public void encodeIntForKey(int intv, String 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 119)
decodeLongForKey (page 120)
decodeShortForKey (page 121)

encodeLong

Encodes *aLong*.

```
public void encodeLong(long aLong)
```

Discussion
Subclasses must override this method.

This method must be matched by a subsequent decodeLong (page 119) message.

encodeLongForKey

Encodes *longv* and associates it with the string *key*.

```
public void encodeLongForKey(long longv, String key)
```

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

See Also
decodeLongForKey (page 120)
decodeIntForKey (page 119)
decodeShortForKey (page 121)

encodeObject

Encodes *object*.

```
public void encodeObject(Object object)
```

Discussion
Subclasses must override this method. 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 120) message.

encodeObjectForKey

Encodes the object *objv* and associates it with the string *key*.

```
public void encodeObjectForKey(Object objv, String 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 120)

encodeShort

Encodes *aShort*.

```
public void encodeShort(short aShort)
```

Discussion
Subclasses must override this method.

This method must be matched by a subsequent decodeShort (page 120) message.

encodeShortForKey

Encodes *shortv* and associates it with the string *key*.

```
public void encodeShortForKey(short shortv, String key)
```

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

See Also
`decodeShortForKey` (page 121)
`decodeLongForKey` (page 120)
`decodeIntForKey` (page 119)

systemVersion

During encoding, this method should return the system version currently in effect.

```
public int 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.

versionForClassName

Returns the version in effect for the class named *`className`* or `NSArray.NotFound` if no class named *`className`* exists.

```
public abstract int versionForClassName(String className)
```

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.

Constants

The following exceptions may be thrown when an error is encountered:

Constant	Description
`InconsistentArchiveException`	Archive contains invalid data and may be corrupted
`InvalidArchiveOperationException`	Attempted to perform an illegal action on a keyed archive, such as trying to encode a value after finished encoding
`InvalidUnarchive-OperationException`	Attempted to perform an illegal action on a keyed archive, such as trying to decode a `float` value as a boolean

NSCountCommand

Inherits from	NSScriptCommand : NSObject
Package:	com.apple.cocoa.foundation
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.

Tasks

Constructors

`NSCountCommand` (page 127)
Returns an NSCountCommand with no data.

Constructors

NSCountCommand

Returns an NSCountCommand with no data.

```
public NSCountCommand()
```

Discussion
Do not use this constructor.

Initializes an instance of NSCountCommand with the command description supplied by *aScriptCommandDescription*.

```
public NSCountCommand(NSScriptCommandDescription aScriptCommandDescription)
```

Discussion
Note that such an instance has no receiver specifier, arguments, or direct parameter and is not a fully functional command.

NSCreateCommand

Inherits from	NSScriptCommand : NSObject
Package:	com.apple.cocoa.foundation
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.

If an NSCreateCommand 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 `false` when sent an `isLocationToRequiredToCreateForKey:toManyRelationshipKey` message, the NSCreateCommand creates a new object and sends the receiver an `insertValueAtIndexInPropertyWithKey` (page 696) 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

Constructors

`NSCreateCommand` (page 130)
Returns an NSCreateCommand with no data.

Getting Information About a Create Command

`createClassDescription` (page 130)
Returns the class description for the class that is to be created.

`resolvedKeyDictionary` (page 130)
Returns a dictionary that contains the properties that were specified in the Make Apple event script command that has been converted to this NSCreateCommand.

Constructors

NSCreateCommand

Returns an NSCreateCommand with no data.

```
public NSCreateCommand()
```

Discussion
Do not use this constructor.

Initializes an instance of NSCreateCommand with the command description supplied by *aScriptCommandDescription*.

```
public NSCreateCommand(NSScriptCommandDescription aScriptCommandDescription)
```

Discussion
Note that such an instance has no receiver specifier, arguments, or direct parameter and is not a fully functional command.

Instance Methods

createClassDescription

Returns the class description for the class that is to be created.

```
public NSScriptClassDescription createClassDescription()
```

resolvedKeyDictionary

Returns a dictionary that contains the properties that were specified in the Make Apple event script command that has been converted to this NSCreateCommand.

```
public NSDictionary resolvedKeyDictionary()
```

Discussion
The keys in the 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 Create command are specified in the core suite definition, `NSCoreSuite.scriptSuite`.

NSData

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.foundation
Companion guides	Binary Data Programming Guide for Cocoa Property List Programming Guide

Class at a Glance

An NSData object stores immutable data in the form of bytes. The size of the data is subject to a 2GB limit.

Principal Attributes

- A count of the number of bytes in the data object
- The sequence of bytes contained in the data object

`NSData` (page 133)
Creates a data object.

Commonly Used Methods

`length` (page 134)
Returns the number of bytes contained by the data object.

Primitive Methods

`length` (page 134)
Returns the number of bytes contained by the data object.

Overview

NSData and its 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.

The mutable subclass of NSData is NSMutableData (page 321).

Tasks

Constructors

`NSData` (page 133)

Creating Data Objects

`dataWithContentsOfMappedFile` (page 133)
Creates and returns a data object from the mapped file specified by *file*.

Accessing Data

`bytes` (page 134)
Returns a byte array of *length* bytes from the receiver's contents starting at *start*.

`subdataWithRange` (page 134)
Returns a data object containing a copy of the receiver's bytes that fall within the limits specified by *range*.

Testing Data

`isEqualToData` (page 134)
Compares the receiving data object to *otherData*.

`length` (page 134)
Returns the number of bytes contained in the receiver.

Storing Data

`writeToURL` (page 135)
Writes the bytes in the receiver to the location specified by *aURL*.

Constructors

NSData

```
public NSData()
```

Discussion
Creates an empty data object. This method is declared primarily for the use of mutable subclasses of NSData.

```
public NSData(byte[] bytes, int start, int length)
```

Discussion
Creates a data object with *length* bytes from the buffer *bytes*, starting at *start*.

```
public NSData(byte[] bytes)
```

Discussion
Creates a data object with all the data in the buffer *bytes*.

```
public NSData(java.io.File aFile)
```

Discussion
Creates a data object with the data from the file specified by *aFile*.

```
public NSData(java.net.URL aURL)
```

Discussion
Creates a data object with the data from the location specified by *aURL*.

```
public NSData(NSData aData)
```

Discussion
Creates a data object containing the contents of another data object, *aData*.

```
public NSData(String aString)
```

Discussion
Deprecated. To create an NSData from a property list use `propertyListFromString` (page 461); to initialize an NSData from a file, pass either a `java.io.file` or a `java.net.url` object.

Static Methods

dataWithContentsOfMappedFile

Creates and returns a data object from the mapped file specified by *file*.

```
public static NSData dataWithContentsOfMappedFile(java.io.File file)
```

Discussion
Returns null if the data object could not be creates. 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.

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.

Instance Methods

bytes

Returns a byte array of *length* bytes from the receiver's contents starting at *start*.

```
public byte[] bytes(int start, int length)
```

isEqualToData

Compares the receiving data object to *otherData*.

```
public boolean isEqualToData(NSData otherData)
```

Discussion
If the contents of *otherData* are equal to the contents of the receiver, this method returns `true`. If not, it returns `false`. 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.

length

Returns the number of bytes contained in the receiver.

```
public int length()
```

subdataWithRange

Returns a data object containing a copy of the receiver's bytes that fall within the limits specified by *range*.

```
public NSData subdataWithRange(NSRange range)
```

Discussion
If *range* isn't within the receiver's range of bytes, a `RangeException` is thrown.

For example, the following code excerpt initializes a data object, `data2`, to contain a subrange of `data1`:

```
String myString = "ABCDEFG";
range = new NSRange(2,4);
NSData data1 = new NSData(myString.getBytes());
NSData data2 = data1.subdataWithRange(range);
```

The result of this excerpt is that `data2` contains "`CDEF`".

writeToURL

Writes the bytes in the receiver to the location specified by *aURL*.

```
public boolean writeToURL(java.net.URL aURL, boolean atomically)
```

Discussion
If *atomically* is `true`, the data is written to a backup location, and then, assuming no errors occur, the backup location is renamed to the specified name. Otherwise, the data is written directly to the specified location. *atomically* is ignored if *aURL* is not of a type the supports atomic writes.

This method returns `true` if the operation succeeds; otherwise, it returns `false`.

NSDate

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.foundation
Companion guides	Date and Time Programming Guide for Cocoa Property List Programming Guide

Class at a Glance

An NSDate object stores a date and time that can be compared to other dates and times.

Principal Attributes

- Seconds since absolute reference date (1 January 2001, GMT)

Commonly Used Methods

`earlierDate` (page 142)
Compares the receiver to the argument and returns the earlier of the two.

`isEqualToDate` (page 143)
Returns `true` if the receiver and the argument are equal.

`laterDate` (page 143)
Compares the receiver to the argument and returns the later of the two.

`timeIntervalSinceNow` (page 144)
Returns the number of seconds difference between the receiver and the current date and time.

Primitive Methods

`timeIntervalSinceReferenceDate` (page 144)

Overview

NSDate objects represent a single point in time. 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.

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. Its subclass NSGregorianDate offers date objects that are suitable for representing dates according to western calendrical systems.

NSDate's sole primitive method, `timeIntervalSinceReferenceDate` (page 144), provides the basis for all the other methods in the NSDate interface. This method returns a time value relative to an absolute reference date.

Subclassing Notes

The major reason for subclassing NSDate is to create a class that expresses a calendrical system other than the western, Gregorian calendar (for which Cocoa provides the NSCalendarDate class). 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 144) instance method to provide the correct date and time value based on your instance variable.

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. See the NSCalendarDate class for examples of such methods.

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 method `timeIntervalSinceReferenceDate` (page 144) . 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.

Tasks

Constructors

`NSDate` (page 140)
Creates an NSDate set to the current date and time.

Creating an NSDate Instance

`distantFuture` (page 141)
Creates and returns an object representing a date in the distant future (in terms of centuries).

`distantPast` (page 141)
Creates and returns an object representing a date in the distant past (in terms of centuries).

`dateByAddingTimeInterval` (page 142)
Returns an NSDate object that is set to a specified number of seconds, *`seconds`*, relative to the receiver.

Comparing Dates

`isEqualToDate` (page 143)
Returns `true` if the two objects compared are NSDate objects and are exactly equal to each other, `false` if one of the objects is not of the NSDate class or their date and time values differ.

`earlierDate` (page 142)
Compares the receiver date to *`anotherDate`*, using `timeIntervalSinceDate` (page 144), and returns the earlier of the two.

`laterDate` (page 143)
Compares the receiver to *`anotherDate`*, using `timeIntervalSinceDate` (page 144), and returns the later of the two.

`compare` (page 142)
Compares the receiving date to *`anotherDate`*, using `timeIntervalSinceDate` (page 144), and returns a value of type `int`.

`equals` (page 143)
Returns `true` if *`anObject`* is an instance of NSDate and satisfies `isEqualToDate` (page 143).

`hashCode` (page 143)
Returns a hash value for the receiver.

Getting Time Intervals

`timeIntervalSinceDate` (page 144)
Returns the interval between the receiver and *`anotherDate`*.

`timeIntervalSinceNow` (page 144)
Returns the interval between the receiver and the current date and time.

`currentTimeIntervalSinceReferenceDate` (page 141)
Returns the interval between the system's absolute reference date (the first instant of 1 January 2001, GMT) and the current date and time

`timeIntervalSinceReferenceDate` (page 144)
Returns the interval between the receiver and the system's absolute reference date, 1 January 2001, GMT.

Representing Dates as Strings

`toString` (page 144)
Returns a string representation of the receiver.

Working with Milliseconds

`millisecondsToTimeInterval` (page 141)
Converts a time span, *`milliseconds`*, measured in milliseconds, to a time interval, which is in seconds.

`timeIntervalToMilliseconds` (page 142)
Converts a time interval, *`seconds`*, which is in seconds, to milliseconds.

Constructors

NSDate

Creates an NSDate set to the current date and time.

```
public NSDate()
```

Creates an NSDate relative to the absolute reference date (the first instant of 1 January 2001, GMT) by the specified number of *`seconds`* (plus or minus).

```
public NSDate(double seconds)
```

Creates an NSDate relative to *`refDate`* by a specified number of *`seconds`* (plus or minus).

```
public NSDate(double seconds, NSDate refDate)
```

Static Methods

currentTimeIntervalSinceReferenceDate

Returns the interval between the system's absolute reference date (the first instant of 1 January 2001, GMT) and the current date and time

```
public static double currentTimeIntervalSinceReferenceDate()
```

Discussion

.

See Also

`timeIntervalSinceReferenceDate` (page 144)
`timeIntervalSinceDate` (page 144)
`timeIntervalSinceNow` (page 144)

distantFuture

Creates and returns an object representing a date in the distant future (in terms of centuries).

```
public static Object distantFuture()
```

Discussion

You can pass this value when an NSDate is required to have the date argument essentially ignored. For example, the NSWindow method `nextEventMatchingMask` returns `null` 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.

See Also

`distantPast` (page 141)

distantPast

Creates and returns an object representing a date in the distant past (in terms of centuries).

```
public static Object distantPast()
```

Discussion

You can use this object in your code as a control date, a guaranteed temporal boundary.

See Also

`distantFuture` (page 141)

millisecondsToTimeInterval

Converts a time span, *`milliseconds`*, measured in milliseconds, to a time interval, which is in seconds.

```
public static double millisecondsToTimeInterval(long milliseconds)
```

timeIntervalToMilliseconds

Converts a time interval, *seconds*, which is in seconds, to milliseconds.

```
public static long timeIntervalToMilliseconds(double seconds)
```

Instance Methods

compare

Compares the receiving date to *anotherDate*, using `timeIntervalSinceDate` (page 144), and returns a value of type `int`.

```
public int compare(NSDate anotherDate)
```

Discussion
If the two dates are exactly equal to each other, this method returns `OrderedSame`. If the receiving object in the comparison is more recent than *anotherDate*, the method returns `OrderedDescending`. If the receiving object is older, this method returns `OrderedAscending`.

This method detects subsecond differences between dates. If you want to compare dates with a less fine granularity, use `timeIntervalSinceDate` (page 144) to compare the two dates or use NSGregorianDate objects instead.

See Also
`earlierDate` (page 142)
`laterDate` (page 143)

dateByAddingTimeInterval

Returns an NSDate object that is set to a specified number of seconds, *seconds*, relative to the receiver.

```
public NSDate dateByAddingTimeInterval(double seconds)
```

Discussion
Use a negative value for seconds to have the returned object specify a date before the receiver. The date returned might have a representation different from the receiver's.

See Also
`timeIntervalSinceDate` (page 144)

earlierDate

Compares the receiver date to *anotherDate*, using `timeIntervalSinceDate` (page 144), and returns the earlier of the two.

```
public NSDate earlierDate(NSDate anotherDate)
```

See Also
`compare` (page 142)

`laterDate` (page 143)

equals

Returns `true` if *`anObject`* is an instance of NSDate and satisfies `isEqualToDate` (page 143).

```
public boolean equals(Object anObject)
```

Discussion
Returns `false` otherwise.

hashCode

Returns a hash value for the receiver.

```
public int hashCode()
```

Discussion
The hash value is the integer value of the time interval returned from `timeIntervalSinceReferenceDate` (page 144).

isEqualToDate

Returns `true` if the two objects compared are NSDate objects and are exactly equal to each other, `false` if one of the objects is not of the NSDate class or their date and time values differ.

```
public boolean isEqualToDate(NSDate anotherDate)
```

Discussion
This method detects subsecond differences between dates. If you want to compare dates with a less fine granularity, either use `timeIntervalSinceDate` (page 144) to compare the two dates or use NSGregorianDate objects instead.

See Also
`compare` (page 142)
`earlierDate` (page 142)
`laterDate` (page 143)

laterDate

Compares the receiver to *`anotherDate`*, using `timeIntervalSinceDate` (page 144), and returns the later of the two.

```
public NSDate laterDate(NSDate anotherDate)
```

See Also
`compare` (page 142)
`earlierDate` (page 142)

timeIntervalSinceDate

Returns the interval between the receiver and *anotherDate*.

```
public double timeIntervalSinceDate(NSDate anotherDate)
```

Discussion
If the receiver is earlier than *anotherDate*, the return value is negative.

See Also
timeIntervalSinceNow (page 144)
currentTimeIntervalSinceReferenceDate (page 141)

timeIntervalSinceNow

Returns the interval between the receiver and the current date and time.

```
public double timeIntervalSinceNow()
```

Discussion
If the receiver is earlier than the current date and time, the return value is negative.

See Also
timeIntervalSinceDate (page 144)
currentTimeIntervalSinceReferenceDate (page 141)

timeIntervalSinceReferenceDate

Returns the interval between the receiver and the system's absolute reference date, 1 January 2001, GMT.

```
public double timeIntervalSinceReferenceDate()
```

Discussion
If the receiver is earlier than the absolute reference date, the return value is negative.

This method is the primitive method for NSDate. If you subclass NSDate, you must override this method with your own implementation for it.

See Also
timeIntervalSinceDate (page 144)
timeIntervalSinceNow (page 144)
currentTimeIntervalSinceReferenceDate (page 141)

toString

Returns a string representation of the receiver.

```
public String toString()
```

Constants

NSDate provides the following constants as a convenience:

Constant	Type	Description
`DateFor1970`	NSDate	Date object for 1 January 1970
`TimeIntervalSince1970`	double	Seconds from 1 January 1970 to reference date, 1 January 2001

NSDecimalMappingBehavior

Inherits from	Object
Package:	com.apple.cocoa.foundation
Companion guide	Number and Value Programming Topics for Cocoa

Overview

The NSDecimalMappingBehavior class controls mapping behavior (rounding mode and not-a-number treatment) used for converting between the Objective-C class NSDecimalNumber and the Java class `java.math.BigDecimal`. The mapping behavior is automatically invoked when a Java object invokes an Objective-C method that takes (or returns) an NSDecimalNumber value; the Java object sends and receives the value in the form of a `java.math.BigDecimal` object.

Tasks

Modifying Behavior

`defaultRoundingMode` (page 148)
: Returns the way arithmetic methods round off.

`setDefaultRoundingMode` (page 148)
: Sets the way arithmetic methods round off to *`roundingMode`*.

Error Handling

`getNotANumberValue` (page 148)
: Returns a value that specifies no number.

`setNotANumberValue` (page 148)
: Sets a value that specifies no number.

`setShouldRaiseForNotANumberArgument` (page 148)
: Sets whether an exception is thrown when `NaN` is passed to a method.

`shouldRaiseForNotANumberArgument` (page 149)
: Returns whether an exception is thrown when `NaN` is passed to a method.

Static Methods

defaultRoundingMode

Returns the way arithmetic methods round off.

```
public static int defaultRoundingMode()
```

Discussion
Possible values are described in "Constants" (page 149).

getNotANumberValue

Returns a value that specifies no number.

```
public static java.math.BigDecimal getNotANumberValue()
```

Discussion
Any arithmetic method receiving this value as an argument returns this value.

This value can be a useful way of handling nonnumeric data in an input file. It can also be a useful response to calculation errors.

setDefaultRoundingMode

Sets the way arithmetic methods round off to *roundingMode*.

```
public static void setDefaultRoundingMode(int roundingMode)
```

Discussion
Possible values for *roundingMode* are described in "Constants" (page 149).

setNotANumberValue

Sets a value that specifies no number.

```
public static void setNotANumberValue(java.math.BigDecimal aBigDecimal)
```

Discussion
Any arithmetic method receiving this value as an argument returns this value.

This value can be a useful way of handling nonnumeric data in an input file. It can also be a useful response to calculation errors.

setShouldRaiseForNotANumberArgument

Sets whether an exception is thrown when `NaN` is passed to a method.

```
public static void setShouldRaiseForNotANumberArgument(boolean flag)
```

Discussion
If *flag* is `true`, an `IllegalArgumentException` is thrown. If *flag* is `false`, the not-a-number value is returned and a `NotANumberConversionNotification` (page 150) is posted. Default is `false`.

See Also
`shouldRaiseForNotANumberArgument` (page 149)

shouldRaiseForNotANumberArgument

Returns whether an exception is thrown when `NaN` is passed to a method.

```
public static boolean shouldRaiseForNotANumberArgument()
```

Discussion
Default is `false`.

See Also
`setShouldRaiseForNotANumberArgument` (page 148)

Constants

The following constants are provided by NSDecimalMappingBehavior:

Constant	Description
`RoundBankers`	Methods round to the closest possible return value. When they are caught halfway between two possibilities, they 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.
`RoundDown`	Methods round their return values down.
`RoundPlain`	Methods round to the closest possible return value. When they are caught halfway between two positive numbers, they round up; when caught between two negative numbers, they round down.
`RoundUp`	Methods round their return values up.

Notifications

DecimalLossOfPrecisionNotification

Posted when loss of precision occurs in converting `java.math.BigDecimal` to NSDecimalNumber. The notification does not contain a notification object or *userInfo* dictionary.

NotANumberConversionNotification

Posted when `NaN` is passed as an argument to a method or the not-a-number value is returned from a method. The notification does not contain a notification object or *`userInfo`* dictionary.

See Also

`setShouldRaiseForNotANumberArgument` (page 148)

NSDeleteCommand

Inherits from	NSScriptCommand : NSObject
Package:	com.apple.cocoa.foundation
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 to perform the operation.When the command is executed, it uses the key-value coding mechanism to remove the specified object or objects from their container.

NSDeleteCommand is part of Cocoa's built-in scripting support. Most applications don't need to subclass NSDeleteCommand or call its methods.

Tasks

Constructors

`NSDeleteCommand` (page 152)
Returns an NSDeleteCommand with no data.

Working with Specifiers

`keySpecifier` (page 152)
Returns a specifier for the object or objects to be deleted.

`setReceiversSpecifier` (page 152)
Sets the receiver's object specifier; when evaluated, the specifier indicates the receiver or receivers of the Delete command.

Constructors

NSDeleteCommand

Returns an NSDeleteCommand with no data.

```
public NSDeleteCommand()
```

Discussion
Do not use this constructor.

Initializes an instance of NSDeleteCommand with the command description supplied by *aScriptCommandDescription*.

```
public NSDeleteCommand(NSScriptCommandDescription aScriptCommandDescription)
```

Discussion
Note that such an instance has no receiver specifier, arguments, or direct parameter and is not a fully functional command.

Instance Methods

keySpecifier

Returns a specifier for the object or objects to be deleted.

```
public NSScriptObjectSpecifier keySpecifier()
```

Discussion
Note that this may be different than the specifier or specifiers set by `setReceiversSpecifier` (page 152).

setReceiversSpecifier

Sets the receiver's object specifier; when evaluated, the specifier indicates the receiver or receivers of the Delete command.

```
public void setReceiversSpecifier(NSScriptObjectSpecifier receiversRef)
```

Discussion
This method overrides `setReceiversSpecifier` (page 525) 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`.

NSDictionary

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.foundation
Companion guides	Collections Programming Topics for Cocoa Property List Programming Guide

Class at a Glance

An NSDictionary object stores an immutable 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

`NSDictionary` (page 155)
Creates a new dictionary.

Commonly Used Methods

`count` (page 157)
Returns the number of objects currently in the dictionary.

`objectForKey` (page 158)
Returns the object that corresponds to the specified key.

`keyEnumerator` (page 157)
Returns an enumerator object that lets you access each key in the dictionary.

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.)

The mutable subclass of NSDictionary is NSMutableDictionary (page 325).

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 `equals` (page 424)).

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.

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 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 during construction (for all dictionaries) or modification (for mutable dictionaries)—add each value object to the dictionary directly, but copy each key argument and add the copy to the dictionary.

NSDictionary's three primitive methods—`count` (page 157), `objectForKey` (page 158), and `keyEnumerator` (page 157)—provide the basis for all of the other methods in its interface. The `count` method returns the number of entries in the dictionary. `objectForKey` returns the value associated with a given key. `keyEnumerator` 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 nonprimitive methods provide convenient ways of accessing multiple entries at once.

Tasks

Constructors

`NSDictionary` (page 155)

Counting Entries

`count` (page 157)
Returns the number of entries in the receiver.

Comparing Dictionaries

`isEqualToDictionary` (page 157)
Compares the receiving dictionary to *otherDictionary*.

Accessing Keys and Values

`allKeys` (page 156)
Returns a new array containing the receiver's keys or an empty array if the receiver has no entries.

`allKeysForObject` (page 156)
Finds all occurrences of the value *anObject* in the receiver and returns a new array with the corresponding keys.

`allValues` (page 156)
Returns a new array containing the receiver's values, or an empty array if the receiver has no entries.

`keyEnumerator` (page 157)
Returns an enumerator object that lets you access each key in the receiver.

`objectEnumerator` (page 158)
Returns an enumerator object that lets you access each value in the receiver.

`objectForKey` (page 158)
Returns an entry's value given its key, or `null` if no value is associated with *aKey*.

`objectsForKeys` (page 158)
Returns the set of objects from the receiver that corresponds to the specified *keys* as an NSArray.

Constructors

NSDictionary

```
public NSDictionary()
```

Discussion
Creates and returns an empty dictionary.

```
public NSDictionary(Object[] objects, Object[] keys)
```

Discussion
Creates a dictionary with entries constructed from the contents of the *objects* and *keys* arrays. This method steps through the *objects* and *keys* arrays, creating entries in the new dictionary as it goes. Each value object is added directly to the dictionary. Each key object is copied, and the copy is added to the dictionary. An `InvalidArgumentException` is thrown if the objects and keys arrays do not have the same number of elements.

```
public NSDictionary(Object anObject, Object aKey)
```

Discussion
Creates a dictionary containing a single object, *anObject*, for a single key, *aKey*.

```
public NSDictionary(NSDictionary otherDictionary)
```

Discussion
Creates a dictionary containing the keys and values found in *otherDictionary*.

Instance Methods

allKeys

Returns a new array containing the receiver's keys or an empty array if the receiver has no entries.

```
public NSArray allKeys()
```

Discussion
The order of the elements in the array isn't defined.

See Also
allValues (page 156)
allKeysForObject (page 156)

allKeysForObject

Finds all occurrences of the value *anObject* in the receiver and returns a new array with the corresponding keys.

```
public NSArray allKeysForObject(Object anObject)
```

Discussion
Each object in the receiver is sent an equals (page 424) message to determine if it's equal to *anObject*. If no object matching *anObject* is found, this method returns an empty array.

See Also
allKeys (page 156)
keyEnumerator (page 157)

allValues

Returns a new array containing the receiver's values, or an empty array if the receiver has no entries.

```
public NSArray allValues()
```

Discussion
The order of the values in the array isn't defined.

See Also
allKeys (page 156)
objectEnumerator (page 158)

count

Returns the number of entries in the receiver.

```
public int count()
```

isEqualToDictionary

Compares the receiving dictionary to *otherDictionary*.

```
public boolean isEqualToDictionary(NSDictionary otherDictionary)
```

Discussion
If the contents of *otherDictionary* are equal to the contents of the receiver, this method returns `true`. If not, it returns `false`.

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 `equals` (page 424) test.

See Also
`equals` (page 424) (NSObject)

keyEnumerator

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

```
public java.util.Enumeration keyEnumerator()
```

Discussion

```
java.util.Enumeration enumerator = myDict.keyEnumerator();

while (enumerator.hasMoreElements()) {{
    Object anObject = enumerator.nextElement();
    /* code to act on each element */
}
```

When this method is used with mutable subclasses of NSDictionary, your code shouldn't modify the entries during enumeration. If you intend to modify the entries, use the `allKeys` (page 156) 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 158) method provides a convenient way to access each value in the dictionary.

See Also
`allKeys` (page 156)
`allKeysForObject` (page 156)
`objectEnumerator` (page 158)
`nextElement` (page 167) (NSEnumerator)

objectEnumerator

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

```
public java.util.Enumeration objectEnumerator()
```

Discussion

```
java.util.Enumeration enumerator = myDict.objectEnumerator();

while (enumerator.hasMoreElements()) {{
    Object anObject = enumerator.nextElement();
    /* code to act on each element */
}
```

When this method is used with mutable subclasses of NSDictionary, your code shouldn't modify the entries during enumeration. If you intend to modify the entries, use the `allValues` (page 156) method to create a "snapshot" of the dictionary's values. Work from this snapshot to modify the values.

See Also
`keyEnumerator` (page 157)
`nextElement` (page 167) (NSEnumerator)

objectForKey

Returns an entry's value given its key, or `null` if no value is associated with *`aKey`*.

```
public Object objectForKey(Object akey)
```

See Also
`allKeys` (page 156)
`allValues` (page 156)

objectsForKeys

Returns the set of objects from the receiver that corresponds to the specified *`keys`* as an NSArray.

```
public NSArray objectsForKeys(NSArray keys, Object anObject)
```

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`*. If an object isn't found in the receiver to correspond to a given key, the marker object, specified by *`anObject`*, is placed in the corresponding element of the returned array.

NSDistributedNotificationCenter

Inherits from	NSNotificationCenter : NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Notification Programming Topics for Cocoa

Class at a Glance

NSDistributedNotificationCenter 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 NSDistributedNotificationCenter.

Principal Attributes

- A table of objects that want to receive notifications, the notifications they want to receive, and identifying strings they are interested in

Each task has a default distributed notification center. You typically don't create your own.

Commonly Used Methods

`defaultCenter` (page 161)
: Accesses the default notification center.

`addObserver` (page 162)
: Registers an object to receive a notification with a specified behavior when notification delivery is suspended.

`postNotification` (page 162)
: Creates and posts a notification.

Overview

An NSDistributedNotificationCenter object (or simply, distributed notification center) is a notification center that can distribute notifications asynchronously to tasks other than the one in which the notification was posted.

Each task has a default distributed notification center that you access with the `defaultCenter` (page 161) static method. There may be different types of distributed notification centers. Right now there is a single type—`LocalNotificationCenterType`. This type of distributed notification center handles notifications that can be sent between tasks on a single machine. For communication between tasks on different machines, use "Distributed Objects".

Posting a distributed notification is an expensive operation. The notification gets sent to a system-wide server that then 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, if too many notifications are being posted and the server's queue fills up, notifications can 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 `NSRunLoop.DefaultRunLoopMode`, 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.

Tasks

Constructors

`NSDistributedNotificationCenter` (page 161)
Creates an empty NSDistributedNotificationCenter.

Accessing Distributed Notification Centers

`defaultCenter` (page 161)
Returns the default distributed notification center, representing the local notification center for the machine by calling `notificationCenterForType` (page 161) with an argument of `LocalNotificationCenterType`.

`notificationCenterForType` (page 161)
Returns the distributed notification center for the specified *`type`*.

Adding and Removing Observers

`addObserver` (page 162)
Registers *`anObserver`* to receive notifications with the name *`notificationName`* and/or the identifying string *`anObject`*.

Posting Notifications

`postNotification` (page 162)
Creates a notification with the name *`notificationName`*, associates it with the string *`anObject`* and dictionary *`userInfo`*, and posts it to the notification center with delivery scheduled for *`deliverImmediately`*, as supplied by the invoker.

Suspending and Enabling Notification Delivery

`setSuspended` (page 163)
: Suspends notification delivery when set to `true` and resumes immediate notification delivery when set to `false`.

`suspended` (page 163)
: Returns `true` if the notification center is delivering notifications for this application according to their suspension behavior, `false` if it is delivering them immediately.

Constructors

NSDistributedNotificationCenter

Creates an empty NSDistributedNotificationCenter.

```
public NSDistributedNotificationCenter()
```

Discussion
This center is not the default notification center. To obtain the default center, use `defaultCenter` (page 161).

Static Methods

defaultCenter

Returns the default distributed notification center, representing the local notification center for the machine by calling `notificationCenterForType` (page 161) with an argument of `LocalNotificationCenterType`.

```
public static NSNotificationCenter defaultCenter()
```

notificationCenterForType

Returns the distributed notification center for the specified *type*.

```
public static NSDistributedNotificationCenter notificationCenterForType(String
    type)
```

Discussion
Currently only one type, `LocalNotificationCenterType`, is supported.

Instance Methods

addObserver

Registers *anObserver* to receive notifications with the name *notificationName* and/or the identifying string *anObject*.

```
public void addObserver(Object anObserver, NSSelector aSelector, String
    notificationName, String anObject, int suspensionBehavior)
```

Discussion
When a notification of name *notificationName* with the identifying string *anObject* is posted, *anObserver* receives an *aSelector* message with this notification as the argument. The method for the selector specified in *aSelector* must have one and only one argument. If *notificationName* is `null`, the notification center notifies the observer of all notifications with an identifying string matching *anObject*. If *anObject* is `null`, the notification center notifies the observer of all notifications with the name *notificationName*. The *suspensionBehavior* determines how the notification center handles notifications when notification delivery has been suspended. The possible values are described in "Constants" (page 164).

See Also
`postNotification` (page 162)

postNotification

Creates a notification with the name *notificationName*, associates it with the string *anObject* and dictionary *userInfo*, and posts it to the notification center with delivery scheduled for *deliverImmediately*, as supplied by the invoker.

```
public void postNotification(String notificationName, String anObject, NSDictionary
     userInfo, boolean deliverImmediately)
```

Discussion
This method is the preferred method for posting notifications.

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 *userInfo* dictionary. See "XML Property Lists" for details.

Posting with *deliverImmediately* set to `false` allows the normal suspension behavior of the observers to take place. If *deliverImmediately* is set to `true`, the notification is delivered immediately to all observers, regardless of their suspension behavior or suspension state.

Creates a notification with the name *notificationName*, associates it with the string *anObject* and dictionary *userInfo*, and posts it to the notification center.

```
public void postNotification(String name, String anObject, NSDictionary userInfo,
     int options)
```

Discussion
Possible values for *options* are described in the "Constants" (page 164) section. Pass in 0 for no options.

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 *userInfo* dictionary. See "XML Property Lists" for details.

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

See Also
encodeRootObject (page 53) (NSArchiver)
unarchiveObjectWithData (page 626) (NSUnarchiver)

setSuspended

Suspends notification delivery when set to `true` and resumes immediate notification delivery when set to `false`.

```
public void setSuspended(boolean suspended)
```

Discussion
Distributed notification centers enable or suspend notification delivery on a per-task basis. When a task suspends notification delivery, notifications are delivered according to the suspension behavior of the observer. When delivery is not suspended, notifications are always delivered immediately. See "Constants" (page 164) for the available types of suspension behaviors.

NSApplication automatically suspends delivery when the application is not active. Applications based on the Application Kit should let the Application Kit manage the suspension of distributed notification delivery. Foundation-only programs may have occasional need to use this method.

See Also
addObserver (page 162)
postNotification (page 162)
suspended (page 163)

suspended

Returns `true` if the notification center is delivering notifications for this application according to their suspension behavior, `false` if it is delivering them immediately.

```
public boolean suspended()
```

Discussion
Applications based on the Application Kit should let the Application Kit manage the suspension of distributed notification delivery. Foundation-only programs may have occasional need to use this method.

See Also
setSuspended (page 163)

Constants

NSDistributedNotificationCenter defines the following notification center type:

Constant	Description
`LocalNotificationCenterType`	Distributes notifications to all tasks on the sender's machine.

There are four different types of suspension behavior, each useful in different circumstances:

Constant	Description
`NotificationSuspensionBehaviorDrop`	The server does not queue any notifications with this name and object until `setSuspended` (page 163) with an argument of `false` is called.
`NotificationSuspensionBehaviorCoalesce`	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, `NotificationSuspensionBehaviorCoalesce` is the default.
`NotificationSuspensionBehaviorHold`	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.
`NotificationSuspensionBehaviorDeliverImmediately`	The server delivers notifications matching this registration irrespective of whether `setSuspended` (page 163) with an argument of `true` 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 163) with an argument of `false` 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.

NSDistributedNotificationCenter defines these constants to specify the behavior of notifications posted using `postNotification` (page 162):

Constant	Description
`NotificationDeliverImmediately`	If not set, allows the normal suspension behavior of notification observers to take place. If set, the notification is delivered immediately to all observers, regardless of their suspension behavior or suspension state.
`NotificationPostToAllSessions`	If not set, the notification is sent only to applications within the same login session as the posting process. If set, the notification is posted to all sessions.

NSEnumerator

Inherits from	Object
Implements	java.util.Enumeration
Package:	com.apple.cocoa.foundation
Companion guide	Collections Programming Topics for Cocoa

Class at a Glance

An abstract class whose instances enumerate collections of other objects, such as arrays and dictionaries.

Principal Attributes

- A set of objects to enumerate
- The next object in the enumeration

All creation methods are defined in the collection classes such as NSArray and NSDictionary. These methods contain the word "Enumerator," as in NSArray's `objectEnumerator` (page 64) method or NSDictionary's `keyEnumerator` (page 157) method.

Commonly Used Methods

`nextElement` (page 167)
Returns the next object in the collection being enumerated.

Overview

NSEnumerator is a simple abstract class whose subclasses enumerate collections of other objects. Collection objects—such as arrays, sets, and dictionaries—provide special NSEnumerator objects with which to enumerate their contents. You send `nextElement` (page 167) repeatedly to a newly created NSEnumerator object to have it return the next object in the original collection. When the collection is exhausted, `null` is returned. You can't "reset" an enumerator after it's exhausted its collection. To enumerate a collection again, you need a new enumerator.

Collection classes such as NSArray, NSSet, and NSDictionary include methods that return an enumerator appropriate to the type of collection. For instance, NSArray has two methods that return an NSEnumerator object: `objectEnumerator` (page 64) and `reverseObjectEnumerator` (page 65). NSDictionary also has two methods that return an NSEnumerator object: `keyEnumerator` (page 157) and `objectEnumerator` (page 158). These methods let you enumerate the contents of an NSDictionary by key or by value, respectively.

Note: It isn't safe to modify a mutable collection while enumerating through it.

The enumerator subclasses used by NSArray, NSDictionary, and NSSet retain the collection during enumeration. When the enumeration is exhausted, the collection is released.

Tasks

Getting the Objects

`nextElement` (page 167)
Returns the next object from the collection being enumerated.

Querying Enumerators

`getObjCEnumerator` (page 166)
Returns an integer value identifying the underlying Objective-C enumerator.

`hasMoreElements` (page 166)
Returns a Boolean value that indicates whether there are more elements in the collection that can be enumerated by the receiver.

Instance Methods

getObjCEnumerator

Returns an integer value identifying the underlying Objective-C enumerator.

```
public int getObjCEnumerator()
```

hasMoreElements

Returns a Boolean value that indicates whether there are more elements in the collection that can be enumerated by the receiver.

```
public boolean hasMoreElements()
```

nextElement

Returns the next object from the collection being enumerated.

```
public Object nextElement()
```

Discussion
When `nextElement` returns `null`, all objects have been enumerated.

NSError

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.foundation
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Error Handling Programming Guide For Cocoa

Overview

NSError encapsulates richer and more extensible error information than is possible using only an error code or error string. The core attributes of an NSError 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. In addition, NSError allows an arbitrary user info dictionary to be specified, 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 171).

In general, the presence of an error should be indicated by other means, for example by returning `false` or `null` from the method. The method can then optionally return an NSError object by-reference, in order to further describe the error.

Tasks

Constructors

`NSError` (page 170)
Creates an empty NSError.

Getting Error Properties

`code` (page 171)
Returns the receiver's error code.

`domain` (page 171)
Returns a string containing the receiver's error domain.

`userInfo` (page 173)
Returns an NSDictionary containing the user info associated with the receiver or `null` if the user info dictionary has not been set.

Getting a Localized Error Description

`localizedDescription` (page 171)
Returns a string containing the localized description of the error.

`localizedRecoveryOptions` (page 172)
Returns an array containing the localized titles of buttons appropriate for displaying in an alert panel.

`localizedRecoverySuggestion` (page 172)
Returns a string containing the localized recovery suggestion for the error.

`localizedFailureReason` (page 172)
Returns a string containing the localized explanation of the reason for the error.

Getting the Error Recovery Attempter

`recoveryAttempter` (page 173)
Returns an object that conforms to the `NSErrorRecoveryAttempting` informal protocol.

Constructors

NSError

Creates an empty NSError.

```
public NSError()
```

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

Creates an NSError object for *`domain`* with the specified error *`code`* and the dictionary of arbitrary data *`userInfo`*.

```
public NSError(String domain, int code, NSDictionary dict)
```

Discussion
The *`domain`* must not be `null`. The *`userInfo`* may be `null`.

The *`domain`* can be one of the predefined NSError domains, or an arbitrary string describing a custom domain.

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

Instance Methods

code

Returns the receiver's error code.

```
public int code()
```

Discussion
Note that errors are domain specific.

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

See Also
`localizedDescription` (page 171)
`domain` (page 171)
`userInfo` (page 173)

domain

Returns a string containing the receiver's error domain.

```
public String domain()
```

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

See Also
`code` (page 171)
`localizedDescription` (page 171)
`userInfo` (page 173)

localizedDescription

Returns a string containing the localized description of the error.

```
public String localizedDescription()
```

Discussion
By default this method will attempt to return the object in the user info dictionary for the key `LocalizedDescriptionKey`. If the user info dictionary doesn't contain a value for `LocalizedDescriptionKey`, a default string will be constructed from the domain and code.

This method can be overridden by subclasses to present customized error strings.

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

See Also
code (page 171)
domain (page 171)
userInfo (page 173)

localizedFailureReason

Returns a string containing the localized explanation of the reason for the error.

```
public String localizedFailureReason()
```

Discussion
By default this method will attempt to return the object in the user info dictionary for the key `LocalizedFailureReasonErrorKey`.

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 171)
domain (page 171)
userInfo (page 173)

localizedRecoveryOptions

Returns an array containing the localized titles of buttons appropriate for displaying in an alert panel.

```
public Array localizedRecoveryOptions()
```

Discussion
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 172). By default this method will attempt to return the object in the user info dictionary for the key `LocalizedRecoveryOptionsErrorKey`. If the user info dictionary doesn't contain a value for `LocalizedRecoveryOptionsErrorKey`, `null` is returned and 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 10.4 and later.

localizedRecoverySuggestion

Returns a string containing the localized recovery suggestion for the error.

```
public String localizedRecoverySuggestion()
```

Discussion
This string is suitable for displaying as the secondary message in an alert panel. By default this method will attempt to return the object in the user info dictionary for the key `LocalizedRecoverySuggestionErrorKey`. If the user info dictionary doesn't contain a value for `LocalizedRecoverySuggestionErrorKey`, `null` is returned.

This method can be overridden by subclasses to present customized recovery suggestion strings.

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

recoveryAttempter

Returns an object that conforms to the `NSErrorRecoveryAttempting` informal protocol.

```
public Object recoverAttempter()
```

Discussion
The recovery attempter must be an object that can correctly interpret an index into the array returned by `localizedRecoveryOptions` (page 172). By default this method will attempt to return the object for the user info dictionary for the key `RecoveryAttempterErrorKey`. If the user info dictionary doesn't contain a value for `RecoveryAttempterErrorKey`, `null` is returned.

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

See Also
`localizedRecoveryOptions` (page 172)

userInfo

Returns an NSDictionary containing the user info associated with the receiver or `null` if the user info dictionary has not been set.

```
public NSDictionary userInfo()
```

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

See Also
`code` (page 171)
`domain` (page 171)
`localizedDescription` (page 171)

Constants

The following keys may exist in the user info dictionary:

Constant	Description
`LocalizedFailure-ReasonErrorKey`	A localized string representation containing the reason for the failure that, if present, will be returned by `localizedFailureReason` (page 172). This string p;rovides a more detailed explanation of the error than the description. Available in Mac OS X 10.4 and later.
`LocalizedRecovery-SuggestionErrorKey`	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 10.4 and later.
`LocalizedRecovery-OptionsErrorKey`	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 172). Available in Mac OS X 10.4 and later.
`RecoveryAttempter-ErrorKey`	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 173). Available in Mac OS X 10.4 and later.

The following error domains are predefined:

Constant	Description
`POSIXErrorDomain`	POSIX/BSD errors
`OSStatusErrorDomain`	Mac OS 9/Carbon errors
`MachErrorDomain`	Mach errors
`NSURLErrorDomain`	URL loading system errors
`NSCocoaErrorDomain`	Application Kit and Foundation Kit errors. Available in Mac OS X v10.4 and later.

NSException

Inherits from	java.lang.RuntimeException
Package:	com.apple.cocoa.foundation
Companion guide	Exception Programming Topics for Cocoa

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.

The string constants for exceptions are listed and described in Foundation Types and Constants.

Tasks

Constructors

`NSException` (page 176)
Creates an NSException object with a `null` message string.

Querying an NSException

`getStackTrace` (page 176)
Returns the stack trace for where *`anException`* was thrown.

`name` (page 176)
Returns a String used to uniquely identify the receiver.

`toString` (page 176)
Returns the receiver's name and reason message, so that formatted strings produce a meaningful description of the exception.

`userInfo` (page 177)
Returns an Object containing application-specific data pertaining to the receiver.

Constructors

NSException

Creates an NSException object with a `null` message string.

```
public NSException()
```

Creates an NSException object with the human-readable message string *reason*.

```
public NSException(String reason)
```

Creates an NSException object named *name* with the human-readable message string *reason* and user-defined *userInfo*.

```
public NSException(String name, String reason, Object userInfo)
```

Static Methods

getStackTrace

Returns the stack trace for where *anException* was thrown.

```
public static String getStackTrace(Throwable anException)
```

Discussion
The exception's name and reason message are included in the first line. If *anException* has no name, the exception's class is used.

Instance Methods

name

Returns a String used to uniquely identify the receiver.

```
public String name()
```

toString

Returns the receiver's name and reason message, so that formatted strings produce a meaningful description of the exception.

```
public String toString()
```

Discussion
If the receiver is not named, the receiver's class name is used.

userInfo

Returns an Object containing application-specific data pertaining to the receiver.

```
public Object userInfo()
```

Discussion
Returns `null` if no application-specific data exists. As an example, if a method's return value caused the exception to be thrown, the return value might be available to the exception handler through this method.

NSExistsCommand

Inherits from	NSScriptCommand : NSObject
Package:	com.apple.cocoa.foundation
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.

Tasks

Constructors

`NSExistsCommand` (page 179)
Returns an NSExistsCommand with no data.

Constructors

NSExistsCommand

Returns an NSExistsCommand with no data.

```
public NSExistsCommand()
```

Discussion
Do not use this constructor.

Initializes an instance of NSExistsCommand with the command description supplied by *aScriptCommandDescription*.

```
public NSExistsCommand(NSScriptCommandDescription aScriptCommandDescription)
```

Discussion
Note that such an instance has no receiver specifier, arguments, or direct parameter and is not a fully functional command.

NSExpression

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.foundation
Availability	Available in Mac OS X v10.4 and later.
Companion guide	Predicate Programming Guide

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.

Tasks

Constructors

`NSExpression` (page 182)
Returns an empty NSExpression object.

Constructors and Initialization

`expressionForConstantValue` (page 182)

`expressionForEvaluatedObject` (page 183)

`expressionForFunction` (page 183)
Returns a new expression that invokes a predefined function.

`expressionForKeyPath` (page 183)

`expressionForVariable` (page 184)

Getting Information About an Expression

`arguments` (page 184)
Returns 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.

`constantValue` (page 184)
Returns the constant value for the receiver.

`expressionType` (page 184)
Returns the expression type for the receiver.

`function` (page 185)
Returns the function for the receiver.

`keyPath` (page 185)
Returns the key path for the receiver.

`operand` (page 185)
Returns the operand for the receiver—that is, the object on which the selector will be invoked.

`variable` (page 186)
Returns the variable for the receiver.

Evaluating an Expression

`expressionValueWithObject` (page 185)
Evaluates the expression using the specified object and context.

Constructors

NSExpression

Returns an empty NSExpression object.

```
public NSExpression()
```

Returns an NSExpression object initialized with the specified expression type.

```
public NSExpression(int type)
```

Discussion
The *type* parameter represents the expression type as shown in "Constants" (page 186).

Static Methods

expressionForConstantValue

```
public static NSExpression expressionForConstantValue(Object obj)
```

Discussion
Returns a new expression that represents a constant value.

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

expressionForEvaluatedObject

```
public static NSExpression expressionForEvaluatedObject()
```

Discussion
Returns a new expression that represents the object being evaluated.

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

expressionForFunction

Returns a new expression that invokes a predefined function.

```
public static NSExpression expressionForFunction(String name, NSArray parameters)
```

Discussion
The *name* parameter can be one of the following predefined functions.

Function	Parameter	Returns
avg	NSArray of NSExpressions	NSNumber
count	NSArray of NSExpressions	NSNumber
max	NSArray of NSExpressions	NSNumber
min	NSArray of NSExpressions	NSNumber
sum	NSArray of NSExpressions	NSNumber

This method throws an exception immediately if the selector is invalid; it throws an exception at runtime if the parameters are incorrect.

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

expressionForKeyPath

```
public static NSExpression expressionForKeyPath(String keyPath)
```

Discussion
Returns a new expression that invokes `valueForKeyPath` (page 263) with *keyPath*.

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

expressionForVariable

```
public static NSExpression expressionForVariable(String string)
```

Discussion
Returns a new expression that extracts a value from the variable bindings dictionary.

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

Instance Methods

arguments

Returns 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.

```
public NSArray arguments()
```

Discussion
Throws an exception if not applicable.

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

constantValue

Returns the constant value for the receiver.

```
public Object constantValue()
```

Discussion
Throws an exception if not applicable.

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

expressionType

Returns the expression type for the receiver.

```
public int expressionType()
```

Discussion
The expression type is described in "Constants" (page 186).Throws an exception if not applicable.

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

expressionValueWithObject

Evaluates the expression using the specified object and context.

```
public Object expressionValueWithObject(Object object, NSMutableDictionary context)
```

Discussion
Note that *context* is mutable—it can be used by expressions to store temporary state for one predicate evaluation.

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

function

Returns the function for the receiver.

```
public String function()
```

Discussion
Throws an exception if not applicable.

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

keyPath

Returns the key path for the receiver.

```
public String keyPath()
```

Discussion
Throws an exception if not applicable.

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

operand

Returns the operand for the receiver—that is, the object on which the selector will be invoked.

```
public NSExpression operand()
```

Discussion
The object is the result of evaluating a key path or one of the defined functions. Throws an exception if not applicable.

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

variable

Returns the variable for the receiver.

```
public String variable()
```

Discussion
Throws an exception if not applicable.

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

Constants

These constants describe the possible types of NSExpression:

Constant	Description
`ConstantValueExpressionType`	An expression that always returns the same value.
`EvaluatedObjectExpressionType`	An expression that always returns the parameter object itself.
`VariableExpressionType`	An expression that always returns whatever value is associated with the key specified by 'variable' in the bindings dictionary.
`KeyPathExpressionType`	An expression that returns something that can be used as a key path.
`FunctionExpressionType`	An expression that returns the result of evaluating a function.

NSFormatter

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.foundation
Companion guide	Data Formatting Programming Guide for Cocoa

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 (page 411) and NSGregorianDateFormatter (page 209).

Subclassing Notes

NSFormatter is similar to other abstract classes such as NSView or NSDocument in that it 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 NSGregorianDateFormatter (page 209) and NSNumberFormatter (page 411) to satisfy your requirements.

For instructions on how to create your own custom formatter, see “Creating A Custom Formatter”.

Tasks

Constructors

`NSFormatter` (page 188)
NSFormatter is an abstract class; use the constructor of one of its concrete classes instead.

Textual Representation of Cell Content

`stringForObjectValue` (page 190)
: This method is an abstract method that you must override in your subclass. The default implementation of this method throws an exception.

`attributedStringForObjectValue` (page 188)
: This method is an abstract method that you must override in your subclass.

`editingStringForObjectValue` (page 189)
: The default implementation of this method invokes `stringForObjectValue` (page 190).

Object Equivalent to Textual Representation

`objectValueForString` (page 189)
: This method is an abstract method that you must override in your subclass.

Dynamic Cell Editing

`isPartialStringValid` (page 189)
: Since this method is invoked each time the user presses a key while the cell has the keyboard focus, it lets you verify the cell text as the user types it. *`partialString`* is the text currently in the cell.

`replacementStringForString` (page 190)
: The default implementation of this method returns *`aString`*.

Constructors

NSFormatter

NSFormatter is an abstract class; use the constructor of one of its concrete classes instead.

```
public NSFormatter()
```

Instance Methods

attributedStringForObjectValue

This method is an abstract method that you must override in your subclass.

```
public abstract NSAttributedString attributedStringForObjectValue(Object anObject,
    NSDictionary attributes)
```

Discussion
When implementing a subclass, return an NSAttributedString 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 190) to get the nonattributed string. Then create an NSAttributedString with it. The default attributes for text in the cell are passed in with *`attributes`*; use this NSDictionary to reset the attributes of the string when a change in value warrants it (for example, a negative value becomes positive) If a NSAttributedString cannot be created for *`anObject`*, an `NSFormatter.FormattingException` is thrown. For information on creating attributed strings, see the NSAttributedString (page 67) class.

See Also
`editingStringForObjectValue` (page 189)

editingStringForObjectValue

The default implementation of this method invokes `stringForObjectValue` (page 190).

```
public String editingStringForObjectValue(Object 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 a String that is used for editing, following the logic recommended for implementing `stringForObjectValue`. As an example, you would implement this method if you want the dollar signs in displayed strings removed for editing.

See Also
`attributedStringForObjectValue` (page 188)

isPartialStringValid

Since this method is invoked each time the user presses a key while the cell has the keyboard focus, it lets you verify the cell text as the user types it. *`partialString`* is the text currently in the cell.

```
public boolean isPartialStringValid(String partialString)
```

Discussion
Return `true` if it is acceptable and `false` if it is not. If you return `false`, the cell displays *`partialString`* minus the last character typed.

See Also
`replacementStringForString` (page 190)

objectValueForString

This method is an abstract method that you must override in your subclass.

```
public abstract Object objectValueForString(String aString)
```

Discussion
When implementing a subclass, return an object you've created from *`aString`*. If an object cannot be created from *`aString`*, an `NSFormatter.ParsingException` is thrown.

See Also
`stringForObjectValue` (page 190)

replacementStringForString

The default implementation of this method returns *aString*.

```
public String replacementStringForString(String aString)
```

Discussion
When implementing a subclass, check whether *aString* is a valid string for the cell. If it is, return it unmodified. Otherwise, correct it and return the modified string. For example, you might convert all lowercase letters to uppercase or insert separator characters in a telephone number.

See Also
`isPartialStringValid` (page 189)

stringForObjectValue

This method is an abstract method that you must override in your subclass. The default implementation of this method throws an exception.

```
public abstract String stringForObjectValue(Object anObject)
```

Discussion
When implementing a subclass, return the String that textually represents the cell's object for display and—if `editingStringForObjectValue` (page 189) is unimplemented—for editing. First test the passed-in object to see if it's of the correct class. If it isn't, return `null`; but if it is of the right class, return a properly formatted and, if necessary, localized string. If a string cannot be created for *anObject*, an `NSFormatter.FormattingException` is thrown.

See Also
`attributedStringForObjectValue` (page 188)
`editingStringForObjectValue` (page 189)
`objectValueForString` (page 189)

NSFormatter.FormattingException

Inherits from	java.lang.Exception
Package:	com.apple.cocoa.foundation
Companion guides	Data Formatting Programming Guide for Cocoa Exception Programming Topics for Cocoa

Overview

FormattingException is a custom exception thrown by the NSFormatter methods `stringForObjectValue` (page 190) and `attributedStringForObjectValue` (page 188) when they encounter an error trying to convert the object to its string representation.

Tasks

Constructors

`FormattingException` (page 191)
Creates a new FormattingException exception with the message *reason*.

Constructors

FormattingException

Creates a new FormattingException exception with the message *reason*.

```
public NSFormatter.FormattingException(String reason)
```

NSFormatter.FormattingException

NSFormatter.ParsingException

Inherits from	java.lang.Exception
Package:	com.apple.cocoa.foundation
Companion guides	Data Formatting Programming Guide for Cocoa Exception Programming Topics for Cocoa

Overview

ParsingException is a custom exception thrown by the NSFormatter method `objectValueForString` (page 189) when it encounters a problem converting the string to the appropriate object representation.

Tasks

Constructors

`ParsingException` (page 193)
Creates a new ParsingException exception with the message *reason*.

Constructors

ParsingException

Creates a new ParsingException exception with the message *reason*.

```
public NSFormatter.ParsingException(String reason)
```

NSGetCommand

Inherits from	NSScriptCommand : NSObject
Package:	com.apple.cocoa.foundation
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.

Tasks

Constructors

`NSGetCommand` (page 195)
Returns an NSGetCommand with no data.

Constructors

NSGetCommand

Returns an NSGetCommand with no data.

```
public NSGetCommand()
```

Discussion
Do not use this constructor.

Initializes an instance of NSGetCommand with the command description supplied by *aScriptCommandDescription*.

```
public NSGetCommand(NSScriptCommandDescription aScriptCommandDescription)
```

Discussion
Note that such an instance has no receiver specifier, arguments, or direct parameter and is not a fully functional command.

NSGregorianDate

Inherits from	NSDate : NSObject
Implements	NSCoding (NSDate)
Package:	com.apple.cocoa.foundation
Companion guides	Date and Time Programming Guide for Cocoa Data Formatting Programming Guide for Cocoa

Overview

NSGregorianDate 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. NSGregorianDates are immutable objects.

An NSGregorianDate 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 NSGregorianDate 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 NSGregorianDates 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 NSGregorianDate object with a different time zone to see how the particular date is represented in that time zone.

To retrieve conventional elements of an NSGregorianDate object, use the `...Of...` methods. For example, `dayOfWeek` (page 201) returns a number that indicates the day of the week (0 is Sunday). The `monthOfYear` (page 204) method returns a number from 1 through 12 that indicates the month.

To format a date as a string or to parse a date from a string, use an NSGregorianDateFormatter.

Tasks

Constructors

`NSGregorianDate` (page 199)

Retrieving Date Elements

`dayOfCommonEra` (page 200)

`dayOfMonth` (page 201)
Returns a number that indicates the day of the month (1 through 31) of the receiver.

`dayOfWeek` (page 201)
Returns a number that indicates the day of the week (0 through 6) of the receiver; 0 indicates Sunday.

`dayOfYear` (page 202)
Returns a number that indicates the day of the year (1 through 366) of the receiver.

`hourOfDay` (page 203)
Returns the hour value (0 through 23) of the receiver.

`microsecondOfSecond` (page 204)
Returns the microseconds value (0 through 999,999) of the receiver.

`minuteOfHour` (page 204)
Returns the minutes value (0 through 59) of the receiver.

`monthOfYear` (page 204)
Returns a number that indicates the month of the year (1 through 12) of the receiver.

`secondOfMinute` (page 205)
Returns the seconds value (0 through 59) of the receiver.

`yearOfCommonEra` (page 205)
Returns a number that indicates 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.).

Adjusting a Date

`dateByAddingGregorianUnits` (page 200)

Computing Date Intervals

`gregorianUnitsSinceDate` (page 202)
Computes the calendrical time difference between the receiver and *`date`* and returns it in *`years`*, *`months`*, *`days`*, *`hours`*, *`minutes`*, and *`seconds`*.

Comparing Dates

`equals` (page 202)
Returns `true` if *`anObject`* is an instance of NSGregorianDate and satisfies `isEqualToGregorianDate` (page 203).

`hashCode` (page 203)
Returns a hash value for the receiver.

`isEqualToGregorianDate` (page 203)
Returns `true` if the receiver and *`aGregorianDate`* represent the same date and have the same time zone set.

Representing Dates as Strings

`toString` (page 205)
Returns a string representation of the receiver.

Getting the Time Zone

`timeZone` (page 205)
Returns the time zone object associated with the receiver.

Constructors

NSGregorianDate

```
public NSGregorianDate()
```

Discussion
Creates a new Gregorian date initialized to the current date and time.

```
public NSGregorianDate(double seconds)
```

Discussion
Creates a new Gregorian date initialized to the absolute reference date (the first instant of 1 January 2001, GMT) plus *`seconds`*, which may be positive or negative. This constructor sets the date's time zone to the default time zone.

```
public NSGregorianDate(double seconds, NSDate aDate)
```

Discussion
Creates a new Gregorian date initialized to *`aDate`* plus *`seconds`*, which may be positive or negative. This constructor sets the date's time zone to the default time zone.

```
public NSGregorianDate(double seconds, NSTimeZone aTimeZone)
```

Discussion
Creates a new Gregorian date initialized to the absolute reference date (the first instant of 1 January 2001, GMT) plus *`seconds`*, which may be positive or negative. This constructor sets the date's time zone to *`aTimeZone`*.

```
public NSGregorianDate(int year, int month, int day, int hour, int minute, int
    second, NSTimeZone aTimeZone)
```

Discussion
Creates a new Gregorian date with the specified values for *year*, *month*, *day*, *hour*, *minute*, *second*, and time zone, *aTimeZone*. The year value must include the century (for example, 1995 instead of 95). The other values are the standard ones: 1 through 12 for months, 1 through 31 for days, 0 through 23 for hours, and 0 through 59 for both minutes and seconds.

On days when daylight savings "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 `dateByAddingGregorianUnits` (page 200) to add an hour.

The following code fragment shows a Gregorian date created for 4 July 1994, 9 PM, eastern standard time:

```
NSGregorianDate fireworks = new NSGregorianDate(1994, 7, 4, 21,  0, 0,
        new NSTimeZone("EST", true));
```

Instance Methods

dateByAddingGregorianUnits

```
public NSGregorianDate dateByAddingGregorianUnits(int year, int month, int day,
    int hour, int minute, int second)
```

Discussion
Returns a Gregorian date that is updated with the *year*, *month*, *day*, *hour*, *minute*, and *second* offsets specified as arguments. The offsets can be positive (future) or negative (past).

This method preserves "clock time" across changes in daylight savings time zones and leap years. For example, adding one month to a Gregorian date with a time of 12 noon correctly maintains time at 12 noon. One thing to be aware of is if you add one day to 2:30 AM on the day before daylight savings "springs ahead," it will actually result in 1:30 AM on the next day (which is one day, or 24 hours, later).

Note that the arguments 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.

The following code fragment shows a Gregorian date created with a date a week later than an existing Gregorian date:

```
NSCalendarDate now = new NSGregorianDate();
NSCalendarDate nextWeek =
    now.dateByAddingGregorianUnits(0, 0, 7, 0, 0, 0);
```

See Also
`gregorianUnitsSinceDate` (page 202)

dayOfCommonEra

```
public int dayOfCommonEra()
```

Discussion
Returns the number of days since the beginning of the Common Era. The base year of the Common Era is 1 C.E. (which is the same as 1 A.D.).

See Also
dayOfMonth (page 201)
dayOfWeek (page 201)
dayOfYear (page 202)
hourOfDay (page 203)
microsecondOfSecond (page 204)
minuteOfHour (page 204)
monthOfYear (page 204)
secondOfMinute (page 205)
yearOfCommonEra (page 205)

dayOfMonth

Returns a number that indicates the day of the month (1 through 31) of the receiver.

```
public int dayOfMonth()
```

See Also
dayOfCommonEra (page 200)
dayOfWeek (page 201)
dayOfYear (page 202)
hourOfDay (page 203)
microsecondOfSecond (page 204)
minuteOfHour (page 204)
monthOfYear (page 204)
secondOfMinute (page 205)
yearOfCommonEra (page 205)

dayOfWeek

Returns a number that indicates the day of the week (0 through 6) of the receiver; 0 indicates Sunday.

```
public int dayOfWeek()
```

See Also
dayOfCommonEra (page 200)
dayOfMonth (page 201)
dayOfYear (page 202)
hourOfDay (page 203)
microsecondOfSecond (page 204)
minuteOfHour (page 204)
monthOfYear (page 204)
secondOfMinute (page 205)
yearOfCommonEra (page 205)

dayOfYear

Returns a number that indicates the day of the year (1 through 366) of the receiver.

```
public int dayOfYear()
```

See Also
dayOfCommonEra (page 200)
dayOfMonth (page 201)
dayOfWeek (page 201)
hourOfDay (page 203)
microsecondOfSecond (page 204)
minuteOfHour (page 204)
monthOfYear (page 204)
secondOfMinute (page 205)
yearOfCommonEra (page 205)

equals

Returns `true` if *anObject* is an instance of NSGregorianDate and satisfies `isEqualToGregorianDate` (page 203).

```
public boolean equals(Object anObject)
```

Discussion
Returns `false` otherwise.

gregorianUnitsSinceDate

Computes the calendrical time difference between the receiver and *date* and returns it in *years*, *months*, *days*, *hours*, *minutes*, and *seconds*.

```
public void gregorianUnitsSinceDate(NSGregorianDate date, NSGregorianDate.IntRef
    years, NSGregorianDate.IntRef months, NSGregorianDate.IntRef days,
    NSGregorianDate.IntRef hours, NSGregorianDate.IntRef minutes,
    NSGregorianDate.IntRef seconds)
```

Discussion
NSGregorianDate.IntRef is a local class that contains a single element: the integer `value`.

You can choose any representation you wish for the time difference by passing `null` for the arguments you want to ignore. For example, the following code fragment computes the difference in months, days, and years between two dates:

```
NSGregorianDate momsBDay =
    new NSGregorianDate(1936, 1, 8, 7, 30, 0, new NSTimeZone("EST",  true));
NSGregorianDate dateOfBirth =
    new NSGregorianDate(1965, 12, 7, 17, 25, 0, new NSTimeZone("EST",  true));

NSGregorianDate.IntRef years = new NSGregorianDate.IntRef();
NSGregorianDate.IntRef months = new NSGregorianDate.IntRef();
NSGregorianDate.IntRef days = new NSGregorianDate.IntRef();
```

```
dateOfBirth.gregorianUnitsSinceDate(momsBDay, years, months, days,
    null, null, null)
```

This message returns 29 years, 10 months, and 29 days. If you want to express the years in terms of months, you pass `null` for the `years` argument:

```
dateOfBirth.gregorianUnitsSinceDate(momsBDay, null, months, days,
    null, null, null);
```

This message returns 358 months and 29 days.

See Also
`dateByAddingGregorianUnits` (page 200)

hashCode

Returns a hash value for the receiver.

```
public int hashCode()
```

Discussion
The hash value is the integer value of the time interval returned from `timeIntervalSinceReferenceDate` (page 144) (NSDate).

hourOfDay

Returns the hour value (0 through 23) of the receiver.

```
public int hourOfDay()
```

Discussion
On daylight savings "fall back" days, a value of 1 is returned for two consecutive hours, but with a different time zone (the first in daylight savings time and the second in standard time).

See Also
`dayOfCommonEra` (page 200)
`dayOfMonth` (page 201)
`dayOfWeek` (page 201)
`dayOfYear` (page 202)
`microsecondOfSecond` (page 204)
`minuteOfHour` (page 204)
`monthOfYear` (page 204)
`secondOfMinute` (page 205)
`yearOfCommonEra` (page 205)

isEqualToGregorianDate

Returns `true` if the receiver and *`aGregorianDate`* represent the same date and have the same time zone set.

```
public boolean isEqualToGregorianDate(NSGregorianDate aGregorianDate)
```

Discussion
Returns `false` otherwise.

microsecondOfSecond

Returns the microseconds value (0 through 999,999) of the receiver.

```
public int microsecondOfSecond()
```

See Also
`dayOfCommonEra` (page 200)
`dayOfMonth` (page 201)
`dayOfWeek` (page 201)
`dayOfYear` (page 202)
`hourOfDay` (page 203)
`minuteOfHour` (page 204)
`monthOfYear` (page 204)
`secondOfMinute` (page 205)
`yearOfCommonEra` (page 205)

minuteOfHour

Returns the minutes value (0 through 59) of the receiver.

```
public int minuteOfHour()
```

See Also
`dayOfCommonEra` (page 200)
`dayOfMonth` (page 201)
`dayOfWeek` (page 201)
`dayOfYear` (page 202)
`hourOfDay` (page 203)
`microsecondOfSecond` (page 204)
`monthOfYear` (page 204)
`secondOfMinute` (page 205)
`yearOfCommonEra` (page 205)

monthOfYear

Returns a number that indicates the month of the year (1 through 12) of the receiver.

```
public int monthOfYear()
```

See Also
`dayOfCommonEra` (page 200)
`dayOfMonth` (page 201)

dayOfWeek (page 201)
dayOfYear (page 202)
hourOfDay (page 203)
microsecondOfSecond (page 204)
minuteOfHour (page 204)
secondOfMinute (page 205)
yearOfCommonEra (page 205)

secondOfMinute

Returns the seconds value (0 through 59) of the receiver.

```
public int secondOfMinute()
```

See Also
dayOfCommonEra (page 200)
dayOfMonth (page 201)
dayOfWeek (page 201)
dayOfYear (page 202)
hourOfDay (page 203)
microsecondOfSecond (page 204)
minuteOfHour (page 204)
monthOfYear (page 204)
yearOfCommonEra (page 205)

timeZone

Returns the time zone object associated with the receiver.

```
NSTimeZone timeZone()
```

Discussion
You can explicitly set the time zone to an NSTimeZone object using a constructor that takes an NSTimeZone object as an argument. If you do not specify a time zone for an object at initialization time, NSGregorianDate uses the default time zone for the locale.

toString

Returns a string representation of the receiver.

```
String toString()
```

yearOfCommonEra

Returns a number that indicates 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.).

```
public int yearOfCommonEra()
```

See Also

dayOfCommonEra (page 200)
dayOfMonth (page 201)
dayOfWeek (page 201)
dayOfYear (page 202)
hourOfDay (page 203)
microsecondOfSecond (page 204)
minuteOfHour (page 204)
monthOfYear (page 204)
secondOfMinute (page 205)

NSGregorianDate.IntRef

Inherits from	Object
Package:	com.apple.cocoa.foundation
Companion guide	Date and Time Programming Guide for Cocoa

Overview

The NSGregorianDate.IntRef class is used by the NSGregorianDate method `gregorianUnitsSinceDate` (page 202) to return the integer values for the number of years, months, and so on separating two dates. The class contains a single element: the integer `value`.

Tasks

Constructors

`IntRef` (page 207)
Creates a new NSGregorianDate.IntRef object.

Constructors

IntRef

Creates a new NSGregorianDate.IntRef object.

```
public NSGregorianDate.IntRef()
```

NSGregorianDateFormatter

Inherits from	NSFormatter : NSObject
Implements	NSCoding (NSFormatter)
Package:	com.apple.cocoa.foundation
Companion guide	Data Formatting Programming Guide for Cocoa

Overview

Instances of NSGregorianDateFormatter format the textual representation of cells that contain NSDates (including NSGregorianDates) and convert textual representations of dates and times into NSDates. You can express the representation of dates and times very flexibly: "Thu 22 Dec 1994" is just as acceptable as "12/22/94." With natural-language processing for dates enabled, users can also express dates colloquially, such as "today," "day after tomorrow," and "a month from today."

With Mac OS X version 10.4 and later, NSGregorianDateFormatter has two modes of operation (or behaviors). By default, instances of NSGregorianDateFormatter 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 for a full description of the old and new behaviors.

Tasks

Constructors

`NSGregorianDateFormatter` (page 210)
Creates an empty NSGregorianDateFormatter.

Getting Behavior

`allowsNaturalLanguage` (page 211)
Returns `true` if the receiver attempts to process dates entered as a vernacular string ("today," "day before yesterday," and so on).

Getting and Setting Attributes

`dateFormat` (page 211)
Returns the date format string used by the receiver.

String Manipulation

`attributedStringForObjectValue` (page 211)
Returns an NSAttributedString if the string for display should have some attributes.

`isPartialStringValid` (page 211)
Since this method is invoked each time the user presses a key while the cell has the keyboard focus, it lets you verify the cell text as the user types it. *`partialString`* is the text currently in the cell.

`objectValueForString` (page 212)
Returns an object you've created from *`aString`*.

`replacementStringForString` (page 212)
Checks whether *`aString`* is a valid string for the cell.

`stringForObjectValue` (page 212)
Returns the string that textually represents the cell's object for display and for editing.

Constructors

NSGregorianDateFormatter

Creates an empty NSGregorianDateFormatter.

```
public NSGregorianDateFormatter()
```

Discussion
The formatter processes dates entered as expressions in the vernacular (for example, "tomorrow"); NSGregorianDateFormatter attempts natural-language processing only after it fails to interpret an entered string according to format.

Creates an NSGregorianDateFormatter instance that uses the date format in its conversions.

```
public NSGregorianDateFormatter(String format, boolean naturalLanguageFlag)
```

Discussion
See "The Calendar Format" for a list of conversion specifiers permitted in date format strings. Set *`naturalLanguageFlag`* to `true` if you want the NSGregorianDateFormatter to process dates entered as expressions in the vernacular (for example, "tomorrow"); NSGregorianDateFormatter attempts natural-language processing only after it fails to interpret an entered string according to format.

Instance Methods

allowsNaturalLanguage

Returns `true` if the receiver attempts to process dates entered as a vernacular string ("today," "day before yesterday," and so on).

```
public boolean allowsNaturalLanguage()
```

Discussion
Returns `false` if the receiver does not do any natural-language processing of these date expressions.

attributedStringForObjectValue

Returns an NSAttributedString if the string for display should have some attributes.

```
public NSAttributedString attributedStringForObjectValue(Object anObject,
    NSDictionary attributes)
```

Discussion
For instance, you might want past dates to appear in red text. Invoke your implementation of `stringForObjectValue` (page 212) to get the nonattributed string. Then create an NSAttributedString with it. The default attributes for text in the cell are passed in with *`attributes`*; use this NSDictionary to reset the attributes of the string when a change in value warrants it (for example, a negative value becomes positive). If an NSAttributedString cannot be created for *`anObject`*, an `NSFormatter.FormattingException` is thrown. For information on creating attributed strings, see the NSAttributedString (page 67) class.

dateFormat

Returns the date format string used by the receiver.

```
public String dateFormat()
```

Discussion
See "The Calendar Format" for a list of the conversion specifiers permitted in date format strings.

isPartialStringValid

Since this method is invoked each time the user presses a key while the cell has the keyboard focus, it lets you verify the cell text as the user types it. *`partialString`* is the text currently in the cell.

```
public boolean isPartialStringValid(String partialString)
```

Discussion
Return `true` if *`partialString`* is acceptable and `false` if it is not. If you return `false`, the cell displays *`partialString`* minus the last character typed.

See Also
`replacementStringForString` (page 212)

objectValueForString

Returns an object you've created from *aString*.

```
public Object objectValueForString(String aString)
```

Discussion
If an object cannot be created from *aString*, an `NSFormatter.ParsingException` is thrown.

See Also
`stringForObjectValue` (page 212)

replacementStringForString

Checks whether *aString* is a valid string for the cell.

```
public String replacementStringForString(String aString)
```

Discussion
If it is, returns it unmodified. Otherwise, corrects it and returns the modified string. For example, you might convert all lowercase letters to uppercase or insert different separator characters in a date.

See Also
`isPartialStringValid` (page 211)

stringForObjectValue

Returns the string that textually represents the cell's object for display and for editing.

```
public String stringForObjectValue(Object anObject)
```

Discussion
First tests the passed-in object to see if it's of the correct class. If it isn't, returns `null`; if it is of the right class, returns a properly formatted and, if necessary, localized string. If a string cannot be created for *anObject*, an `NSFormatter.FormattingException` is thrown.

See Also
`attributedStringForObjectValue` (page 211)
`objectValueForString` (page 212)

Constants

The following constants specify predefined date and time format styles.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.

Date and Time Format Styles

NSDateFormatterNoStyle
: Specifies no style. Equal to `kCFDateFormatterNoStyle`. Available in Mac OS X version 10.4 and later.

NSDateFormatterShortStyle
: Specifies a short style, typically numeric only, such as "11/23/37" or "3:30pm". Equal to `kCFDateFormatterShortStyle`. Available in Mac OS X version 10.4 and later.

NSDateFormatterMediumStyle
: Specifies a medium style, typically with abbreviated text, such as "Nov 23, 1937". Equal to `kCFDateFormatterMediumStyle`. Available in Mac OS X version 10.4 and later.

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 version 10.4 and later.

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 version 10.4 and later.

Date Formatter Behavior

NSDateFormatterBehaviorDefault
: Specifies default formatting behavior.

NSDateFormatterBehavior10_0
: Specifies formatting behavior equivalent to that in Mac OS X 10.0.

NSDateFormatterBehavior10_4
: Specifies formatting behavior equivalent for Mac OS X 10.4.

NSHFSFileTypes

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Low-Level File Management Programming Topics

Overview

NSHFSFileTypes supports an environment in which the type of a file may be indicated by either a filename extension or an HFS file type.

Tasks

Constructors

`NSHFSFileTypes` (page 215)
Creates an NSHFSFileTypes object.

Working with HFS File Types

`fileTypeForHFSTypeCode` (page 216)
Returns a string that encodes *typeCode*.

`hfsTypeCodeFromFileType` (page 216)
Given a string of the type encoded by `fileTypeForHFSTypeCode` (page 216), returns the corresponding HFS file type code.

`hfsTypeOfFile` (page 216)
Returns a string encoding the file type of *filePath*, or `null` if unsuccessful.

Constructors

NSHFSFileTypes

Creates an NSHFSFileTypes object.

```
public NSHFSFileTypes()
```

Discussion
All of its methods are static, so there is no need to create instances.

Static Methods

fileTypeForHFSTypeCode

Returns a string that encodes *typeCode*.

```
public static String fileTypeForHFSTypeCode(int typeCode)
```

hfsTypeCodeFromFileType

Given a string of the type encoded by `fileTypeForHFSTypeCode` (page 216), returns the corresponding HFS file type code.

```
public static int hfsTypeCodeFromFileType(String fileType)
```

Discussion
If this cannot be done, 0 is returned.

hfsTypeOfFile

Returns a string encoding the file type of *filePath*, or `null` if unsuccessful.

```
public static String hfsTypeOfFile(String filePath)
```

NSIndexSet

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.foundation
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Collections Programming Topics for Cocoa

Overview

NSIndexSet manages an immutable collection of unique unsigned integers, also known as indexes because of the way they are used. You use NSIndexSet 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 NSIndexSet to store an arbitrary collection of integer values. To illustrate how this works, if you created a new NSIndexSet 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 NSIndexSet is more efficient than storing a collection of individual integers. Internally, indexes are represented 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.

NSIndexSet is not intended to be subclassed.

The mutable subclass of NSIndexSet is NSMutableIndexSet (page 329).

Tasks

Constructors

`NSIndexSet` (page 218)
Creates and returns an NSIndexSet containing the indexes specified by `NSRange.ZeroRange`.

Testing an Index Set

`isEqualToIndexSet` (page 222)
Returns `true` if the receiver contains the same indexes as *`indexSet`*.

`containsIndex` (page 219)
Returns `true` if the receiver contains the index represented by *`value`*.

`containsIndexes` (page 219)
Returns `true` if the receiver contains all of the indexes present in *`indexSet`*.

`containsIndexesInRange` (page 220)
Returns `true` if the receiver contains all the indexes in the range specified by *`range`*.

`intersectsIndexesInRange` (page 222)
Returns `true` if the receiver contains any indexes in the range specified by *`range`*.

Getting Information About an Index Set

`count` (page 220)
Returns the number of indexes in the receiver or 0 if it is empty.

Accessing Indexes

`firstIndex` (page 220)
Returns the first index in the index set or `NotFound` if the index set is empty.

`lastIndex` (page 222)
Returns the last index in the index set or `NotFound` if the index set is empty.

`indexGreaterThanIndex` (page 220)
Returns the next closest index that is greater than *`value`* or `NotFound` if *`value`* is equal to or beyond the last index in the set.

`indexLessThanIndex` (page 221)
Returns the next closest index that is less than *`value`* or `NotFound` if *`value`* is equal to or before the first index in the set.

`indexGreaterThanOrEqualToIndex` (page 221)
Returns the next closest index that is greater than or equal to *`value`* or `NotFound` if *`value`* is beyond the last index in the set.

`indexLessThanOrEqualToIndex` (page 221)
Returns the next closest index that is less than or equal to *`value`* or `NotFound` if *`value`* is before the first index in the set.

Constructors

NSIndexSet

Creates and returns an NSIndexSet containing the indexes specified by `NSRange.ZeroRange`.

```
public NSIndexSet()
```

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

Creates and returns an NSIndexSet containing a single index, *value*.

```
public NSIndexSet(int value)
```

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

Creates and returns an NSIndexSet containing the indexes specified by *range*.

```
public NSIndexSet(NSRange range)
```

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

Creates and returns an NSIndexSet containing the indexes in *indexSet*.

```
public NSIndexSet(NSIndexSet indexSet)
```

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

Instance Methods

containsIndex

Returns `true` if the receiver contains the index represented by *value*.

```
public boolean containsIndex(int value)
```

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

See Also
`containsIndexes` (page 219)
`containsIndexesInRange` (page 220)

containsIndexes

Returns `true` if the receiver contains all of the indexes present in *indexSet*.

```
public boolean containsIndexes(NSIndexSet indexSet)
```

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

See Also
containsIndex (page 219)
containsIndexesInRange (page 220)

containsIndexesInRange

Returns `true` if the receiver contains all the indexes in the range specified by *range*.

```
public boolean containsIndexesInRange(NSRange range)
```

Discussion
For example, if an index set contains indexes 20 through 30, this method would return `true` for the range (20, 8) and `false` for the range (20, 14).

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

See Also
containsIndex (page 219)
containsIndexes (page 219)
intersectsIndexesInRange (page 222)

count

Returns the number of indexes in the receiver or 0 if it is empty.

```
public int count()
```

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

firstIndex

Returns the first index in the index set or `NotFound` if the index set is empty.

```
public int firstIndex()
```

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

See Also
lastIndex (page 222)

indexGreaterThanIndex

Returns the next closest index that is greater than *value* or `NotFound` if *value* is equal to or beyond the last index in the set.

```
public int indexGreaterThanIndex(int value)
```

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

See Also
indexLessThanIndex (page 221)
indexGreaterThanOrEqualToIndex (page 221)
indexLessThanOrEqualToIndex (page 221)

indexGreaterThanOrEqualToIndex

Returns the next closest index that is greater than or equal to *value* or NotFound if *value* is beyond the last index in the set.

```
public int indexGreaterThanOrEqualToIndex(int value)
```

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

See Also
indexGreaterThanIndex (page 220)
indexLessThanIndex (page 221)
indexLessThanOrEqualToIndex (page 221)

indexLessThanIndex

Returns the next closest index that is less than *value* or NotFound if *value* is equal to or before the first index in the set.

```
public int indexLessThanIndex(int value)
```

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

See Also
indexGreaterThanIndex (page 220)
indexGreaterThanOrEqualToIndex (page 221)
indexLessThanOrEqualToIndex (page 221)

indexLessThanOrEqualToIndex

Returns the next closest index that is less than or equal to *value* or NotFound if *value* is before the first index in the set.

```
public int indexLessThanOrEqualToIndex(int value)
```

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

See Also
indexGreaterThanIndex (page 220)

indexLessThanIndex (page 221)
indexGreaterThanOrEqualToIndex (page 221)

intersectsIndexesInRange

Returns `true` if the receiver contains any indexes in the range specified by *range*.

```
public boolean intersectsIndexesInRange(NSRangerange
```

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

See Also
containsIndexesInRange (page 220)

isEqualToIndexSet

Returns `true` if the receiver contains the same indexes as *indexSet*.

```
public boolean isEqualToIndexSet(NSIndexSet indexSet)
```

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

lastIndex

Returns the last index in the index set or `NotFound` if the index set is empty.

```
public int lastIndex()
```

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

See Also
firstIndex (page 220)

Constants

NSIndexSet provides the following constant as a convenience; you can use it to compare to values returned by some NSIndexSet methods:

Constant	Description
`NotFound`	Returned when an object is not found in an NSIndexSet.

NSIndexSpecifier

Inherits from	NSScriptObjectSpecifier : NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Cocoa Scripting Guide

Overview

Specifies an object in a collection (or container) by index number. The script terms `first` and `front` specify the object with index 0, while "last" specifies the object with index of count minus 1. A negative index indicates a location by counting backward from the last object in the collection.

You don't normally subclass NSIndexSpecifier.

Tasks

Constructors

`NSIndexSpecifier` (page 223)
Returns an NSIndexSpecifier with no data.

Accessing Index Information

`index` (page 224)
Returns the index number encapsulated with the receiver for the specified object in the container.

`setIndex` (page 224)
Sets *index* as the index number encapsulated by the receiver for the specified object in the container.

Constructors

NSIndexSpecifier

Returns an NSIndexSpecifier with no data.

```
public NSIndexSpecifier()
```

Discussion
Do not use this constructor.

Returns an NSIndexSpecifier 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.

```
public NSIndexSpecifier(NSScriptClassDescription classDescription,
    NSScriptObjectSpecifier specifier, String key)
```

Discussion
The receiver's child specifier reference is set to `null`. You use the `setIndex` (page 224) method to set the zero-based index value for the specifier.

Returns an NSIndexSpecifier initialized with container specifier *specifier* and key *key*.

```
public NSIndexSpecifier(NSScriptObjectSpecifier specifier, String key)
```

Discussion
The class description of container is set automatically.

Returns an NSIndexSpecifier initialized with an index value of *index* for container specifier *specifier*, key *key*, and the class description of the object specifier *classDescription*, derived from the value of the specifier's key.

```
public NSIndexSpecifier(NSScriptClassDescription classDescription,
    NSScriptObjectSpecifier specifier, String key, int index)
```

Instance Methods

index

Returns the index number encapsulated with the receiver for the specified object in the container.

```
public int index()
```

setIndex

Sets *index* as the index number encapsulated by the receiver for the specified object in the container.

```
public void setIndex(int index)
```

NSKeyedArchiver

Inherits from	NSCoder : NSObject
Package:	com.apple.cocoa.foundation
Availability	Available in Mac OS X v10.2 and later.
Companion guide	Archives and Serializations Programming Guide for Cocoa

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 (NSMutableData) that you provide.

Tasks

Constructors

`NSKeyedArchiver` (page 228)
Creates an empty NSKeyedArchiver.

Archiving Data

`archivedDataWithRootObject` (page 229)
: Returns a data object containing the encoded form of the object graph whose root object is *rootObject*.

`archiveRootObjectToFile` (page 229)
: Archives *rootObject* by encoding it into a data object and atomically writes the resulting data object to the file *path*.

`finishEncoding` (page 236)
: Tells the receiver that you have finished encoding objects, allowing it to construct the final data stream.

`outputFormat` (page 236)
: Returns the format in which the receiver encodes its data.

`setOutputFormat` (page 237)
: Sets the format in which the receiver encodes its data.

Encoding Data

`encodeBoolForKey` (page 230)
: Encodes *boolv* and associates it with the string *key*.

`encodeByte` (page 231)
: Encodes *bytev*.

`encodeByteForKey` (page 231)
: Encodes *bytev* and associates it with the string *key*.

`encodeChar` (page 231)
: Encodes *charv*.

`encodeCharForKey` (page 231)
: Encodes *charv* and associates it with the string *key*.

`encodeConditionalObject` (page 232)
: Encodes a reference to *objv* only if *objv* has been unconditionally encoded elsewhere in the archive.

`encodeConditionalObjectForKey` (page 232)
: Encodes a reference to *objv* and associates it with the string *key* only if *objv* has been unconditionally encoded elsewhere in the archive with `encodeObjectForKey` (page 235).

`encodeDataObject` (page 232)
: Encodes *datav*.

`encodeDouble` (page 232)
: Encodes *realv*.

`encodeDoubleForKey` (page 233)
: Encodes *realv* and associates it with the string *key*.

`encodeFloat` (page 233)
: Encodes *realv*.

`encodeFloatForKey` (page 233)
: Encodes *realv* and associates it with the string *key*.

`encodeInt` (page 233)
Encodes *intv*.

`encodeIntForKey` (page 234)
Encodes *intv* and associates it with the string *key*.

`encodeLong` (page 234)
Encodes *longv*.

`encodeLongForKey` (page 234)
Encodes *longv* and associates it with the string *key*.

`encodeObject` (page 234)
Encodes *objv*.

`encodeObjectForKey` (page 235)
Encodes *objv* and associates it with the string *key*.

`encodePointForKey` (page 235)
Encodes *pointv* and associates it with the string *key*.

`encodeRectForKey` (page 235)
Encodes *rectv* and associates it with the string *key*.

`encodeShort` (page 235)
Encodes *shortv*.

`encodeShortForKey` (page 236)
Encodes *shortv* and associates it with the string *key*.

`encodeSizeForKey` (page 236)
Encodes *sizev* and associates it with the string *key*.

Managing Delegates

`delegate` (page 230)
Returns the receiver's delegate.

`setDelegate` (page 237)
Sets the receiver's delegate.

Managing Classes and Class Names

`setGlobalClassNameForClass` (page 229)
Adds a class translation mapping to NSKeyedArchiver whereby instances of *cls* are encoded with the class name *codedName* instead of their real class names.

`globalClassNameForClass` (page 229)
Returns the class name with which NSKeyedArchiver encodes instances of *cls*.

`setClassNameForClass` (page 237)
Adds a class translation mapping to the receiver whereby instances of *cls* are encoded with the class name *codedName* instead of their real class names.

`classNameForClass` (page 230)
Returns the class name with which the receiver encodes instances of *cls*.

Querying an Archiver

`versionForClassName` (page 238)
Returns the current class version number for the class named *`className`*.

Encoding objects

`archiverDidEncodeObject` (page 238) *delegate method*
Informs the delegate that *`object`* has been encoded.

`archiverWillEncodeObject` (page 238) *delegate method*
Informs the delegate that *`object`* is about to be encoded.

`archiverWillReplaceObject` (page 239) *delegate method*
Informs the delegate that *`newObject`* is being substituted for *`object`*.

Finishing encoding

`archiverDidFinish` (page 238) *delegate method*
Notifies the delegate that encoding has finished.

`archiverWillFinish` (page 239) *delegate method*
Notifies the delegate that encoding is about to finish.

Constructors

NSKeyedArchiver

Creates an empty NSKeyedArchiver.

```
public NSKeyedArchiver()
```

Discussion
Use the other constructor or the static methods `archivedDataWithRootObject` (page 229) or `archiveRootObjectToFile` (page 229), instead.

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

Creates an NSKeyedArchiver with the *`data`* object as its archive and prepares the NSKeyedArchiver for a subsequent encode operation.

```
public NSKeyedArchiver(NSMutableData data)
```

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

Static Methods

archivedDataWithRootObject

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

```
public static NSData archivedDataWithRootObject(Object rootObject)
```

Discussion
The format of the archive is `NSPropertyList.PropertyListBinaryFormat`.

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

archiveRootObjectToFile

Archives *rootObject* by encoding it into a data object and atomically writes the resulting data object to the file *path*.

```
public static boolean archiveRootObjectToFile(Object rootObject, String path)
```

Discussion
Returns `true` upon success. The format of the archive is `NSPropertyList.PropertyListBinaryFormat`.

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

globalClassNameForClass

Returns the class name with which NSKeyedArchiver encodes instances of *cls*.

```
public static String globalClassNameForClass(Class cls)
```

Discussion
Returns `null` if NSKeyedArchiver does not have a translation mapping for *cls*.

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

See Also
`setGlobalClassNameForClass` (page 229)
`classNameForClass` (page 230)

setGlobalClassNameForClass

Adds a class translation mapping to NSKeyedArchiver whereby instances of *cls* are encoded with the class name *codedName* instead of their real class names.

```
public static void setGlobalClassNameForClass(String codedName, Class cls)
```

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
`globalClassNameForClass` (page 229)
`setClassNameForClass` (page 237)

Instance Methods

classNameForClass

Returns the class name with which the receiver encodes instances of *`cls`*.

```
public String classNameForClass(Class cls)
```

Discussion
Returns `null` 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
`setClassNameForClass` (page 237)
`globalClassNameForClass` (page 229)

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

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

See Also
`setDelegate` (page 237)

encodeBoolForKey

Encodes *`boolv`* and associates it with the string *`key`*.

```
public void encodeBoolForKey(boolean boolv, String key)
```

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

See Also
`decodeBoolForKey` (page 246) (NSKeyedUnarchiver)

encodeByte

Encodes *bytev*.

```
public void encodeByte(byte bytev)
```

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

See Also
`decodeByte` (page 247) (NSKeyedUnarchiver)

encodeByteForKey

Encodes *bytev* and associates it with the string *key*.

```
public void encodeByteForKey(byte bytev, String key)
```

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

See Also
`decodeByteForKey` (page 247) (NSKeyedUnarchiver)

encodeChar

Encodes *charv*.

```
public void encodeChar(char charv)
```

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

See Also
`decodeChar` (page 247) (NSKeyedUnarchiver)

encodeCharForKey

Encodes *charv* and associates it with the string *key*.

```
public void encodeCharForKey(char charv, String key)
```

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

See Also
decodeCharForKey (page 247) (NSKeyedUnarchiver)

encodeConditionalObject

Encodes a reference to *objv* only if *objv* has been unconditionally encoded elsewhere in the archive.

```
public void encodeConditionalObject(Object objv)
```

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

See Also
encodeConditionalObjectForKey (page 232)
encodeObject (page 234)
encodeObjectForKey (page 235)

encodeConditionalObjectForKey

Encodes a reference to *objv* and associates it with the string *key* only if *objv* has been unconditionally encoded elsewhere in the archive with encodeObjectForKey (page 235).

```
public void encodeConditionalObjectForKey(Object objv, String key)
```

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

encodeDataObject

Encodes *datav*.

```
public void encodeDataObject(NSData datav)
```

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

See Also
decodeDataObject (page 248) (NSKeyedUnarchiver)

encodeDouble

Encodes *realv*.

```
public void encodeDouble(double realv)
```

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

See Also
decodeDouble (page 248) (NSKeyedUnarchiver)

encodeDoubleForKey

Encodes *realv* and associates it with the string *key*.

```
public void encodeDoubleForKey(double realv, String key)
```

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

See Also
decodeDoubleForKey (page 248) (NSKeyedUnarchiver)
decodeFloatForKey (page 249) (NSKeyedUnarchiver)

encodeFloat

Encodes *realv*.

```
public void encodeFloat(float realv)
```

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

See Also
decodeFloat (page 248) (NSKeyedUnarchiver)

encodeFloatForKey

Encodes *realv* and associates it with the string *key*.

```
public void encodeFloatForKey(float realv, String key)
```

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

See Also
decodeFloatForKey (page 249) (NSKeyedUnarchiver)
decodeDoubleForKey (page 248) (NSKeyedUnarchiver)

encodeInt

Encodes *intv*.

```
public void encodeInt(int intv)
```

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

See Also
decodeInt (page 249) (NSKeyedUnarchiver)

encodeIntForKey

Encodes *intv* and associates it with the string *key*.

```
public void encodeIntForKey(int intv, String key)
```

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

See Also
decodeIntForKey (page 249) (NSKeyedUnarchiver)
decodeShortForKey (page 252) (NSKeyedUnarchiver)
decodeLongForKey (page 250) (NSKeyedUnarchiver)

encodeLong

Encodes *longv*.

```
public void encodeLong(long longv)
```

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

See Also
decodeLong (page 250) (NSKeyedUnarchiver)

encodeLongForKey

Encodes *longv* and associates it with the string *key*.

```
public void encodeLongForKey(long longv, String key)
```

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

See Also
decodeLongForKey (page 250) (NSKeyedUnarchiver)
decodeShortForKey (page 252) (NSKeyedUnarchiver)
decodeIntForKey (page 249) (NSKeyedUnarchiver)

encodeObject

Encodes *objv*.

```
public void encodeObject(Object objv)
```

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

See Also
decodeObject (page 250) (NSKeyedUnarchiver)

encodeObjectForKey

Encodes *objv* and associates it with the string *key*.

```
public void encodeObjectForKey(Object objv, String key)
```

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

See Also
decodeObjectForKey (page 251) (NSKeyedUnarchiver)

encodePointForKey

Encodes *pointv* and associates it with the string *key*.

```
public void encodePointForKey(NSPoint pointv, String key)
```

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

See Also
decodePointForKey (page 251) (NSKeyedUnarchiver)

encodeRectForKey

Encodes *rectv* and associates it with the string *key*.

```
public void encodeRectForKey(NSRect rectv, String key)
```

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

See Also
decodeRectForKey (page 251) (NSKeyedUnarchiver)

encodeShort

Encodes *shortv*.

```
public void encodeShort(short shortv)
```

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

See Also
decodeShort (page 251) (NSKeyedUnarchiver)

encodeShortForKey

Encodes *shortv* and associates it with the string *key*.

```
public void encodeShortForKey(short shortv, String key)
```

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

See Also
decodeShortForKey (page 252) (NSKeyedUnarchiver)
decodeIntForKey (page 249) (NSKeyedUnarchiver)
decodeLongForKey (page 250) (NSKeyedUnarchiver)

encodeSizeForKey

Encodes *sizev* and associates it with the string *key*.

```
public void encodeSizeForKey(NSSize sizev, String key)
```

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

See Also
decodeSizeForKey (page 252) (NSKeyedUnarchiver)

finishEncoding

Tells the receiver that you have finished encoding objects, allowing it to construct the final data stream.

```
public 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
NSKeyedArchiver (page 228)

outputFormat

Returns the format in which the receiver encodes its data.

```
public int outputFormat()
```

Discussion
The available formats are `NSPropertyList.PropertyListXMLFormat` and `NSPropertyList.PropertyListBinaryFormat`.

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

See Also
`setOutputFormat` (page 237)

setClassNameForClass

Adds a class translation mapping to the receiver whereby instances of *`cls`* are encoded with the class name *`codedName`* instead of their real class names.

```
public void setClassNameForClass(String codedName, Class cls)
```

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 230)
`setGlobalClassNameForClass` (page 229)

setDelegate

Sets the receiver's delegate.

```
public void setDelegate(Object delegate)
```

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

See Also
`delegate` (page 230)

setOutputFormat

Sets the format in which the receiver encodes its data.

```
public void setOutputFormat(int format)
```

Discussion
`format` can be `NSPropertyList.PropertyListXMLFormat` or `NSPropertyList.PropertyListBinaryFormat`.

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

See Also
`outputFormat` (page 236)

versionForClassName

Returns the current class version number for the class named *className*.

```
public int versionForClassName(String className)
```

Discussion
Keyed archives do not record class version numbers like non-keyed archives do. Objects can explicitly encode and decode version numbers if desired.

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

Delegate Methods

archiverDidEncodeObject

Informs the delegate that *object* has been encoded.

```
public abstract void archiverDidEncodeObject(NSKeyedArchiver archiver, Object
    object)
```

Discussion
The delegate might restore some state it had modified previously, or use this opportunity to keep track of the objects that are encoded. *object* may be `null`.

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.

archiverDidFinish

Notifies the delegate that encoding has finished.

```
public abstract void archiverDidFinish(NSKeyedArchiver archiver)
```

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

archiverWillEncodeObject

Informs the delegate that *object* is about to be encoded.

```
public abstract Object archiverWillEncodeObject(NSKeyedArchiver archiver, Object
    object)
```

Discussion
The delegate either returns *object* or can return a different object to be encoded instead. The delegate can also modify the coder state. If the delegate returns `null`, `null` is encoded.

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 `null` is about to be encoded.

This method is called whether or not the object is being encoded conditionally.

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

archiverWillFinish

Notifies the delegate that encoding is about to finish.

```
public abstract void archiverWillFinish(NSKeyedArchiver archiver)
```

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

archiverWillReplaceObject

Informs the delegate that *newObject* is being substituted for *object*.

```
public abstract void archiverWillReplaceObject(NSKeyedArchiver archiver, Object
    object, Object newObject)
```

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.

NSKeyedUnarchiver

Inherits from	NSCoder : NSObject
Package:	com.apple.cocoa.foundation
Availability	Available in Mac OS X v10.2 and later.
Companion guide	Archives and Serializations Programming Guide for Cocoa

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 246) returns false, `decodeIntForKey` (page 249) returns 0, and `decodeObjectForKey` (page 251) returns null.

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 throws a `RangeException`. Further, when trying to coerce a value to an incompatible type, for example decoding an `int` as a `float`, the decoding method throws an `InvalidUnarchiveOperationException`.

Tasks

Constructors

`NSKeyedUnarchiver` (page 244)
Creates an empty NSKeyedUnarchiver.

Unarchiving Data

`unarchiveObjectWithData` (page 245)
Decodes the object graph previously encoded by NSKeyedArchiver and stored in *data*.

`unarchiveObjectWithFile` (page 245)
Decodes the object graph previously encoded by NSKeyedArchiver written to the file *path*.

Decoding Data

`containsValueForKey` (page 246)
Returns a Boolean value that indicates whether the archive contains a value for a string within the current decoding scope.

`decodeBoolForKey` (page 246)
Decodes a boolean value associated with the string *key*.

`decodeByte` (page 247)
Decodes a byte.

`decodeByteForKey` (page 247)
Decodes a byte associated with the string *key*.

`decodeChar` (page 247)
Decodes a `char` value.

`decodeCharForKey` (page 247)
Decodes a `char` value associated with the string *key*.

`decodeDataObject` (page 248)
Decodes an NSData object.

`decodeDouble` (page 248)
Decodes a double-precision floating-point value.

`decodeDoubleForKey` (page 248)
Decodes a double-precision floating-point value associated with the string *key*.

`decodeFloat` (page 248)
Decodes a single-precision floating-point value.

`decodeFloatForKey` (page 249)
Decodes a single-precision floating-point value associated with the string *key*.

`decodeInt` (page 249)
Decodes an integer value.

`decodeIntForKey` (page 249)
Decodes an integer value associated with the string *key*.

`decodeLong` (page 250)
Decodes a `long` value.

`decodeLongForKey` (page 250)
Decodes a `long` value associated with the string *key*.

`decodeObject` (page 250)
Decodes an arbitrary Object.

`decodeObjectForKey` (page 251)
Decodes an object associated with the string *key*.

`decodePointForKey` (page 251)
Decodes an NSPoint associated with the string *key*.

`decodeRectForKey` (page 251)
Decodes an NSRect associated with the string *key*.

`decodeShort` (page 251)
Decodes a `short` value.

`decodeShortForKey` (page 252)
Decodes a `short` value associated with the string *key*.

`decodeSizeForKey` (page 252)
Decodes an NSSize associated with the string *key*.

`finishDecoding` (page 253)
Tells the receiver that you are finished decoding objects, allowing the receiver to notify its delegate and to perform any final operations on the archive.

Managing Delegates

`setDelegate` (page 253)
Sets the receiver's delegate.

`delegate` (page 252)
Returns the receiver's delegate.

Managing Class Names

`setGlobalClassForClassName` (page 245)
Adds a class translation mapping to NSKeyedUnarchiver whereby encoded objects with the class name *codedName* are decoded as instances of the class *cls* instead.

`globalClassForClassName` (page 244)
Returns the class from which NSKeyedUnarchiver instantiates an encoded object with the class name *codedName*.

`setClassForClassName` (page 253)
Adds a class translation mapping to the receiver whereby encoded objects with the class name *codedName* are decoded as instances of the class *cls* instead.

`classForClassName` (page 246)
Returns the class from which the receiver instantiates an encoded object with the class name *codedName*.

Querying an Unarchiver

`versionForClassName` (page 253)
Returns the current class version number for the class named *className*.

Decoding objects

`unarchiverCannotDecodeObject` (page 254) *delegate method*
Informs the delegate that the named class, *name*, is not available during decoding.

`unarchiverDidDecodeObject` (page 254) *delegate method*
Informs the delegate that *object* has been decoded.

`unarchiverWillReplaceObject` (page 255) *delegate method*
Informs the delegate that *newObject* is being substituted for *object*.

Finishing decoding

`unarchiverDidFinish` (page 254) *delegate method*
Notifies the delegate that decoding has finished.

`unarchiverWillFinish` (page 255) *delegate method*
Notifies the delegate that decoding is about to finish.

Constructors

NSKeyedUnarchiver

Creates an empty NSKeyedUnarchiver.

```
public NSKeyedUnarchiver()
```

Discussion
Use the other constructor or the static methods `unarchiveObjectWithData` (page 245) or `unarchiveObjectWithFile` (page 245), instead.

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

Creates an NSKeyedUnarchiver with the *data* object as its archive and prepare the NSKeyedUnarchiver for a subsequent decode operation.

```
public NSKeyedUnarchiver(NSData data)
```

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

Static Methods

globalClassForClassName

Returns the class from which NSKeyedUnarchiver instantiates an encoded object with the class name *codedName*.

```
public static Class globalClassForClassName(String codedName)
```

Discussion
Returns `null` if NSKeyedUnarchiver does not have a translation mapping for *codedName*.

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

See Also
`setGlobalClassForClassName` (page 245)
`classForClassName` (page 246)

setGlobalClassForClassName

Adds a class translation mapping to NSKeyedUnarchiver whereby encoded objects with the class name *codedName* are decoded as instances of the class *cls* instead.

```
public static void setGlobalClassForClassName(Class cls, String codedName)
```

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
`globalClassForClassName` (page 244)
`setClassForClassName` (page 253)

unarchiveObjectWithData

Decodes the object graph previously encoded by NSKeyedArchiver and stored in *data*.

```
public static Object unarchiveObjectWithData(NSData data)
```

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

unarchiveObjectWithFile

Decodes the object graph previously encoded by NSKeyedArchiver written to the file *path*.

```
public static Object unarchiveObjectWithFile(String path)
```

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

Instance Methods

classForClassName

Returns the class from which the receiver instantiates an encoded object with the class name *codedName*.

```
public Class classForClassName(String codedName)
```

Discussion
Returns `null` if the receiver does not have a translation mapping for *codedName*. The class's separate translation map is not searched.

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

See Also
`setClassForClassName` (page 253)
`globalClassForClassName` (page 244)

containsValueForKey

Returns a Boolean value that indicates whether the archive contains a value for a string within the current decoding scope.

```
public boolean containsValueForKey(String key)
```

Discussion
The string is represented by *key*.

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

decodeBoolForKey

Decodes a boolean value associated with the string *key*.

```
public boolean decodeBoolForKey(String key)
```

Discussion
Returns `false` if *key* does not exist.

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

See Also
`encodeBoolForKey` (page 230) (NSKeyedArchiver)

decodeByte

Decodes a byte.

```
public byte decodeByte()
```

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

See Also
`encodeByte` (page 231) (NSKeyedArchiver)

decodeByteForKey

Decodes a byte associated with the string *key*.

```
public byte decodeByteForKey(String key)
```

Discussion
Returns 0 if *key* does not exist.

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

See Also
`encodeByteForKey` (page 231) (NSKeyedArchiver)

decodeChar

Decodes a `char` value.

```
public char decodeChar()
```

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

See Also
`encodeChar` (page 231) (NSKeyedArchiver)

decodeCharForKey

Decodes a `char` value associated with the string *key*.

```
public char decodeCharForKey(String key)
```

Discussion
Returns 0 if *key* does not exist.

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

See Also
encodeCharForKey (page 231) (NSKeyedArchiver)

decodeDataObject

Decodes an NSData object.

```
public NSData decodeDataObject()
```

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

See Also
encodeDataObject (page 232) (NSKeyedArchiver)

decodeDouble

Decodes a double-precision floating-point value.

```
public double decodeDouble()
```

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

See Also
encodeDouble (page 232) (NSKeyedArchiver)

decodeDoubleForKey

Decodes a double-precision floating-point value associated with the string *key*.

```
public double decodeDoubleForKey(String key)
```

Discussion
If the archived value was encoded as single-precision, the type is coerced. Returns 0.0 if *key* does not exist.

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

See Also
encodeDoubleForKey (page 233) (NSKeyedArchiver)
encodeFloatForKey (page 233) (NSKeyedArchiver)

decodeFloat

Decodes a single-precision floating-point value.

```
public float decodeFloat()
```

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

See Also
`encodeFloat` (page 233) (NSKeyedArchiver)

decodeFloatForKey

Decodes a single-precision floating-point value associated with the string *key*.

```
public float decodeFloatForKey(String key)
```

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 throws a `RangeException`. Returns 0.0 if *key* does not exist.

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

See Also
`encodeFloatForKey` (page 233) (NSKeyedArchiver)
`encodeDoubleForKey` (page 233) (NSKeyedArchiver)

decodeInt

Decodes an integer value.

```
public int decodeInt()
```

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

See Also
`encodeInt` (page 233) (NSKeyedArchiver)

decodeIntForKey

Decodes an integer value associated with the string *key*.

```
public int decodeIntForKey(String key)
```

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 throws a `RangeException`. Returns 0 if *key* does not exist.

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

See Also
encodeIntForKey (page 234) (NSKeyedArchiver)
encodeShortForKey (page 236) (NSKeyedArchiver)
encodeLongForKey (page 234) (NSKeyedArchiver)

decodeLong

Decodes a `long` value.

```
public long decodeLong()
```

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

See Also
encodeLong (page 234) (NSKeyedArchiver)

decodeLongForKey

Decodes a `long` value associated with the string *key*.

```
public long decodeLongForKey(String key)
```

Discussion
Returns 0 if *key* does not exist.

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

See Also
encodeLongForKey (page 234) (NSKeyedArchiver)
encodeShortForKey (page 236) (NSKeyedArchiver)
encodeIntForKey (page 234) (NSKeyedArchiver)

decodeObject

Decodes an arbitrary Object.

```
public Object decodeObject()
```

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

See Also
encodeObject (page 234)
encodeConditionalObject (page 232)

decodeObjectForKey

Decodes an object associated with the string *key*.

```
public Object decodeObjectForKey(String key)
```

Discussion
Returns `null` if *key* does not exist.

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

See Also
`encodeObjectForKey` (page 235) (NSKeyedArchiver)

decodePointForKey

Decodes an NSPoint associated with the string *key*.

```
public NSPoint decodePointForKey(String key)
```

Discussion
Returns `NSPoint.ZeroPoint` if *key* does not exist.

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

See Also
`encodePointForKey` (page 235) (NSKeyedArchiver)

decodeRectForKey

Decodes an NSRect associated with the string *key*.

```
public NSRect decodeRectForKey(String key)
```

Discussion
Returns `NSRect.ZeroRect` if *key* does not exist.

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

See Also
`encodeRectForKey` (page 235) (NSKeyedArchiver)

decodeShort

Decodes a `short` value.

```
public short decodeShort()
```

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

See Also
`encodeShort` (page 235) (NSKeyedArchiver)

decodeShortForKey

Decodes a `short` value associated with the string *key*.

```
public short decodeShortForKey(String key)
```

Discussion
Returns 0 if *key* does not exist.

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

See Also
`encodeShortForKey` (page 236) (NSKeyedArchiver)
`encodeIntForKey` (page 234) (NSKeyedArchiver)
`encodeLongForKey` (page 234) (NSKeyedArchiver)

decodeSizeForKey

Decodes an NSSize associated with the string *key*.

```
public NSSize decodeSizeForKey(String key)
```

Discussion
Returns `NSSize.ZeroSize` if *key* does not exist.

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

See Also
`encodeSizeForKey` (page 236) (NSKeyedArchiver)

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

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

See Also
`setDelegate` (page 253)

finishDecoding

Tells the receiver that you are finished decoding objects, allowing the receiver to notify its delegate and to perform any final operations on the archive.

```
public void finishDecoding()
```

Discussion
Once this method is invoked, the receiver cannot decode any further values.

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

setClassForClassName

Adds a class translation mapping to the receiver whereby encoded objects with the class name *codedName* are decoded as instances of the class *cls* instead.

```
public void setClassForClassName(Class cls, String codedName)
```

Discussion
When decoding, 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
`classForClassName` (page 246)
`setGlobalClassForClassName` (page 245)

setDelegate

Sets the receiver's delegate.

```
public void setDelegate(Object delegate)
```

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

See Also
`delegate` (page 252)

versionForClassName

Returns the current class version number for the class named *className*.

```
public int versionForClassName(String className)
```

Discussion
Keyed archives do not record class version numbers like non-keyed archives do, so the value returned by this method is not the version number for the class in the archive. Objects can explicitly encode and decode version numbers if desired.

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

Delegate Methods

unarchiverCannotDecodeObject

Informs the delegate that the named class, *name*, is not available during decoding.

```
public abstract Class unarchiverCannotDecodeObject(NSKeyedUnarchiver unarchiver,
    String name, NSArray classNames)
```

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 `null`, unarchiving aborts and the method throws an `InvalidUnarchiveOperationException`. The first element in *classNames* is the class name string of the encoded object, the second element is the class name of its immediate superclass, and so on.

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

unarchiverDidDecodeObject

Informs the delegate that *object* has been decoded.

```
public abstract Object unarchiverDidDecodeObject(NSKeyedUnarchiver unarchiver,
    Object object)
```

Discussion
The delegate can either return *object* or return a different object to replace the decoded one. *object* may be `null`. If the delegate returns `null`, `null` is the result of decoding the object.

The delegate may use this method to keep track of the decoded objects.

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

unarchiverDidFinish

Notifies the delegate that decoding has finished.

```
public abstract void unarchiverDidFinish(NSKeyedUnarchiver unarchiver)
```

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

unarchiverWillFinish

Notifies the delegate that decoding is about to finish.

```
public abstract void unarchiverWillFinish(NSKeyedUnarchiver unarchiver)
```

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

unarchiverWillReplaceObject

Informs the delegate that *newObject* is being substituted for *object*.

```
public abstract void unarchiverWillReplaceObject(NSKeyedUnarchiver unarchiver,
    Object object, Object newObject)
```

Discussion
This method is called even when the delegate itself is doing, or has done, the substitution with `unarchiverDidDecodeObject` (page 254).

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.

NSKeyValue

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Key-Value Coding Programming Guide

Overview

The NSKeyValue class provides methods for modifying the properties of an object using key-value coding. This class goes beyond the capabilities of the NSKeyValueCoding interface by taking advantage of the information available for an object's NSClassDescription. A class description identifies by name (or key) the properties (or attributes) that can be accessed for objects of a given class. Properties can contain multiple values or consist of one-to-one or one-to-many relationships with other objects. NSKeyValue uses this information to provide convenient methods for accessing individual elements of array properties and for automatically managing reciprocal relationships between objects.

Tasks

Constructors

`NSKeyValue` (page 259)
Returns an empty NSKeyValue.

Getting Values

`valueAtIndexInPropertyWithKey` (page 263)
Returns the contents of location *`index`* of the to-many property identified by *`key`* for *`anObject`*.

`valueForKey` (page 263)
Returns the property identified by *`key`* for *`anObject`*.

`valueForKeyPath` (page 263)
Returns the value of *`anObject`* for the derived property identified by *`keyPath`*.

`valuesForKeys` (page 263)
Returns a dictionary containing the property values for *`anObject`* identified by each element of *`keys`*.

`valueWithNameInPropertyWithKey` (page 264)
Retrieves a single value from a multi-value key for the object *`anObject`*.

`valueWithUniqueIDInPropertyWithKey` (page 264)
Retrieves a single value from a multi-value key for the object *anObject*.

`coerceValueForKey` (page 260)
Returns *value* coerced to the proper data type needed for the property identified by *key* for *anObject*.

Setting Values

`takeStoredValueForKey` (page 261)
Sets *anObject*'s property identified by *key* to *value*.

`takeValueForKey` (page 262)
Sets the property of *anObject* identified by *key* to *value*.

`takeValueForKeyPath` (page 262)
Sets the property of *anObject* identified by *keyPath* to *value*.

Adding

`addObjectToBothSidesOfRelationshipWithKey` (page 259)
Sets or adds *value* as the destination for the relationship identified by *key* for *anObject* and also sets or adds *anObject* for any reciprocal relationship with *value*, if there is one.

`addObjectToPropertyWithKey` (page 259)
Adds *value* to *anObject*'s to-many property identified by *key*, without setting a reciprocal relationship.

`insertValueAtIndexInPropertyWithKey` (page 260)
Inserts *value* at location *index* of the to-many property identified by *key* for *anObject*.

`insertValueInPropertyWithKey` (page 260)
Inserts a single value in a multivalue key at a reasonable index for *anObject*.

Removing

`removeObjectFromBothSidesOfRelationshipWithKey` (page 261)
Removes *value* as the destination of the relationship identified by *key* for *anObject* and also removes *anObject* for any reciprocal relationship with *value*, if there is one.

`removeObjectFromPropertyWithKey` (page 261)
Removes *value* from the to-many property identified by *key* for *anObject*.

`removeValueAtIndexFromPropertyWithKey` (page 261)
Removes the value at location *index* of the to-many property identified by *key* for *anObject*.

Replacing

`replaceValueAtIndexInPropertyWithKeyWithValue` (page 261)
Replaces the contents of location *index* of the to-many property identified by *key* for *anObject* with the value *value*.

Using Metadata

`classDescription` (page 260)
Returns the NSClassDescription object for *anObject*.

`objectSpecifier` (page 260)
Returns the object specifier for *anObject*.

Constructors

NSKeyValue

Returns an empty NSKeyValue.

```
public NSKeyValue()
```

Discussion
NSKeyValue contains only static methods, so you do not need to create instances.

Static Methods

addObjectToBothSidesOfRelationshipWithKey

Sets or adds *value* as the destination for the relationship identified by *key* for *anObject* and also sets or adds *anObject* for any reciprocal relationship with *value*, if there is one.

```
public static void addObjectToBothSidesOfRelationshipWithKey(Object anObject, Object
    value, String key)
```

Discussion
This method removes any previous link *anObject* has for relationship *key*. For example, if an Employee object belongs to the Research department, invoking this method with the Maintenance department as the new *value* removes the Employee from the Research department employee list as well as setting the Employee's department to Maintenance.

See Also
`removeObjectFromBothSidesOfRelationshipWithKey` (page 261)

addObjectToPropertyWithKey

Adds *value* to *anObject*'s to-many property identified by *key*, without setting a reciprocal relationship.

```
public static void addObjectToPropertyWithKey(Object anObject, Object value, String
    key)
```

Discussion
The *key* property should be an NSMutableArray.

See Also
removeObjectFromPropertyWithKey (page 261)

classDescription

Returns the NSClassDescription object for *anObject*.

```
public static NSClassDescription classDescription(Object anObject)
```

coerceValueForKey

Returns *value* coerced to the proper data type needed for the property identified by *key* for *anObject*.

```
public static Object coerceValueForKey(Object anObject, Object value, String key)
```

Discussion
If *anObject* does not have an NSScriptClassDescription, which allows properties to be typed, registered as its class description, `value` is returned unchanged.

insertValueAtIndexInPropertyWithKey

Inserts *value* at location *index* of the to-many property identified by *key* for *anObject*.

```
public static void insertValueAtIndexInPropertyWithKey(Object anObject, Object
    value, int index, String key)
```

Discussion
The *key* property should be an NSMutableArray.

See Also
removeValueAtIndexFromPropertyWithKey (page 261)
replaceValueAtIndexInPropertyWithKeyWithValue (page 261)
valueAtIndexInPropertyWithKey (page 263)

insertValueInPropertyWithKey

Inserts a single value in a multivalue key at a reasonable index for *anObject*.

```
public static void insertValueInPropertyWithKey(Object anObject, Object value,
    String key)
```

Discussion
The method `insertIn<Key>` is invoked on *anObject* if it exists. Otherwise, throws an `NSUnknownKeyException`. This is part of Cocoa's scripting support for inserting newly-created objects into containers without explicitly specifying a location.

objectSpecifier

Returns the object specifier for *anObject*.

```
public static NSScriptObjectSpecifier objectSpecifier(Object anObject)
```

removeObjectFromBothSidesOfRelationshipWithKey

Removes *value* as the destination of the relationship identified by *key* for *anObject* and also removes *anObject* for any reciprocal relationship with *value*, if there is one.

```
public static void removeObjectFromBothSidesOfRelationshipWithKey(Object anObject,
    Object value, String key)
```

See Also
addObjectToBothSidesOfRelationshipWithKey (page 259)

removeObjectFromPropertyWithKey

Removes *value* from the to-many property identified by *key* for *anObject*.

```
public static void removeObjectFromPropertyWithKey(Object anObject, Object value,
    String key)
```

See Also
addObjectToPropertyWithKey (page 259)

removeValueAtIndexFromPropertyWithKey

Removes the value at location *index* of the to-many property identified by *key* for *anObject*.

```
public static void removeValueAtIndexFromPropertyWithKey(Object anObject, int index,
    String key)
```

See Also
insertValueAtIndexInPropertyWithKey (page 260)

replaceValueAtIndexInPropertyWithKeyWithValue

Replaces the contents of location *index* of the to-many property identified by *key* for *anObject* with the value *value*.

```
public static void replaceValueAtIndexInPropertyWithKeyWithValue(Object anObject,
    int index, String key, Object value)
```

See Also
insertValueAtIndexInPropertyWithKey (page 260)

takeStoredValueForKey

Sets *anObject*'s property identified by *key* to *value*.

```
public static void takeStoredValueForKey(Object anObject, Object value, String key)
```

Discussion
Similar to the implementation of `takeValueForKey` (page 262), 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`", `takeStoredValueForKey` 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`", `takeStoredValueForKey` looks for an instance variable named `_lastName` or `lastName`.
3. If neither a private accessor nor an instance variable is found, `takeStoredValueForKey` searches for a public accessor method based on *key*. For the key "`lastName`", this would be `setLastName`.

takeValueForKey

Sets the property of *anObject* identified by *key* to *value*.

```
public static void takeValueForKey(Object anObject, Object value, String key)
```

Discussion
The default implementation works as follows:

1. Searches for a public accessor method of the form `set`*Key*, invoking it if there is one.
2. If a public accessor method is not found, searches for a private accessor method of the form `_set`*Key*, invoking it if there is one.
3. If an accessor method is not found, `takeValueForKey` searches for an instance variable based on *key* and sets *value* directly. For the *key* "`lastName`", this would be `_lastName` or `lastName`.

See Also
`valueForKey` (page 263)

takeValueForKeyPath

Sets the property of *anObject* identified by *keyPath* to *value*.

```
public static void takeValueForKeyPath(Object anObject, Object value, String keyPath)
```

Discussion
A key path has the form *relationship.property* (with one or more relationships). The default implementation gets the destination object for each relationship using `valueForKey` (page 263) and sends the final object a `takeValueForKey` (page 262) message with *value* and *property*.

See Also
`valueForKeyPath` (page 263)

valueAtIndexInPropertyWithKey

Returns the contents of location *index* of the to-many property identified by *key* for *anObject*.

```
public static Object valueAtIndexInPropertyWithKey(Object anObject, int index,
    String key)
```

See Also
insertValueAtIndexInPropertyWithKey (page 260)

valueForKey

Returns the property identified by *key* for *anObject*.

```
public static Object valueForKey(Object anObject, String key)
```

Discussion
The default implementation works as follows:

1. Searches for a public accessor method based on *key*. For example, with a *key* of "lastName", valueForKey looks for a method named getLastName or lastName.
2. If a public accessor method is not found, searches for a private accessor method based on *key* (a method preceded by an underbar). For example, with a *key* of "lastName", valueForKey looks for a method named _getLastName or _lastName.
3. If an accessor method is not found, valueForKey searches for an instance variable based on *key* and returns its value directly. For the *key* "lastName", this would be _lastName or lastName.

See Also
takeValueForKey (page 262)

valueForKeyPath

Returns the value of *anObject* for the derived property identified by *keyPath*.

```
public static Object valueForKeyPath(Object anObject, String keyPath)
```

Discussion
A key path has the form *relationship.property* (with one or more relationships). The default implementation gets the destination object for each relationship using valueForKey (page 263) and returns the result of a valueForKey (page 263) message to the final object.

See Also
takeValueForKeyPath (page 262)

valuesForKeys

Returns a dictionary containing the property values for *anObject* identified by each element of *keys*.

```
public static NSDictionary valuesForKeys(Object anObject, NSArray keys)
```

Discussion
The default implementation invokes `valueForKey` (page 263) for each key in keys, substituting NSNulls in the dictionary for returned `null` values.

valueWithNameInPropertyWithKey

Retrieves a single value from a multi-value key for the object *anObject*.

```
public static Object valueWithNameInPropertyWithKey(Object anObject, String name,
    String key)
```

Discussion
The method `valueIn<Key>WithName` is invoked on *anObject* if it exists. Otherwise, an exception is thrown.

valueWithUniqueIDInPropertyWithKey

Retrieves a single value from a multi-value key for the object *anObject*.

```
public static Object valueWithUniqueIDInPropertyWithKey(Object anObject, Object
    uniqueID, String key)
```

Discussion
The method `valueIn<Key>WithUniqueID` is invoked on *anObject* if it exists. Otherwise, an exception is thrown. The declared type of *uniqueID* in the constructed method must be Object, String, or one of the scalar types that can be encapsulated by the numeric classes, such as Integer or Double.

Constants

The following constant is defined in this informal interface.

Constant	Description
`OperationNot-SupportedForKeyException`	Can be thrown by key-value coding methods that want to explicitly disallow certain manipulations or accesses. For instance, a `takeValueForKey` method for a read-only key can throw this exception.

NSLogicalTest

Inherits from	NSScriptWhoseTest : NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Cocoa Scripting Guide

Overview

Instances of this class perform logical operations of AND, OR, and NOT on Boolean expressions represented by NSSpecifierTests. These operators are equivalent to "&&," "||," and "!" in the C language. For AND and OR operations, an NSLogicalTest is typically initialized with an array containing two or more NSSpecifierTests. `isTrue` (page 558), inherited from NSScriptWhoseTest, evaluates the array in a manner appropriate to the logical operation. For NOT operations, an NSLogicalTest is initialized with only one NSSpecifierTest; it simply reverses the Boolean outcome of the `isTrue` (page 558) method.

You don't normally subclass NSLogicalTest.

Constants

The following constants are defined by NSLogicalTest:

Constant	Description
`AndLogicalTest`	Specifies a logical AND test.
`NotLogicalTest`	Specifies a logical NOT test.
`OrLogicalTest`	Specifies a logical OR test.

Tasks

Constructors

`NSLogicalTest` (page 266)
Returns an NSLogicalTest with no data.

Constructors

NSLogicalTest

Returns an NSLogicalTest with no data.

```
public NSLogicalTest()
```

Discussion
Do not use this constructor.

Returns an NSLogicalTest initialized with logical test of type *testType* and argument *anObject*.

```
public NSLogicalTest(int testType, Object anObject)
```

Discussion
If *testType* is `NotLogicalTest`, *anObject* is a single NSScriptWhoseTest object. For other *testType* values, *anObject* is an NSArray object holding two or more NSScriptWhoseTest objects.

NSMetadataItem

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Availability	Available in Mac OS X v10.4 and later.

Overview

NSMetadataItem encapsulates the metadata associated with a file, providing a simple interface to retrieve the available attribute names and values.

Tasks

Constructors

`NSMetadataItem` (page 267)

Getting Item Attributes

`attributes` (page 268)
Returns an array containing the attribute names of the receiver's values.

`valueForAttribute` (page 268)
Returns the receiver's metadata attribute name specified by *key*.

`valuesForAttributes` (page 268)
Returns a dictionary containing the key-value pairs for the attribute names specified by *keys*.

Constructors

NSMetadataItem

```
public NSMetadataItem()
```

Discussion
Creates an empty NSMetadataItem object.

Instance Methods

attributes

Returns an array containing the attribute names of the receiver's values.

```
public NSArray attributes()
```

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

valueForAttribute

Returns the receiver's metadata attribute name specified by *key*.

```
public Object valueForAttribute(String key)
```

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

valuesForAttributes

Returns a dictionary containing the key-value pairs for the attribute names specified by *keys*.

```
public NSDictionary valuesForAttributes(NSArray keys)
```

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

NSMetadataQuery

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Availability	Available in Mac OS X v10.4 and later.

Overview

NSMetadataQuery provides an object-oriented encapsulation of the MDQuery functionality for querying the Spotlight metadata.

NSMetadataQuery provides the 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.

Tasks

Constructors

`NSMetadataQuery` (page 271)

Setting the Search Scope

`searchScopes` (page 275)
Returns an array containing the receiver's search scopes.

`setSearchScopes` (page 277)
Sets the locations searched by the receiver.

Setting the Delegate

`delegate` (page 272)
Returns the delegate used by the receiver, or `null` if there is none.

`setDelegate` (page 276)
Sets the receiver's delegate to *`delegate`*.

Setting the Query Attributes

`predicate` (page 274)
Returns the predicate the receiver uses to filter query results.

`setPredicate` (page 277)
Sets the predicate used by the receiver to filter the query results.

`sortDescriptors` (page 278)
Returns an array containing the receiver's sort descriptors.

`setSortDescriptors` (page 277)
Sets the sort descriptors used by the receiver to *`descriptors`*.

`valueListAttributes` (page 279)
Returns an array containing the value list attributes the receiver generates.

`setValueListAttributes` (page 278)
Sets the value list attributes for the receiver to the specified attribute names.

`groupingAttributes` (page 273)
Returns the receiver's grouping attributes.

`setGroupingAttributes` (page 276)
Sets the receiver's grouping attributes to the attribute names specified in *`attrs`*.

`notificationBatchingInterval` (page 274)
Returns the interval that the receiver provides notification of updated query results.

`setNotificationBatchingInterval` (page 276)
Sets the interval between update notifications sent by the receiver to *`timeInterval`*.

Running the Query

`startQuery` (page 278)
Attempts to start the query, returning `true` if successful.

`stopQuery` (page 279)
Stops the receiver's current query from gathering any further results.

`isStarted` (page 274)
Returns a Boolean value that indicates whether the receiver has received a `startQuery` (page 278) message.

`isGathering` (page 273)
Returns a Boolean value that indicates whether the receiver is in the initial gathering phase of the query.

`isStopped` (page 274)
Returns a Boolean value that indicates whether the receiver has received a `stopQuery` (page 279) message.

Getting Query Results

`resultCount` (page 275)
Returns the number of results returned by the receiver.

`resultAtIndex` (page 275)
Returns the query result at *`idx`*.

`results` (page 275)
Returns an array representation of the result objects for the receiver.

`disableUpdates` (page 272)
Disables updates to the query results.

`enableUpdates` (page 272)
Enables updates to the query results.

`indexOfResult` (page 273)
Returns the index of *`result`* in the receiver's results array.

`valueLists` (page 279)
Returns a dictionary containing the value lists generated by the receiver.

`groupedResults` (page 272)
Returns an array containing hierarchical groups of query results based on the receiver's grouping attributes.

`valueOfAttributeForResultAtIndex` (page 279)
Returns the value for the attribute name *`attrName`* at the index in the results specified by *`idx`*.

Constructors

NSMetadataQuery

```
public NSMetadataQuery()
```

Discussion
Creates an empty NSMetadataQuery object.

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

Instance Methods

delegate

Returns the delegate used by the receiver, or `null` if there is none.

```
public Object delegate()
```

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

See Also
`setDelegate` (page 276)

disableUpdates

Disables updates to the query results.

```
public void disableUpdates()
```

Discussion
This method should be called 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 272)

enableUpdates

Enables updates to the query results.

```
public void enableUpdates()
```

Discussion
This method should be called when finished iterating over the query results.

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

See Also
`disableUpdates` (page 272)

groupedResults

Returns an array containing hierarchical groups of query results based on the receiver's grouping attributes.

```
public NSArray groupedResults()
```

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

See Also
`groupingAttributes` (page 273)
`setGroupingAttributes` (page 276)

groupingAttributes

Returns the receiver's grouping attributes.

```
public NSArray groupingAttributes()
```

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

See Also
`setGroupingAttributes` (page 276)

indexOfResult

Returns the index of *result* in the receiver's results array.

```
public int indexOfResult(Object result)
```

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

See Also
`resultAtIndex` (page 275)

isGathering

Returns a Boolean value that indicates whether the receiver is in the initial gathering phase of the query.

```
public boolean isGathering()
```

Discussion
Queries have two phases: the initial gathering phase that collects all currently matching results and a second live-update phase.

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

See Also
`isStarted` (page 274)
`isStopped` (page 274)

isStarted

Returns a Boolean value that indicates whether the receiver has received a `startQuery` (page 278) message.

```
public boolean isStarted()
```

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

See Also
`isGathering` (page 273)
`isStopped` (page 274)

isStopped

Returns a Boolean value that indicates whether the receiver has received a `stopQuery` (page 279) message.

```
public boolean isStopped()
```

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

See Also
`isGathering` (page 273)
`isStarted` (page 274)

notificationBatchingInterval

Returns the interval that the receiver provides notification of updated query results.

```
public double notificationBatchingInterval()
```

Discussion
The default is 1.0.

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

See Also
`setNotificationBatchingInterval` (page 276)

predicate

Returns the predicate the receiver uses to filter query results.

```
public NSPredicate predicate()
```

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

See Also
`setPredicate` (page 277)

resultAtIndex

Returns the query result at *idx*.

```
public Object resultAtIndex(int idx)
```

Discussion
For performance reasons, you should use this method when retrieving a specific result, rather than they array returned by `results` (page 275).

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

See Also
`indexOfResult` (page 273)

resultCount

Returns the number of results returned by the receiver.

```
public int resultCount()
```

Discussion
For performance reasons, you should use this method, rather than invoking `count` on `results` (page 275).

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

results

Returns an array representation of the result objects for the receiver.

```
public NSArray results()
```

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 275) and `resultAtIndex` (page 275) methods.

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

See Also
`groupedResults` (page 272)

searchScopes

Returns an array containing the receiver's search scopes.

```
public NSArray searchScopes()
```

Discussion
The array can contain NSString or NSURL objects that represent file system directories or the search scopes specified in "Constants" (page 280). 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 277)

setDelegate

Sets the receiver's delegate to *`delegate`*.

```
public void setDelegate(Object delegate)
```

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

See Also
`delegate` (page 272)

setGroupingAttributes

Sets the receiver's grouping attributes to the attribute names specified in *`attrs`*.

```
public void setGroupingAttributes(NSArray attrs)
```

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
`groupingAttributes` (page 273)

setNotificationBatchingInterval

Sets the interval between update notifications sent by the receiver to *`timeInterval`*.

```
public void setNotificationBatchingInterval(double timeInterval)
```

Discussion
The default is 1.0.

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

See Also
notificationBatchingInterval (page 274)

setPredicate

Sets the predicate used by the receiver to filter the query results.

```
public void setPredicate(NSPredicate predicate)
```

Discussion
The predicate is represented by *predicate*. You must set a predicate before starting a query. 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
setPredicate (page 277)

setSearchScopes

Sets the locations searched by the receiver.

```
public void setSearchScopes(NSArray scopes)
```

Discussion
By default the receiver has no limitation on its search scope. The *scopes* parameter is an array of NSString or NSURL objects that specify file system directories. You can also include the predefined search scopes specified in "Constants" (page 280). If *scopes* is an empty array, the receiver removes any search scope limitations.

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

See Also
searchScopes (page 275)

setSortDescriptors

Sets the sort descriptors used by the receiver to *descriptors*.

```
public void setSortDescriptors(NSArray 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 278)

setValueListAttributes

Sets the value list attributes for the receiver to the specified attribute names.

```
public void setValueListAttributes(NSArray attrs)
```

Discussion
The attribute names are passed in the *attrs* array. The query will collect the values of these attributes into uniqued lists that can be used to summarize the results of the query. If *attrs* is null, no value lists are generated. Note that value list collection increases CPU usage and significantly increases the memory usage of an NSMetadataQuery.

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
valueListAttributes (page 279)

sortDescriptors

Returns an array containing the receiver's sort descriptors.

```
public NSArray sortDescriptors()
```

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

See Also
setSortDescriptors (page 277)

startQuery

Attempts to start the query, returning true if successful.

```
public boolean startQuery()
```

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 279)

stopQuery

Stops the receiver's current query from gathering any further results.

```
public void stopQuery()
```

Discussion
The receiver will first complete 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` (page 278) after receiving this message, the existing results are discarded.

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

See Also
`startQuery` (page 278)

valueListAttributes

Returns an array containing the value list attributes the receiver generates.

```
public NSArray valueListAttributes()
```

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

See Also
`setValueListAttributes` (page 278)

valueLists

Returns a dictionary containing the value lists generated by the receiver.

```
public NSDictionary valueLists()
```

Discussion
The values are arrays of `NSMetadataQueryAttributeValueTuple`.

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

valueOfAttributeForResultAtIndex

Returns the value for the attribute name *`attrName`* at the index in the results specified by *`idx`*.

```
public Object valueOfAttributeForResultAtIndex(String attrName, int idx)
```

Discussion
Values can only be returned for attribute names that are specified to `setValueListAttributes` (page 278), as a sorting key in a specified sort descriptor, or as one of the grouping attributes specified set for the query.

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

Constants

The following constants specify the predefined search scopes used by `setSearchScopes` (page 277):

Constant	Description
`NSMetadataQueryUserHomeScope`	Search the user's home directory. Available for Mac OS X v10.4 and later.
`NSMetadataQueryLocal-ComputerScope`	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 for Mac OS X v10.4 and later.
`NSMetadataQueryNetworkScope`	Search all user-mounted remote volumes. Available for Mac OS X v10.4 and later.

In addition to the requested metadata attributes, a query result also includes the following attribute:

Constant	Description
`NSMetadataQuery-ResultContent-RelevanceAttribute`	An NSNumber 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 for Mac OS X v10.4 and later.

Notifications

NSMetadataQueryDidFinishGatheringNotification

Posted when the receiver has finished with the initial result-gathering phase of the query.

NSMetadataQueryDidStartGatheringNotification

Posted when the receiver begins with the initial result-gathering phase of the query.

NSMetadataQueryDidUpdateNotification

Posted when the receiver's results have changed during the live-update phase of the query.

NSMetadataQueryGatheringProgressNotification

Posted as the receiver's is collecting results during the initial result-gathering phase of the query.

NSMetadataQueryAttributeValueTuple

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Availability	Available in Mac OS X v10.4 and later.

Overview

NSMetadataQueryAttributeValueTuple encapsulates the attribute name and value of a metadata attribute. They are returned by NSMetadataQuery as the results in the value lists. Each instance contains the attribute name, the value, and the number of instances of that value that exist for the attribute name.

Tasks

Constructors

`NSMetadataQueryAttributeValueTuple` (page 283)

Getting Attribute Information

`attribute` (page 284)
Returns the receiver's attribute name.

`count` (page 284)
Returns the number of instances of the value that exist for the attribute name of the receiver.

`value` (page 284)
Returns the receiver's attribute value.

Constructors

NSMetadataQueryAttributeValueTuple

```
public NSMetadataQueryAttributeValueTuple()
```

Discussion
Creates an empty NSMetadataQueryAttributeValueTuple object.

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

Instance Methods

attribute

Returns the receiver's attribute name.

```
public String attribute()
```

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

count

Returns the number of instances of the value that exist for the attribute name of the receiver.

```
public int count()
```

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

value

Returns the receiver's attribute value.

```
public Object value()
```

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

NSMetadataQueryResultGroup

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Availability	Available in Mac OS X v10.4 and later.

Overview

NSMetadataQueryResultGroup encapsulates a collection of grouped attribute results returned by an NSMetadataQuery.

Tasks

Constructors

`NSMetadataQueryResultGroup` (page 286)

Getting Result Values

`attribute` (page 286)
Returns the attribute name for the receiver's result group.

`resultAtIndex` (page 286)
Returns the query result at *idx*.

`resultCount` (page 286)
Returns the number of results returned by the receiver.

`results` (page 287)
Returns an array representation of the result objects for the receiver.

`subgroups` (page 287)
Returns an array containing the subgroups of the receiver.

`value` (page 287)
Returns the value of the attribute name for the receiver.

Constructors

NSMetadataQueryResultGroup

```
public NSMetadataQueryResultGroup()
```

Discussion
Creates an empty NSMetadataQueryResultGroup object.

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

Instance Methods

attribute

Returns the attribute name for the receiver's result group.

```
public String attribute()
```

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

resultAtIndex

Returns the query result at *idx*.

```
public Object resultAtIndex(int idx)
```

Discussion
For performance reasons, you should use this method when retrieving a specific result, rather than they array returned by `results` (page 287).

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

resultCount

Returns the number of results returned by the receiver.

```
public int resultCount()
```

Discussion
For performance reasons, you should use this method, rather than invoking `count` on `results` (page 287).

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

results

Returns an array representation of the result objects for the receiver.

```
public NSArray results()
```

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 286) and `resultAtIndex` (page 286) methods.

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

subgroups

Returns an array containing the subgroups of the receiver.

```
public NSArray subgroups()
```

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

value

Returns the value of the attribute name for the receiver.

```
public Object value()
```

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

NSMiddleSpecifier

Inherits from	NSScriptObjectSpecifier : NSObject
Package:	com.apple.cocoa.foundation
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.

Tasks

Constructors

`NSMiddleSpecifier` (page 289)
Returns an NSMiddleSpecifier with no data.

Constructors

NSMiddleSpecifier

Returns an NSMiddleSpecifier with no data.

```
public NSMiddleSpecifier()
```

Discussion
Do not use this constructor.

Returns an NSMiddleSpecifier 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.

```
public NSMiddleSpecifier(NSScriptClassDescription classDescription,
    NSScriptObjectSpecifier specifier, String key)
```

Discussion
The receiver's child specifier reference is set to `null`.

Returns an NSMiddleSpecifier initialized with container specifier *specifier* and key *key*.

```
public NSMiddleSpecifier(NSScriptObjectSpecifier specifier, String key)
```

Discussion
The class description of the container is set automatically.

NSMoveCommand

Inherits from	NSScriptCommand : NSObject
Package:	com.apple.cocoa.foundation
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 command through key-value coding. Most applications don't need to subclass NSMoveCommand or invoke its methods.

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

Constructors

`NSMoveCommand` (page 292)
Returns an NSMoveCommand with no data.

Working with Specifiers

`keySpecifier` (page 292)
Returns a specifier for the object or objects to be moved.

`setReceiversSpecifier` (page 292)
Sets the receiver's object specifier; when evaluated, the specifier indicates the receiver or receivers of the Move command.

Constructors

NSMoveCommand

Returns an NSMoveCommand with no data.

```
public NSMoveCommand()
```

Discussion
Do not use this constructor.

Initializes an instance of NSMoveCommand with the command description supplied by *commandDescription*.

```
public NSMoveCommand(NSScriptCommandDescription commandDescription)
```

Discussion
Note that such an instance has no receiver specifier, arguments, or direct parameter and is not a fully functional command.

Instance Methods

keySpecifier

Returns a specifier for the object or objects to be moved.

```
public NSScriptObjectSpecifier keySpecifier()
```

Discussion
Note that this specifier may be different than the specifier set by `setReceiversSpecifier` (page 292), 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.

setReceiversSpecifier

Sets the receiver's object specifier; when evaluated, the specifier indicates the receiver or receivers of the Move command.

```
public void setReceiversSpecifier(NSScriptObjectSpecifier receiversRef)
```

Discussion
This method overrides `setReceiversSpecifier` (page 525) 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`.

NSMutableArray

Inherits from	NSArray : NSObject
Implements	NSCoding (NSArray)
Package:	com.apple.cocoa.foundation
Companion guides	Collections Programming Topics for Cocoa Key-Value Coding Programming Guide

Class at a Glance

An NSMutableArray stores a modifiable array of objects.

Principal Attributes

- A count of the number of objects in the array
- The list of objects contained in the array

`NSMutableArray` (page 295)
Creates a mutable array.

Commonly Used Methods

`insertObjectAtIndex` (page 297)
Inserts an object at a specified index.

`removeObject` (page 298)
Removes all occurrences of an object.

`removeObjectAtIndex` (page 299)
Removes the object at a given index.

`replaceObjectAtIndex` (page 300)
Replaces the object at a given index.

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.

NSMutableArray methods are conceptually based on these primitive methods:

- `addObject` (page 296)
- `insertObjectAtIndex` (page 297)
- `removeLastObject` (page 298)
- `removeObjectAtIndex` (page 299)
- `replaceObjectAtIndex` (page 300)

The other methods in its 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.

Tasks

Constructors

`NSMutableArray` (page 295)

Adding and Replacing Objects

`addObject` (page 296)
Inserts *`anObject`* at the end of the receiver.

`addObjectsFromArray` (page 296)
Adds the objects contained in *`otherArray`* to the end of the receiver's array of objects.

`insertObjectAtIndex` (page 297)
Inserts *`anObject`* into the receiver at *`index`*.

`insertObjectsAtIndexes` (page 297)
Inserts the objects in *`objects`* into the receiver at the indexes specified by *`indexes`*.

`replaceObjectAtIndex` (page 300)
Replaces the object at *`index`* with *`anObject`*.

`replaceObjectsAtIndexes` (page 300)
Replaces the objects in the receiver at the locations specified by *`indexes`* with the objects from *`objects`*.

`replaceObjectsInRange` (page 301)
Replaces the objects in the receiver specified by *`aRange`* with the objects in *`otherArray`* specified by *`otherRange`*.

`setArray` (page 301)
Sets the receiver's elements to those in *otherArray*.

Removing Objects

`filterUsingPredicate` (page 296)

`removeAllObjects` (page 297)
Empties the receiver of all its elements.

`removeIdenticalObject` (page 298)
This method has been deprecated.

`removeLastObject` (page 298)
Removes the object with the highest-valued index in the receiver.

`removeObject` (page 298)
Removes all occurrences of *anObject* throughout the receiver.

`removeObjectAtIndex` (page 299)
Removes the object at *index* and moves all elements beyond *index* by subtracting 1 from their indices to fill the gap.

`removeObjectsAtIndexes` (page 299)
Removes the objects at the specified indexes from the receiver.

`removeObjectsInArray` (page 299)
This method is similar to `removeObject` (page 298), but allows you to efficiently remove large sets of objects with a single operation.

`removeObjectsInRange` (page 300)
Removes each of the objects within the specified range, *aRange*, in the receiver using `removeObjectAtIndex` (page 299).

Rearranging Objects

`sortUsingDescriptors` (page 301)
Sorts the receiver as specified by *sortDescriptors*.

`sortUsingSelector` (page 302)
Sorts the receiver's elements in ascending order, as determined by the comparison method specified by the selector *selector*.

Constructors

NSMutableArray

```
public NSMutableArray()
```

Discussion
Creates an empty array.

```
public NSMutableArray(Object anObject)
```

Discussion
Creates an array containing the single element *anObject*.

```
public NSMutableArray(Object[] objects)
```

Discussion
Creates an array containing *objects*.

```
public NSMutableArray(NSArray anArray)
```

Discussion
Creates an array containing the objects in *anArray*.

Instance Methods

addObject

Inserts *anObject* at the end of the receiver.

```
public void addObject(Object anObject)
```

Discussion
If *anObject* is `null`, an `InvalidArgumentException` is thrown.

See Also
`addObjectsFromArray` (page 296)
`removeObject` (page 298)
`setArray` (page 301)

addObjectsFromArray

Adds the objects contained in *otherArray* to the end of the receiver's array of objects.

```
public void addObjectsFromArray(NSArray otherArray)
```

See Also
`setArray` (page 301)
`removeObject` (page 298)

filterUsingPredicate

```
public void filterUsingPredicate(NSPredicate predicate)
```

Discussion
Evaluates the *predicate* against the receiver's content and leaves only objects that match.

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

insertObjectAtIndex

Inserts *anObject* into the receiver at *index*.

```
public void insertObjectAtIndex(Object anObject, int index)
```

Discussion
If *index* is already occupied, the objects at *index* and beyond are shifted by adding 1 to their indices to make room. *index* cannot be greater than the number of elements in the array. This method throws an `InvalidArgumentException` if *anObject* is `null` and throws a `RangeException` if *index* is greater than the number of elements in the array.

Note that NSArrays 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. Because of this fact, you can only insert new objects in ascending order—with no gaps. Once you add two objects, the array's 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 throws an exception.

See Also
`removeObjectAtIndex` (page 299)

insertObjectsAtIndexes

Inserts the objects in *objects* into the receiver at the indexes specified by *indexes*.

```
public void insertObjectsAtIndexes(NSArray objects, NSIndexSet indexes)
```

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 locations specified by *indexes* may 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.

Availability
Available in Mac OS X version 10.4 and later.

See Also
`insertObjectAtIndex` (page 297)

removeAllObjects

Empties the receiver of all its elements.

```
public void removeAllObjects()
```

See Also
`removeObject` (page 298)
`removeLastObject` (page 298)
`removeObjectAtIndex` (page 299)
`removeIdenticalObject` (page 298)

removeIdenticalObject

This method has been deprecated.

```
public void removeIdenticalObject(Object anObject)
```

This method has been deprecated.

```
public void removeIdenticalObject(Object anObject, NSRange aRange)
```

See Also
removeAllObjects (page 297)
removeLastObject (page 298)
removeObject (page 298)
removeObjectAtIndex (page 299)

removeLastObject

Removes the object with the highest-valued index in the receiver.

```
public void removeLastObject()
```

Discussion
removeLastObject throws a RangeException if there are no objects in the receiver.

See Also
removeAllObjects (page 297)
removeObject (page 298)
removeObjectAtIndex (page 299)
removeIdenticalObject (page 298)

removeObject

Removes all occurrences of *anObject* throughout the receiver.

```
public void removeObject(Object anObject)
```

Discussion
This method uses indexOfObject (page 63) to locate matches and then removes them by using removeObjectAtIndex (page 299). Thus, matches are determined on the basis of an object's response to the equals message.

Removes all occurrences of *anObject* in the specified range, *aRange*, of the receiver.

```
public void removeObject(Object anObject, NSRange aRange)
```

Discussion
This method uses indexOfObject (page 63) to locate matches and then removes them by using removeObjectAtIndex (page 299). Thus, matches are determined on the basis of an object's response to the equals message.

See Also
removeAllObjects (page 297)

removeLastObject (page 298)
removeObjectAtIndex (page 299)
removeIdenticalObject (page 298)
removeObjectsInArray (page 299)

removeObjectAtIndex

Removes the object at *index* and moves all elements beyond *index* by subtracting 1 from their indices to fill the gap.

```
public void removeObjectAtIndex(int index)
```

Discussion
This method throws a RangeException if *index* is beyond the end of the receiver.

See Also
insertObjectAtIndex (page 297)
removeAllObjects (page 297)
removeLastObject (page 298)
removeObject (page 298)
removeIdenticalObject (page 298)

removeObjectsAtIndexes

Removes the objects at the specified indexes from the receiver.

```
public void removeObjectsAtIndexes(NSIndexSet indexes)
```

Discussion
This method is similar to removeObjectAtIndex (page 299), 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.

The locations specified by *indexes* must lie within the bounds of the receiver.

Availability
Available in Mac OS X version 10.4 and later.

See Also
removeObjectAtIndex (page 299)

removeObjectsInArray

This method is similar to removeObject (page 298), but allows you to efficiently remove large sets of objects with a single operation.

```
public void removeObjectsInArray(NSArray otherArray)
```

Discussion
It assumes that all elements in *otherArray*—which are the objects to be removed—respond to `hash` and `equals`.

See Also
`removeAllObjects` (page 297)
`removeIdenticalObject` (page 298)

removeObjectsInRange

Removes each of the objects within the specified range, *aRange*, in the receiver using `removeObjectAtIndex` (page 299).

```
public void removeObjectsInRange(NSRange aRange)
```

replaceObjectAtIndex

Replaces the object at *index* with *anObject*.

```
public void replaceObjectAtIndex(int index, Object anObject)
```

Discussion
This method throws an `InvalidArgumentException` if *anObject* is `null` and throws a `RangeException` if *index* is beyond the end of the receiver.

See Also
`insertObjectAtIndex` (page 297)
`removeObjectAtIndex` (page 299)

replaceObjectsAtIndexes

Replaces the objects in the receiver at the locations specified by *indexes* with the objects from *objects*.

```
public void replaceObjectsAtIndexes(NSIndexSet indexes, NSArray objects)
```

Discussion
The count of locations in *indexes* must equal the count of *objects*.

Availability
Available in Mac OS X version 10.4 and later.

See Also
`insertObjectAtIndex` (page 297)
`removeObjectAtIndex` (page 299)
`replaceObjectAtIndex` (page 300)

replaceObjectsInRange

Replaces the objects in the receiver specified by *aRange* with the objects in *otherArray* specified by *otherRange*.

```
public void replaceObjectsInRange(NSRange aRange, NSArray otherArray, NSRange
    otherRange)
```

Discussion
aRange and *otherRange* don't have to be equal; if *aRange* is greater than *otherRange*, the extra objects in the receiver are removed. If *otherRange* is greater than *aRange*, the extra objects from *otherArray* are inserted into the receiver.

See Also
insertObjectAtIndex (page 297)
removeObjectAtIndex (page 299)
replaceObjectAtIndex (page 300)

setArray

Sets the receiver's elements to those in *otherArray*.

```
public void setArray(NSArray otherArray)
```

Discussion
Shortens the receiver, if necessary, so that it contains no more than the number of elements in *otherArray*. Replaces existing elements in the receiver with the elements in *otherArray*. Finally, if there are more elements in *otherArray* than there are in the receiver, the additional items are then added.

See Also
addObjectsFromArray (page 296)
insertObjectAtIndex (page 297)

sortUsingDescriptors

Sorts the receiver as specified by *sortDescriptors*.

```
public void sortUsingDescriptors(NSArray sortDescriptors)
```

Discussion
See NSSortDescriptor (page 581) for additional information.

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

See Also
sortUsingSelector (page 302)
sortedArrayUsingDescriptors (page 65) (NSArray)

sortUsingSelector

Sorts the receiver's elements in ascending order, as determined by the comparison method specified by the selector *selector*.

```
public void sortUsingSelector(NSSelector selector)
```

Discussion

The *selector* message is sent to each object in the array and has as its single argument another object in the array. The *selector* method is used to compare two elements at a time and should return `OrderedAscending` if the receiver is smaller than the argument, `OrderedDescending` if the receiver is larger than the argument, and `OrderedSame` if they are equal.

See Also

`sortUsingDescriptors` (page 301)

`sortedArrayUsingSelector` (page 65) (NSArray)

NSMutableAttributedString

Inherits from	NSAttributedString : NSObject
Implements	NSCoding (NSAttributedString)
Package:	com.apple.cocoa.foundation
Companion guide	Attributed Strings Programming Guide

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 (page 67) for more information about attributed strings.

When working with the Application Kit, you must also clean up changed attributes using the various `fix...` methods.

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` (page 312) method replaces a range of characters with those from a string, leaving all attribute information outside that range intact. The primitive `setAttributesInRange` (page 312) 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.

Tasks

Constructors

`NSMutableAttributedString` (page 305)
Creates an empty NSMutableAttributedString.

Retrieving Character Information

`mutableStringReference` (page 310)
Returns the character contents of the receiver as an NSMutableStringReference object.

Changing Characters

`deleteCharactersInRange` (page 308)
Deletes the characters in *aRange* along with their associated attributes.

Changing Attributes

`setAttributesInRange` (page 312)
Sets the attributes for the characters in *aRange* to *attributes*.

`addAttributeInRange` (page 306)
Adds an attribute with the given *name* and *value* to the characters in *aRange*.

`addAttributesInRange` (page 307)
Adds the collection of attributes in *attributes* to the characters in *aRange*.

`removeAttributeInRange` (page 311)
Removes the attribute named *name* from the characters in *aRange*.

Changing Characters and Attributes

`appendAttributedString` (page 307)
Adds the characters and attributes of *attributedString* to the end of the receiver.

`applyFontTraitsInRange` (page 307)
Applies the font attributes specified by *mask* to the characters in *aRange*.

`fixAttachmentAttributeInRange` (page 308)
Cleans up attachment attributes in *aRange*, removing all attachment attributes assigned to characters other than `NSTextAttachment.AttachmentCharacter`.

`fixAttributesInRange` (page 308)
Invokes the other `fix...` methods, allowing you to clean up an attributed string with a single message.

`fixFontAttributeInRange` (page 309)
Fixes the font attribute in *aRange*, assigning default fonts to characters with illegal fonts for their scripts and otherwise correcting font attribute assignments.

`fixParagraphStyleAttributeInRange` (page 309)
Fixes the paragraph style attributes in *aRange*, assigning the first paragraph style attribute value in each paragraph to all characters of the paragraph.

`insertAttributedStringAtIndex` (page 310)
Inserts the characters and attributes of *attributedString* into the receiver, so the new characters and attributes begin at *index* and the existing characters and attributes from *index* to the end are shifted by the length of *attributedString*.

`readFromData` (page 310)

`readFromURL` (page 310)
Sets the contents of receiver from the file at *`url`*.

`replaceCharactersInRange` (page 312)
Replaces the characters and attributes in *`aRange`* with the characters and attributes of *`attributedString`*.

`setAlignmentInRange` (page 312)
Sets the alignment characteristic of the paragraph style attribute for the characters in *`aRange`* to *`alignment`*.

`setAttributedString` (page 312)
Replaces the receiver's entire contents with the characters and attributes of *`attributedString`*.

`subscriptRange` (page 313)
Decrements the value of the superscript attribute for characters in *`aRange`* by 1.

`superscriptRange` (page 313)
Increments the value of the superscript attribute for characters in *`aRange`* by 1.

`unscriptRange` (page 313)
Removes the superscript attribute from the characters in *`aRange`*.

`updateAttachmentsFromPath` (page 314)
Updates all attachments based on files contained in the RTFD file package at *`path`*.

Grouping Changes

`beginEditing` (page 307)
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 308) message, upon which it can consolidate changes and notify any observers that it has changed.

`endEditing` (page 308)
Overridden by subclasses to consolidate changes made since a previous `beginEditing` (page 307) message and to notify any observers of the changes.

Constructors

NSMutableAttributedString

Creates an empty NSMutableAttributedString.

```
public NSMutableAttributedString()
```

Creates an NSMutableAttributedString with the characters and attributes of *`attributedString`*.

```
public NSMutableAttributedString(NSAttributedString attributedString)
```

Creates an NSMutableAttributedString with the contents of *`aData`*, returning document properties in *`attributes`*.

```
public NSMutableAttributedString(NSData aData, NSMutableDictionary attributes)
```

Creates an NSMutableAttributedString from the HTML contained in *data* and base URL *aURL*.

```
public NSMutableAttributedString(NSData data, java.net.url aURL, NSMutableDictionary
    attributes)
```

Discussion
Also returns in *attributes* a dictionary containing document-level attributes described in NSAttributedString's "Constants" (page 75). Returns `null` if the file at *aURL* can't be decoded.

Creates an NSMutableAttributedString from *wrapper*, an NSFileWrapper object containing an RTFD document.

```
public NSMutableAttributedString(NSFileWrapper wrapper, NSMutableDictionary
    attributes)
```

Discussion
Also returns in *attributes* a dictionary containing document-level attributes described in NSAttributedString's "Constants" (page 75). Returns `null` if *wrapper* can't be interpreted as an RTFD document.

Creates an NSMutableAttributedString with the characters of *string*

```
public NSMutableAttributedString(String string)
```

Discussion
and no attribute information

Creates an NSMutableAttributedString with the characters of *aString* and the attributes of *attributes*.

```
public NSMutableAttributedString(String aString, NSDictionary attributes)
```

Creates an NSMutableAttributedString with the contents of *aURL*, returning document properties, which are described in NSAttributedString's "Constants" (page 75), in *attributes*.

```
public NSMutableAttributedString(java.net.URL aURL, NSMutableDictionary attributes)
```

Creates an NSMutableAttributedString with the contents of *aURL*, returning document properties, which are described in NSAttributedString's "Constants" (page 75), in *attributes*.

```
public NSMutableAttributedString(NSData aURL, NSDictionary options,
    NSMutableDictionary attributes)
```

Discussion
options can contain one of the values described in `readFromURL` (page 310).

Instance Methods

addAttributeInRange

Adds an attribute with the given *name* and *value* to the characters in *aRange*.

```
public void addAttributeInRange(String name, Object value, NSRange aRange)
```

Discussion
Throws an `InvalidArgumentException` if *`name`* or *`value`* is `null` and a `RangeException` if any part of *`aRange`* lies beyond the end of the receiver's characters.

See Also
`addAttributesInRange` (page 307)
`removeAttributeInRange` (page 311)

addAttributesInRange

Adds the collection of attributes in *`attributes`* to the characters in *`aRange`*.

```
public void addAttributesInRange(NSDictionary attributes, NSRange aRange)
```

Discussion
Throws an `InvalidArgumentException` if *`attributes`* is `null` and a `RangeException` if any part of *`aRange`* lies beyond the end of the receiver's characters.

See Also
`addAttributeInRange` (page 306)
`removeAttributeInRange` (page 311)

appendAttributedString

Adds the characters and attributes of *`attributedString`* to the end of the receiver.

```
public void appendAttributedString(NSAttributedString attributedString)
```

See Also
`insertAttributedStringAtIndex` (page 310)

applyFontTraitsInRange

Applies the font attributes specified by *`mask`* to the characters in *`aRange`*.

```
public void applyFontTraitsInRange(int mask, NSRange aRange)
```

Discussion
See the NSFontManager class specification for a description of the font traits available. Throws a `RangeException` if any part of *`aRange`* lies beyond the end of the receiver's characters.

See Also
`setAlignmentInRange` (page 312)

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 308) message, upon which it can consolidate changes and notify any observers that it has changed.

```
public void beginEditing()
```

Discussion
You can nest pairs of `beginEditing` and `endEditing` (page 308) messages.

deleteCharactersInRange

Deletes the characters in *aRange* along with their associated attributes.

```
public void deleteCharactersInRange(NSRange aRange)
```

Discussion
Throws a `RangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

See Also
`replaceCharactersInRange` (page 312)

endEditing

Overridden by subclasses to consolidate changes made since a previous `beginEditing` (page 307) message and to notify any observers of the changes.

```
public void endEditing()
```

Discussion
NSMutableAttributedString's implementation does nothing. NSTextStorage, for example, overrides this method to invoke `fixAttributesInRange` and to inform its NSLayoutManagers that they need to re-lay the text.

See Also
`processEditing` (NSTextStorage)

fixAttachmentAttributeInRange

Cleans up attachment attributes in *aRange*, removing all attachment attributes assigned to characters other than `NSTextAttachment.AttachmentCharacter`.

```
public void fixAttachmentAttributeInRange(NSRange aRange)
```

Discussion
Throws a `RangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

See Also
`fixFontAttributeInRange` (page 309)
`fixParagraphStyleAttributeInRange` (page 309)
`fixAttributesInRange` (page 308)

fixAttributesInRange

Invokes the other `fix...` methods, allowing you to clean up an attributed string with a single message.

```
public void fixAttributesInRange(NSRange aRange)
```

Discussion
Throws a `RangeException` if any part of *`aRange`* lies beyond the end of the receiver's characters.

See Also
`fixAttachmentAttributeInRange` (page 308)
`fixFontAttributeInRange` (page 309)
`fixParagraphStyleAttributeInRange` (page 309)

fixFontAttributeInRange

Fixes the font attribute in *`aRange`*, assigning default fonts to characters with illegal fonts for their scripts and otherwise correcting font attribute assignments.

```
public void fixFontAttributeInRange(NSRange aRange)
```

Discussion
For example, Kanji characters assigned a Latin font are reassigned an appropriate Kanji font. Throws a `RangeException` if any part of *`aRange`* lies beyond the end of the receiver's characters.

See Also
`fixParagraphStyleAttributeInRange` (page 309)
`fixAttachmentAttributeInRange` (page 308)
`fixAttributesInRange` (page 308)

fixParagraphStyleAttributeInRange

Fixes the paragraph style attributes in *`aRange`*, assigning the first paragraph style attribute value in each paragraph to all characters of the paragraph.

```
public void fixParagraphStyleAttributeInRange(NSRange aRange)
```

Discussion
This method extends the range as needed to cover the last paragraph partially contained. A paragraph is delimited by any of these characters, the longest possible sequence being preferred to any shorter:

U+000D (`\r` or CR)
U+000A (`\n` or LF)
U+2028 (Unicode line separator)
U+2029 (Unicode paragraph separator) `\r\n`, in that order (also known as CRLF)

Throws a `RangeException` if any part of *`aRange`* lies beyond the end of the receiver's characters.

See Also
`fixFontAttributeInRange` (page 309)
`fixAttachmentAttributeInRange` (page 308)
`fixAttributesInRange` (page 308)

insertAttributedStringAtIndex

Inserts the characters and attributes of *attributedString* into the receiver, so the new characters and attributes begin at *index* and the existing characters and attributes from *index* to the end are shifted by the length of *attributedString*.

```
public void insertAttributedStringAtIndex(NSAttributedString attributedString, int
    index)
```

Discussion
Throws a `RangeException` if *index* lies beyond the end of the receiver's characters.

See Also
`appendAttributedString` (page 307)

mutableStringReference

Returns the character contents of the receiver as an NSMutableStringReference object.

```
public NSMutableStringReference mutableStringReference()
```

Discussion
The receiver tracks changes to this string and keeps its attribute mappings up to date.

readFromData

```
public boolean readFromData(NSData data, NSDictionary options, NSMutableDictionary
    dict)
```

Discussion
Sets the contents of the receiver from the stream at *data*. *options* can contain one of the values described in `readFromURL` (page 310).

On return, the *documentAttributes* dictionary (if provided) contains the various keys described in the "Constants" (page 75) section of NSAttributedString.

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

readFromURL

Sets the contents of receiver from the file at *url*.

```
public boolean readFromURL(java.net.URL url, NSDictionary options,
    NSMutableDictionary documentAttributes)
```

Discussion
Filter services can be used to convert the contents of the URL into a format recognized by Cocoa. *options* can contain:

Key	Description
`CharacterEncoding`	For plain text documents; the unsigned int to be used if the encoding cannot be determined
`BaseURL`	For HTML documents; java.net.URL containing base URL
`DefaultAttributes`	NSDictionary containing attributes to be applied to plain files

Starting with Mac OS X v10.3, these options keys are recognized for HTML documents, both by this method as well as the NSAttributedString constructor with an *options* parameter:

Key	Description
`"UseWebKit"`	An integer. If present and positive, forces WebKit-based HTML importing be used; behavior in this case may differ from HTML import in Mac OS X v10.2 and before, particularly for tables.
`"TextEncodingName"`	String containing the name, IANA or otherwise, of a text encoding to be used if the encoding cannot be determined from the document. Mutually exclusive with `"CharacterEncoding"`.
`"Timeout"`	A float. Time in seconds to wait for a document to finish loading.
`"WebPreferences"`	WebPreferences. If WebKit-based HTML importing is used, specifies a WebPreferences object. If not present, a default set of preferences is used.
`"WebResourceLoad-Delegate"`	NSObject. If WebKit-based HTML importing is used, specifies an object to serve as the WebResourceLoadDelegate. If not present, a default delegate will be used that will permit the loading of subsidiary resources but will not respond to authentication challenges.

On return, the *documentAttributes* dictionary (if provided) contains the various keys described in the "Constants" (page 75) section of NSAttributedString.

removeAttributeInRange

Removes the attribute named *name* from the characters in *aRange*.

```
public void removeAttributeInRange(String name, NSRange aRange)
```

Discussion

Throws a `RangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

See Also

`addAttributeInRange` (page 306)
`addAttributesInRange` (page 307)

replaceCharactersInRange

Replaces the characters and attributes in *aRange* with the characters and attributes of *attributedString*.

```
public void replaceCharactersInRange(NSRange aRange, NSAttributedString
    attributedString)
```

Discussion
Throws a `RangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

Replaces the characters in *aRange* with the characters of *aString*.

```
public void replaceCharactersInRange(NSRange aRange, String aString)
```

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*.

Throws a `RangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

See Also
`insertAttributedStringAtIndex` (page 310)
`deleteCharactersInRange` (page 308)

setAlignmentInRange

Sets the alignment characteristic of the paragraph style attribute for the characters in *aRange* to *alignment*.

```
public void setAlignmentInRange(int alignment, NSRange aRange)
```

Discussion
When attribute fixing takes place, this change will affect only paragraphs whose first character was included in *aRange*. Throws a `RangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

See Also
`applyFontTraitsInRange` (page 307)
`fixParagraphStyleAttributeInRange` (page 309)

setAttributedString

Replaces the receiver's entire contents with the characters and attributes of *attributedString*.

```
public void setAttributedString(NSAttributedString attributedString)
```

See Also
`appendAttributedString` (page 307)

setAttributesInRange

Sets the attributes for the characters in *aRange* to *attributes*.

```
public void setAttributesInRange(NSDictionary attributes, NSRange aRange)
```

Discussion
These new attributes replace any attributes previously associated with the characters in *aRange*. Throws a `RangeException` 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`.

See Also
`addAttributesInRange` (page 307)
`removeAttributeInRange` (page 311)

subscriptRange

Decrements the value of the superscript attribute for characters in *aRange* by 1.

```
public void subscriptRange(NSRange aRange)
```

Discussion
Throws a `RangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

See Also
`superscriptRange` (page 313)
`unscriptRange` (page 313)

superscriptRange

Increments the value of the superscript attribute for characters in *aRange* by 1.

```
public void superscriptRange(NSRange aRange)
```

Discussion
Throws a `RangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

See Also
`subscriptRange` (page 313)
`unscriptRange` (page 313)

unscriptRange

Removes the superscript attribute from the characters in *aRange*.

```
public void unscriptRange(NSRange aRange)
```

Discussion
Throws a `RangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

See Also
`subscriptRange` (page 313)
`superscriptRange` (page 313)

updateAttachmentsFromPath

Updates all attachments based on files contained in the RTFD file package at *path*.

```
public void updateAttachmentsFromPath(String path)
```

See Also
`updateFromPath` (NSFileWrapper)

NSMutableCharacterSet

Inherits from	NSCharacterSet : NSObject
Implements	NSCoding
Package:	com.apple.cocoa.foundation
Companion guide	String Programming Guide for Cocoa

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 NSRanges 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.

Tasks

Constructors

`NSMutableCharacterSet` (page 316)
Creates an empty NSMutableCharacterSet.

Adding and Removing Characters

`addCharacter` (page 317)
Adds the character *`aChar`* to the receiver.

`removeCharacter` (page 318)
Removes the character *`aChar`* from the receiver.

`addCharactersInRange` (page 317)
Adds the characters whose integer values are given by *`aRange`* to the receiver.

`removeCharactersInRange` (page 318)
Removes from the receiver the characters whose integer values are given by *aRange*.

`addCharactersInString` (page 317)
Adds the characters in *aString* to those in the receiver.

`removeCharactersInString` (page 318)
Removes the characters in *aString* from those in the receiver.

Combining Character Sets

`intersectCharacterSet` (page 317)
Modifies the receiver so it contains only characters that exist in both the receiver and *otherSet*.

`subtractCharacterSet` (page 318)
Removes the characters in *otherSet* from those in the receiver.

`unionCharacterSet` (page 319)
Modifies the receiver so it contains all characters that exist in either the receiver or *otherSet*.

Inverting a Character Set

`invertCharacterSet` (page 318)
Replaces all the characters in the receiver with all the characters it didn't previously contain.

Constructors

NSMutableCharacterSet

Creates an empty NSMutableCharacterSet.

```
public NSMutableCharacterSet()
```

Creates a mutable character set containing characters determined by the bitmap representation *data*.

```
public NSMutableCharacterSet(NSData data)
```

Discussion
This method is useful for creating a mutable character set object with data from a file or other external data source.

Creates a mutable character set containing characters whose Unicode values are given by *aRange*.

```
public NSMutableCharacterSet(NSRange aRange)
```

Discussion
aRange.`location` is the value of the first character, and *aRange*.`location` + *aRange*.`length` – 1 is the value of the last. Returns an empty mutable character set if *aRange*.`length` is 0.

Creates a mutable character set containing the characters in *aString*.

```
public NSMutableCharacterSet(String aString)
```

Discussion
Returns an empty mutable character set if *aString* is empty.

Instance Methods

addCharacter

Adds the character *aChar* to the receiver.

```
public void addCharacter(char aChar)
```

See Also
removeCharacter (page 318)

addCharactersInRange

Adds the characters whose integer values are given by *aRange* to the receiver.

```
public void addCharactersInRange(NSRange aRange)
```

Discussion
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.

See Also
removeCharactersInRange (page 318)
addCharactersInString (page 317)

addCharactersInString

Adds the characters in *aString* to those in the receiver.

```
public void addCharactersInString(String aString)
```

Discussion
This method has no effect if *aString* is empty.

See Also
removeCharactersInString (page 318)
addCharactersInRange (page 317)

intersectCharacterSet

Modifies the receiver so it contains only characters that exist in both the receiver and *otherSet*.

```
public void intersectCharacterSet(NSCharacterSet otherSet)
```

See Also
unionCharacterSet (page 319)

invertCharacterSet

Replaces all the characters in the receiver with all the characters it didn't previously contain.

```
public void invertCharacterSet()
```

removeCharacter

Removes the character *aChar* from the receiver.

```
public void removeCharacter(char aChar)
```

See Also
addCharacter (page 317)

removeCharactersInRange

Removes from the receiver the characters whose integer values are given by *aRange*.

```
public void removeCharactersInRange(NSRange aRange)
```

Discussion
aRange.location is the value of the first character to remove, and *aRange*.location + *aRange*.length – 1 is the value of the last. If *aRange*.length is 0 this method has no effect.

See Also
addCharactersInRange (page 317)
removeCharactersInString (page 318)

removeCharactersInString

Removes the characters in *aString* from those in the receiver.

```
public void removeCharactersInString(String aString)
```

Discussion
This method has no effect if *aString* is empty.

See Also
addCharactersInString (page 317)
removeCharactersInRange (page 318)

subtractCharacterSet

Removes the characters in *otherSet* from those in the receiver.

```
public void subtractCharacterSet(NSCharacterSet otherSet)
```

unionCharacterSet

Modifies the receiver so it contains all characters that exist in either the receiver or *otherSet*.

```
public void unionCharacterSet(NSCharacterSet otherSet)
```

See Also
intersectCharacterSet (page 317)

NSMutableData

Inherits from	NSData : NSObject
Implements	NSCoding (NSData)
Package:	com.apple.cocoa.foundation
Companion guide	Binary Data Programming Guide for Cocoa

Class at a Glance

An NSMutableData object stores mutable data in the form of bytes. The size of the data is subject to a 2GB limit.

Principal Attributes

- A count of the number of bytes in the mutable data object
- The sequence of bytes contained in the mutable data object

`NSMutableData` (page 322)
Creates an NSMutableData object.

Primitive Methods

`setLength` (page 323)
Extends or truncates the number of bytes in the NSMutableData object.

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 constructor that takes an NSData or NSMutableData as an argument.

Tasks

Constructors

`NSMutableData` (page 322)

Adjusting Capacity

`increaseLengthBy` (page 323)
Increases the length of the receiver by *extraLength*.

`setLength` (page 323)
Extends or truncates a mutable data object to *length*.

Adding Data

`appendData` (page 323)
Appends the contents of a data object *otherData* to the receiver.

Modifying Data

`resetBytesInRange` (page 323)
Specifies a range within the contents of the receiver to be replaced by zeros.

`setData` (page 323)
Replaces the entire contents of the receiver with the contents of *aData*.

Constructors

NSMutableData

```
public NSMutableData()
```

Discussion
Creates an empty data object. This method is declared primarily for the use of mutable subclasses of NSData.

```
public NSMutableData(java.net.URL aURL)
```

Discussion
Creates a data object with the data from the location specified by *aURL*.

```
public NSMutableData(NSData aData)
```

Discussion
Creates a data object containing the contents of another data object, *aData*.

```
public NSMutableData(int length)
```

Discussion
Creates data object with enough memory to hold *length* bytes. Fills the object with zeros up to *length*.

Instance Methods

appendData

Appends the contents of a data object *otherData* to the receiver.

```
public void appendData(NSData otherData)
```

increaseLengthBy

Increases the length of the receiver by *extraLength*.

```
public void increaseLengthBy(int extraLength)
```

Discussion
The additional bytes are all set to 0.

See Also
setLength (page 323)

resetBytesInRange

Specifies a range within the contents of the receiver to be replaced by zeros.

```
public void resetBytesInRange(NSRange range)
```

Discussion
If the location of *range* isn't within the receiver's range of bytes, a RangeException is thrown. The receiver is resized to accommodate the new bytes, if necessary.

setData

Replaces the entire contents of the receiver with the contents of *aData*.

```
public void setData(NSData aData)
```

setLength

Extends or truncates a mutable data object to *length*.

```
public void setLength(int length)
```

Discussion
If the mutable data object is extended, the additional bytes are filled with zeros.

See Also
`increaseLengthBy` (page 323)

NSMutableDictionary

Inherits from	NSDictionary : NSObject
Implements	NSCoding (NSDictionary)
Package:	com.apple.cocoa.foundation
Companion guide	Collections Programming Topics for Cocoa

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

`NSMutableDictionary` (page 326)
Creates a new dictionary.

Commonly Used Methods

`removeObjectForKey` (page 327)
Removes the specified entry from the dictionary.

`removeObjectsForKeys` (page 327)
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—`setObjectForKey` (page 328) and `removeObjectForKey` (page 327)—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 nonprimitive methods provide convenient ways of adding or removing multiple entries at a time.

Tasks

Constructors

`NSMutableDictionary` (page 326)

Adding and Removing Entries

`addEntriesFromDictionary` (page 327)
Adds the entries from *otherDictionary* to the receiver.

`removeAllObjects` (page 327)
Empties the receiver of its entries.

`removeObjectForKey` (page 327)
Removes *aKey* and its associated value object from the receiver.

`removeObjectsForKeys` (page 327)
Removes one or more entries from the receiver.

`setDictionary` (page 328)
Sets the receiver to entries in *otherDictionary*.

`setObjectForKey` (page 328)
Adds an entry to the receiver, consisting of *aKey* and its corresponding value object *anObject*.

Constructors

NSMutableDictionary

```
public NSMutableDictionary()
```

Discussion
Creates and returns an empty mutable dictionary.

```
public NSMutableDictionary(NSDictionary otherDictionary)
```

Discussion
Creates a mutable dictionary containing the keys and values found in *otherDictionary*.

Instance Methods

addEntriesFromDictionary

Adds the entries from *otherDictionary* to the receiver.

```
public void addEntriesFromDictionary(NSDictionary otherDictionary)
```

Discussion
Each value object from *otherDictionary* is added directly to the receiver.

See Also
setObjectForKey (page 328)

removeAllObjects

Empties the receiver of its entries.

```
public void removeAllObjects()
```

See Also
removeObjectForKey (page 327)
removeObjectsForKeys (page 327)

removeObjectForKey

Removes *aKey* and its associated value object from the receiver.

```
public void removeObjectForKey(Object aKey)
```

Discussion
Does nothing if *aKey* does not exist.

See Also
removeAllObjects (page 327)
removeObjectsForKeys (page 327)

removeObjectsForKeys

Removes one or more entries from the receiver.

```
public void removeObjectsForKeys(NSArray keyArray)
```

Discussion
The entries are identified by the keys in *keyArray*. If a key in *keyArray* does not exist, the entry is ignored.

See Also
removeObjectForKey (page 327)
removeObjectForKey (page 327)

setDictionary

Sets the receiver to entries in *otherDictionary*.

```
public void setDictionary(NSDictionary otherDictionary)
```

Discussion
`setDictionary` does this by removing all entries from the receiver (with `removeAllObjects` (page 327)), then adding each entry from *otherDictionary* into the receiver.

setObjectForKey

Adds an entry to the receiver, consisting of *aKey* and its corresponding value object *anObject*.

```
public void setObjectForKey(Object anObject, Object aKey)
```

Discussion
The value object is added directly to the dictionary. Throws an `InvalidArgumentException` if the key or value object is `null`.

See Also
`removeObjectForKey` (page 327)

NSMutableIndexSet

Inherits from	NSIndexSet : NSObject
Implements	NSCoding (NSIndexSet)
Package:	com.apple.cocoa.foundation
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Collections Programming Topics for Cocoa

Overview

NSMutableIndexSet manages a mutable collection of unsigned integers. This collection is referred to as an index set and is composed of a series of indexes. A given index can appear only once in an index set. The values in an index set are always sorted, so the order in which values are added is irrelevant.

Internally, indexes are represented in ranges. Thus, an index set includes the integer members of a range or of many ranges. For maximum performance and efficiency, overlapping ranges in an index set are automatically coalesced (ranges merge rather than overlap).

NSMutableIndexSet is not intended to be subclassed.

Tasks

Constructors

`NSMutableIndexSet` (page 330)
Creates and returns an NSMutableIndexSet containing the indexes specified by `NSRange.ZeroRange`.

Adding Indexes

`addIndex` (page 331)
Adds the index specified by *`value`* to the receiver.

`addIndexes` (page 331)

`addIndexesInRange` (page 331)
Adds the indexes specified by *`range`* to the receiver.

Removing Indexes

`removeIndex` (page 332)
Removes the index specified by *value* from the receiver.

`removeIndexes` (page 332)
Removes the indexes contained in *indexSet* from the receiver.

`removeAllIndexes` (page 332)
Removes all the indexes from the receiver.

`removeIndexesInRange` (page 332)
Removes the indexes specified by *range* from the receiver.

Shifting Indexes in an Index Set

`shiftIndexes` (page 333)

Constructors

NSMutableIndexSet

Creates and returns an NSMutableIndexSet containing the indexes specified by `NSRange.ZeroRange`.

```
public NSMutableIndexSet()
```

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

Creates and returns an NSMutableIndexSet containing a single index, *value*.

```
public NSMutableIndexSet(int value)
```

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

Creates and returns an NSMutableIndexSet containing the indexes specified by *range*.

```
public NSMutableIndexSet(NSRange range)
```

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

Creates and returns an NSMutableIndexSet containing the indexes in *indexSet*.

```
public NSMutableIndexSet(NSIndexSet indexSet)
```

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

Instance Methods

addIndex

Adds the index specified by *value* to the receiver.

```
public void addIndex(int value)
```

Discussion
This method throws a `RangeException` if the addition of *value* to the index set would exceed the maximum range allowed by NSIndexSet.

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

See Also
`addIndexes` (page 331)
`addIndexesInRange` (page 331)

addIndexes

```
public void addIndexes(NSIndexSet indexSet)
```

Discussion
Adds the indexes specified by *indexSet* to the receiver.

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

See Also
`addIndex` (page 331)
`addIndexesInRange` (page 331)

addIndexesInRange

Adds the indexes specified by *range* to the receiver.

```
public void addIndexesInRange(NSRange range)
```

Discussion
This method throws a `RangeException` if the addition of the indexes specified by *range* would exceed the maximum range allowed by NSIndexSet.

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

See Also
`addIndex` (page 331)
`addIndexes` (page 331)

removeAllIndexes

Removes all the indexes from the receiver.

```
public void removeAllIndexes()
```

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

See Also
removeIndex (page 332)
removeIndexes (page 332)
removeIndexesInRange (page 332)

removeIndex

Removes the index specified by *value* from the receiver.

```
public void removeIndex(int value)
```

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

See Also
removeAllIndexes (page 332)
removeIndexes (page 332)
removeIndexesInRange (page 332)

removeIndexes

Removes the indexes contained in *indexSet* from the receiver.

```
public void removeIndexes(NSIndexSet indexSet)
```

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

See Also
removeIndex (page 332)
removeAllIndexes (page 332)
removeIndexesInRange (page 332)

removeIndexesInRange

Removes the indexes specified by *range* from the receiver.

```
public void removeIndexesInRange(NSRange range)
```

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

See Also
removeIndex (page 332)
removeIndexes (page 332)
removeAllIndexes (page 332)

shiftIndexes

```
public void shiftIndexes(int index, int delta)
```

Discussion
For a positive *delta*, shifts the indexes in [*index*, INT_MAX] to the right, thereby inserting an "empty space" in the range [*index*, *delta*]. For a negative *delta*, shifts the indexes in [*index*, INT_MAX] to the left, thereby deleting the indexes in the range [*index* - *delta*, *delta*].

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

NSMutablePoint

Inherits from	NSPoint : Object
Package:	com.apple.cocoa.foundation
Companion guide	Geometry and Range Utilities

Overview

An NSMutablePoint is an object representing a point that can be changed in a coordinate system. The main purpose for NSMutablePoints is to provide a way for methods to return coordinate values in an "out" parameter. The client creates and passes in one or more NSMutablePoints to a method and gets back changed objects when the method returns. NSMutablePoints are also useful for performance reasons; instead of creating multiple NSPoints in a loop, you can create just one NSMutablePoint and reuse it.

Tasks

Constructors

`NSMutablePoint` (page 336)

Accessing and Setting Coordinate Values

`setX` (page 336)
Sets the x coordinate of the receiver to *`newX`*.

`setY` (page 337)
Sets the y coordinate of the receiver to *`newY`*.

`x` (page 337)
Returns the x coordinate of the receiver.

`y` (page 337)
Returns the y coordinate of the receiver.

Copying

`clone` (page 336)
Creates and returns a copy of the receiver.

Constructors

NSMutablePoint

```
public NSMutablePoint()
```

Discussion
This constructor initializes the *x* and *y* coordinates to 0.

```
public NSMutablePoint(float x, float y)
```

Discussion
Initializes the NSMutablePoint with the horizontal coordinate *x* and the vertical coordinate *y*.

```
public NSMutablePoint(NSPoint aPoint)
```

Discussion
Initializes the new NSMutablePoint with the coordinate values of NSPoint *aPoint*; this constructor is used in cloning the receiver.

```
public NSMutablePoint(java.awt.Point javaPoint)
```

Discussion
Initializes the NSMutablePoint with the values extracted from an AWT Point object, *javaPoint*.

Instance Methods

clone

Creates and returns a copy of the receiver.

```
public Object clone()
```

setX

Sets the x coordinate of the receiver to *newX*.

```
public void setX(float newX)
```

Discussion
Throws an `IllegalArgumentException` if *newX* is NaN (that is, not a valid float value).

See Also
x (page 337)

setY

Sets the y coordinate of the receiver to *newY*.

```
public void setY(float newY)
```

Discussion
Throws an `IllegalArgumentException` if *newY* is NaN (that is, not a valid float value).

See Also
y (page 337)

x

Returns the x coordinate of the receiver.

```
public float x()
```

Discussion
NSMutablePoint overrides this method because implementation details make overriding necessary.

See Also
setX (page 336)

y

Returns the y coordinate of the receiver.

```
public float y()
```

Discussion
NSMutablePoint overrides this method because implementation details make overriding necessary.

See Also
setY (page 337)

NSMutableRange

Inherits from	NSRange : Object
Package:	com.apple.cocoa.foundation
Companion guide	Geometry and Range Utilities

Overview

An NSMutableRange is an object representing a range that can be changed. A range is a measurement of a segment of something linear, such as a byte stream. You can change an NSMutableRange's two primary values, its location and its length. The methods of NSMutableRange also enable you to alter an NSMutableRange based on its union or intersection with another NSRange object.

The main purpose for NSMutableRanges is to provide a way for methods to return range values in an "out" parameter. A client creates and passes in one or more NSMutableRanges to a method and gets back changed objects when the method returns. NSMutableRanges are also useful for performance reasons; instead of creating multiple NSRanges in a loop, you can create just one NSMutableRange and reuse it.

Tasks

Constructors

`NSMutableRange` (page 340)

Accessing and Setting Range Elements

`length` (page 341)
: Returns the length of the receiver, its distance from its starting location.

`setLength` (page 341)
: Sets the length of the receiver to *`newLength`*.

`location` (page 341)
: Returns the starting location of the receiver.

`setLocation` (page 341)
: Sets the length of the receiver to *`newLocation`*.

Transforming Mutable Ranges

`clone` (page 340)
Creates and returns a copy of the receiver.

`intersectRange` (page 340)
Changes the receiver to the range resulting from the intersection of *aRange* and the receiver before the operation.

`unionRange` (page 342)
Changes the receiver to the range resulting from the union of *aRange* and the receiver.

Constructors

NSMutableRange

```
public NSMutableRange()
```

Discussion
Initializes the object to an empty NSMutableRange.

```
public NSMutableRange(int location, int length)
```

Discussion
Initializes the NSMutableRange with the range elements of *location* and *length*. Throws an `IllegalArgumentException` if either integer is negative.

```
public NSMutableRange(NSRange aRange)
```

Discussion
Initializes the new NSMutableRange with the location and length values of *aRange*; this constructor is used in cloning the receiver.

Instance Methods

clone

Creates and returns a copy of the receiver.

```
public Object clone()
```

intersectRange

Changes the receiver to the range resulting from the intersection of *aRange* and the receiver before the operation.

```
public void intersectRange(NSRange aRange)
```

Discussion
Sets the receiver to an empty range if they do not intersect.

See Also
`unionRange` (page 342)

length

Returns the length of the receiver, its distance from its starting location.

```
public int length()
```

Discussion
NSMutableRange overrides this method because of internal implementation requirements.

See Also
`location` (page 341)

location

Returns the starting location of the receiver.

```
public int location()
```

Discussion
NSMutableRange overrides this method because of internal implementation requirements.

See Also
`length` (page 341)

setLength

Sets the length of the receiver to *`newLength`*.

```
public void setLength(int newLength)
```

Discussion
Throws an `IllegalArgumentException` if *`newLength`* is a negative value.

See Also
`setLocation` (page 341)

setLocation

Sets the length of the receiver to *`newLocation`*.

```
public void setLocation(int newLocation)
```

Discussion
Throws an `IllegalArgumentException` if *`newLocation`* is a negative value.

See Also
setLength (page 341)

unionRange

Changes the receiver to the range resulting from the union of *aRange* and the receiver.

```
public void unionRange(NSRange aRange)
```

Discussion
The result is a range with the lowest starting location and the highest ending location of the two NSRanges.

See Also
intersectRange (page 340)

NSMutableRect

Inherits from	NSRect : Object
Package:	com.apple.cocoa.foundation
Companion guide	Geometry and Range Utilities

Overview

An NSMutableRect is an object representing a rectangle that can be modified. The elemental attributes of a rectangle are its origin (its starting x coordinate and y coordinate) and its size (its width and height as measured from the origin). The methods of NSMutableRect allow you to change these elemental values. They also let you inset and offset rectangles by specific amounts and alter an NSMutableRect based on its union or intersection with another NSRect object.

The main purpose for NSMutableRects is to provide a way for methods to return rectangle values in an "out" parameter. A client creates and passes in one or more NSMutableRects to a method and gets back converted objects when the method returns. NSMutableRects are also useful for performance reasons; instead of creating multiple NSRects in a loop, you can create just one NSMutablePoint and reuse it.

Tasks

Constructors

`NSMutableRect` (page 344)

Accessing and Setting Coordinate Values

`setOrigin` (page 347)
: Sets the origin point of the receiver to *`newOrigin`*.

`x` (page 348)
: Returns the origin x coordinate of the receiver.

`y` (page 349)
: Returns the origin y coordinate of the receiver.

`setX` (page 347)
: Sets the x-coordinate of the receiver to *`newX`*.

`setY` (page 348)
Sets the y-coordinate of the receiver to *newY*.

Accessing and Setting Size Values

`height` (page 345)
Returns the height dimension of the receiver.

`setHeight` (page 347)
Sets the width of the receiver to *newHeight*.

`setSize` (page 347)
Sets the size of the receiver to *newSize*.

`width` (page 348)
Returns the width of the receiver.

`setWidth` (page 347)
Sets the width of the receiver to *newWidth*.

Transforming Mutable Rectangles

`insetRect` (page 346)
Modifies the receiver to be inset from both upper and lower edges by *vertDistance* and from both left and right edges by *horizDistance*.

`intersectRect` (page 346)
Modifies the receiver to be the intersection of itself and *otherRectangle*.

`makeIntegral` (page 346)
Changes the receiver so that its origin and size are rounded to the nearest integer, ensuring that the receiver completely contains the original rectangle.

`offsetRect` (page 346)
Changes the receiver so that its x coordinate is moved by *horizOffset* and its y coordinate is moved by *vertOffset*.

`unionRect` (page 348)
Modifies the receiver to be the union of itself and *otherRectangle*.

Copying

`clone` (page 345)
Creates and returns a copy of the receiver.

Constructors

NSMutableRect

```
public NSMutableRect()
```

Discussion
Initializes an empty mutable rectangle (that is, a rectangle with at least one dimension of 0).

```
public NSMutableRect(float x, float y, float w, float h)
```

Discussion
Initializes an NSMutableRect from a starting x coordinate (*x*), a starting y coordinate (*y*), a width value (*w*), and a height value (*h*). If either width and height is 0, it initializes an empty rectangle.

```
public NSMutableRect(NSPoint aPoint, NSSize aSize)
```

Discussion
Initializes an NSMutableRect from an NSPoint object, *aPoint*, and an NSSize object, *aSize*.

```
public NSMutableRect(NSPoint pointOne, NSPoint pointTwo)
```

Discussion
Initializes an NSMutableRect from two NSPoint objects, *pointOne* and *pointTwo*. Creates the smallest rectangle containing the two points.

```
public NSMutableRect(java.awt.Rectangle javaRectangle)
```

Discussion
Initializes an NSMutableRect from an AWT Rectangle object, *javaRectangle*.

```
public NSMutableRect(NSRect aRectangle)
```

Discussion
Initializes an NSMutableRect from an NSRect object, *aRectangle*; this constructor is used in cloning the receiver.

Instance Methods

clone

Creates and returns a copy of the receiver.

```
public Object clone()
```

height

Returns the height dimension of the receiver.

```
public float height()
```

Discussion
NSMutableRect overrides this method because implementation details make overriding necessary.

See Also
`setHeight` (page 347)
`width` (page 348)

insetRect

Modifies the receiver to be inset from both upper and lower edges by *vertDistance* and from both left and right edges by *horizDistance*.

```
public void insetRect(float vertDistance, float horizDistance)
```

Discussion
The values *vertDistance* and *horizDistance* can be negative. An `IllegalArgumentException` is thrown if the resulting width or height would be negative.

See Also
`offsetRect` (page 346)

intersectRect

Modifies the receiver to be the intersection of itself and *otherRectangle*.

```
public void intersectRect(NSRect otherRectangle)
```

Discussion
If either the receiver or *otherRectangle* has an empty dimension, it modifies the receiver to be an empty rectangle (all dimensions) at point {0.0f, 0.0f}.

See Also
`unionRect` (page 348)

makeIntegral

Changes the receiver so that its origin and size are rounded to the nearest integer, ensuring that the receiver completely contains the original rectangle.

```
public void makeIntegral()
```

Discussion
The x coordinate and the y coordinate are rounded down, and the height and width are rounded up. If the receiver has an empty dimension, it is modified to be an empty rectangle (all dimensions) at point {0.0f, 0.0f}.

offsetRect

Changes the receiver so that its x coordinate is moved by *horizOffset* and its y coordinate is moved by *vertOffset*.

```
public void offsetRect(float horizOffset, float vertOffset)
```

Discussion
Both arguments can be negative values.

See Also
`insetRect` (page 346)

setHeight

Sets the width of the receiver to *newHeight*.

```
public void setHeight(float newHeight)
```

Discussion
Throws an `IllegalArgumentException` if *newHeight* is NaN (an invalid float value) or is a negative value.

See Also
`height` (page 345)
`setWidth` (page 347)

setOrigin

Sets the origin point of the receiver to *newOrigin*.

```
public void setOrigin(NSPoint newOrigin)
```

See Also
`origin` (page 487) (NSRect)

setSize

Sets the size of the receiver to *newSize*.

```
public void setSize(NSSize newSize)
```

See Also
`size` (page 489) (NSRect)

setWidth

Sets the width of the receiver to *newWidth*.

```
public void setWidth(float newWidth)
```

Discussion
Throws an `IllegalArgumentException` if *newWidth* is NaN (an invalid float value) or is a negative value.

See Also
`setHeight` (page 347)
`width` (page 348)

setX

Sets the x-coordinate of the receiver to *newX*.

```
public void setX(float newX)
```

Discussion
Throws an `IllegalArgumentException` if *newX* is NaN (an invalid float value) or is a negative value.

See Also
`setY` (page 348)
`x` (page 348)

setY

Sets the y-coordinate of the receiver to *newY*.

```
public void setY(float newY)
```

Discussion
Throws an `IllegalArgumentException` if *newY* is NaN (an invalid float value) or is a negative value.

See Also
`setX` (page 347)
`y` (page 349)

unionRect

Modifies the receiver to be the union of itself and *otherRectangle*.

```
public void unionRect(NSRect otherRectangle)
```

Discussion
If the receiver and *otherRectangle* both have an empty dimension, it modifies the receiver to be an empty rectangle (all dimensions) at point {0.0f, 0.0f}. If *otherRectangle* has an empty dimension, but the receiver doesn't, the receiver is unchanged.

See Also
`intersectRect` (page 346)

width

Returns the width of the receiver.

```
public float width()
```

Discussion
NSMutableRect overrides this method because implementation details make overriding necessary.

See Also
`height` (page 345)

x

Returns the origin x coordinate of the receiver.

```
public float x()
```

Discussion
NSMutableRect overrides this method because implementation details make overriding necessary.

See Also
setX (page 347)
y (page 349)

y

Returns the origin y coordinate of the receiver.

```
public float y()
```

Discussion
NSMutablePoint overrides this method because implementation details make overriding necessary.

See Also
x (page 348)
setY (page 348)

NSMutableSet

Inherits from	NSSet : NSObject
Implements	NSCoding (NSSet)
Package:	com.apple.cocoa.foundation
Companion guide	Collections Programming Topics for Cocoa

Class at a Glance

An NSMutableSet object stores a modifiable set of objects.

Principal Attributes

- The objects that make up the set

`NSMutableSet` (page 353)
Creates a new set.

Commonly Used Methods

`addObject` (page 353)
Adds an object to the set, if it isn't already a member.

`removeObject` (page 354)
Removes an object from the set.

Primitive Methods

`addObject` (page 353)
Adds an object to the set, if it isn't already a member.

`removeObject` (page 354)
Removes an object from the set.

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 NSSet class supports creating and managing immutable sets.

Objects are added to an NSMutableSet with `addObject` (page 353), which adds a single object to the set; `addObjectsFromArray` (page 353), which adds all objects from a specified array to the set; or `unionSet` (page 355), which adds all the objects from another set.

Objects are removed from an NSMutableSet using any of the methods `intersectSet` (page 353), `removeAllObjects` (page 354), `removeObject` (page 354), or `subtractSet` (page 354).

Tasks

Constructors

`NSMutableSet` (page 353)

Adding and Removing Entries

`addObject` (page 353)
Adds the specified object to the receiver if it is not already a member.

`removeObject` (page 354)
Removes *anObject* from the receiver.

`removeAllObjects` (page 354)
Empties the receiver of all of its members.

`addObjectsFromArray` (page 353)
Adds each object contained in *anArray* to the receiver, if that object is not already a member.

Combining and Recombining Sets

`unionSet` (page 355)
Adds each object contained in *otherSet* to the receiver, if that object is not already a member.

`subtractSet` (page 354)
Removes from the receiver each object contained in *otherSet* that is also present in the receiver.

`intersectSet` (page 353)
Removes from the receiver each object that isn't a member of *otherSet*.

`setSet` (page 354)
Empties the receiver, then adds each object contained in *otherSet* to the receiver.

Constructors

NSMutableSet

```
public NSMutableSet()
```

Discussion
Returns an empty mutable set.

```
public NSMutableSet(NSSet aSet)
```

Discussion
Returns a mutable set containing those objects contained within the set *aSet*.

Instance Methods

addObject

Adds the specified object to the receiver if it is not already a member.

```
public void addObject(Object anObject)
```

Discussion
If *anObject* is already present in the set, this method has no effect on either the set or *anObject*.

See Also
addObjectsFromArray (page 353)
unionSet (page 355)

addObjectsFromArray

Adds each object contained in *anArray* to the receiver, if that object is not already a member.

```
public void addObjectsFromArray(NSArray anArray)
```

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.

See Also
addObject (page 353)
unionSet (page 355)

intersectSet

Removes from the receiver each object that isn't a member of *otherSet*.

```
public void intersectSet(NSSet otherSet)
```

See Also
removeObject (page 354)
removeAllObjects (page 354)
subtractSet (page 354)

removeAllObjects

Empties the receiver of all of its members.

```
public void removeAllObjects()
```

See Also
removeObject (page 354)
subtractSet (page 354)
intersectSet (page 353)

removeObject

Removes *anObject* from the receiver.

```
public void removeObject(Object anObject)
```

See Also
removeObject (page 354)
subtractSet (page 354)
intersectSet (page 353)

setSet

Empties the receiver, then adds each object contained in *otherSet* to the receiver.

```
public void setSet(NSSet otherSet)
```

subtractSet

Removes from the receiver each object contained in *otherSet* that is also present in the receiver.

```
public void subtractSet(NSSet otherSet)
```

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.

See Also
removeObject (page 354)
removeAllObjects (page 354)
intersectSet (page 353)

unionSet

Adds each object contained in *otherSet* to the receiver, if that object is not already a member.

```
public void unionSet(NSSet otherSet)
```

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.

See Also
addObject (page 353)
addObjectsFromArray (page 353)

NSMutableSize

Inherits from	NSSize : Object
Package:	com.apple.cocoa.foundation
Companion guide	Geometry and Range Utilities

Overview

An NSMutableSize is an object representing a dimension that can be changed. The main purpose for NSMutableSizes is to provide a way for methods to return size values in an "out" parameter. The client creates and passes in one or more NSMutableSizes to a method and gets back changed objects when the method returns. NSMutableSizes are also useful for performance reasons; instead of creating multiple NSSizes in a loop, you can create just one NSMutableSize and reuse it.

Tasks

Constructors

`NSMutableSize` (page 358)

Accessing and Setting Dimensions

`height` (page 358)
Returns the height dimension of the receiver.

`setHeight` (page 359)
Sets the height dimension of the receiver to *`newHeight`*.

`width` (page 359)
Returns the width of the receiver.

`setWidth` (page 359)
Sets the width dimension of the receiver to *`newWidth`*.

Copying

`clone` (page 358)
Creates and returns a copy of the receiver.

Constructors

NSMutableSize

```
NSMutableSize()
```

Discussion
Initializes an “empty” NSMutableSize (one whose height or width is 0).

```
NSMutableSize(float w, float h)
```

Discussion
Initializes the NSMutableSize with the width dimension *w* and the height dimension *y*; it throws an `IllegalArgumentException` if either value is negative.

```
NSMutableSize(NSSize aSize)
```

Discussion
Initializes the new NSMutableSize with the width and height values of an existing NSSize *aSize*; this constructor is used in cloning the receiver.

```
NSMutableSize(java.awt.Dimension dimension)
```

Discussion
Initializes an NSMutableSize with the values extracted from an AWT Dimension object, *dimension*.

Instance Methods

clone

Creates and returns a copy of the receiver.

```
public Object clone()
```

height

Returns the height dimension of the receiver.

```
public float height()
```

Discussion
NSMutableSize overrides this method because implementation details make overriding necessary

See Also
`setHeight` (page 359)
`width` (page 359)

setHeight

Sets the height dimension of the receiver to *`newHeight`*.

```
public void setHeight(float newHeight)
```

Discussion
Throws an `IllegalArgumentException` if *`newHeight`* is negative or is NaN (an invalid float value).

See Also
`height` (page 358)

setWidth

Sets the width dimension of the receiver to *`newWidth`*.

```
public void setWidth(float newWidth)
```

Discussion
Throws an `IllegalArgumentException` if *`newWidth`* is negative or is NaN (an invalid float value).

See Also
`width` (page 359)

width

Returns the width of the receiver.

```
public float width()
```

Discussion
NSMutableSize overrides this method because implementation details make overriding necessary.

See Also
`height` (page 358)
`setWidth` (page 359)

NSMutableStringReference

Inherits from	NSStringReference : NSObject
Package:	com.apple.cocoa.foundation
Companion guide	String Programming Guide for Cocoa

Overview

The NSMutableStringReference class declares the programmatic interface to an object that manages a mutable string—that is, a string whose contents can be edited. To construct and manage an immutable string—or a string that cannot be changed after it has been created—use an object of the NSStringReference class.

An immutable string is implemented as an array of Unicode characters (in other words, as a text string). The NSMutableStringReference class adds one primitive method—`replaceCharactersInRange` (page 363)—to the basic string-handling behavior inherited from NSStringReference. All other methods that modify a string work through this method. For example, `insertStringAtIndex` (page 363) simply replaces the characters in a range of 0 length, while `deleteCharactersInRange` (page 363) replaces the characters in a given range with no characters.

The Application Kit also uses NSParagraphStyle and its subclass NSMutableParagraphStyle to encapsulate the paragraph or ruler attributes used by the NSAttributedString classes.

Tasks

Constructors

`NSMutableStringReference` (page 362)
Creates an empty NSMutableStringReference.

Modifying a String

`appendString` (page 362)
Adds the characters of *aString* to the end of the receiver.

`deleteCharactersInRange` (page 363)
Removes the characters in *aRange* from the receiver.

`insertStringAtIndex` (page 363)
Inserts the characters of *aString* into the receiver, so the new characters begin at *anIndex* and the existing characters from *anIndex* to the end are shifted by the length of *aString*.

`replaceCharactersInRange` (page 363)
Replaces the characters from *aRange* with those in *aString*.

`replaceOccurrencesOfString` (page 364)
This method replaces all occurrences of *target* with *replacement*, in the specified *searchRange* of the receiver.

`setString` (page 364)
Replaces the characters of the receiver with those in *aString*.

Constructors

NSMutableStringReference

Creates an empty NSMutableStringReference.

```
public NSMutableStringReference()
```

Creates a new NSMutableStringReference by converting the bytes in *aData* into Unicode characters.

```
public NSMutableStringReference(NSData aData, int encoding)
```

Discussion
aData must be 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.

Creates a new NSMutableStringReference by reading characters from the location named by *aURL*.

```
public NSMutableStringReference(java.net.URL aURL)
```

Discussion
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 characters in the default C-string encoding. Returns `null` if the location can't be opened.

Creates a new NSMutableStringReference by converting the bytes at *aURL* into Unicode characters.

```
public NSMutableStringReference(java.net.URL aURL, int encoding)
```

Discussion
aURL must contain bytes in *encoding* and the default plain text format (that is, pure content with no attributes or other markups) for that encoding.

Instance Methods

appendString

Adds the characters of *aString* to the end of the receiver.

```
public void appendString(String aString)
```

Discussion
aString may not be `null`.

Adds the characters of *aStringReference* to the end of the receiver.

```
public void appendString(NSStringReference aStringReference)
```

Discussion
aStringReference may not be `null`.

deleteCharactersInRange

Removes the characters in *aRange* from the receiver.

```
public void deleteCharactersInRange(NSRange aRange)
```

Discussion
Throws a `RangeException` if any part of *aRange* lies beyond the end of the string. This method treats the length of the string as a valid range value that returns an empty string.

insertStringAtIndex

Inserts the characters of *aString* into the receiver, so the new characters begin at *anIndex* and the existing characters from *anIndex* to the end are shifted by the length of *aString*.

```
public void insertStringAtIndex(String aString, int anIndex)
```

Discussion
aString may not be `null`. Throws a `RangeException` if *anIndex* lies beyond the end of the string. This method treats the length of the string as a valid index value that returns an empty string.

Inserts the characters of *aStringReference* into the receiver, so the new characters begin at *anIndex* and the existing characters from *anIndex* to the end are shifted by the length of *aStringReference*.

```
public void insertStringAtIndex(NSStringReference aStringReference, int anIndex)
```

Discussion
aStringReference may not be `null`. Throws a `RangeException` if *anIndex* lies beyond the end of the string.

replaceCharactersInRange

Replaces the characters from *aRange* with those in *aString*.

```
public void replaceCharactersInRange(NSRange aRange, String aString)
```

Discussion
aString may not be `null`. Throws a `RangeException` if any part of *aRange* lies beyond the end of the receiver. This method treats the length of the string as a valid range value that returns an empty string.

Replaces the characters from *aRange* with those in *aStringReference*.

```
public void replaceCharactersInRange(NSRange aRange, NSStringReference
    aStringReference)
```

Discussion
aStringReference may not be `null`. Throws a `RangeException` if any part of *aRange* lies beyond the end of the receiver. This method treats the length of the string as a valid range value that returns an empty string.

replaceOccurrencesOfString

This method replaces all occurrences of *target* with *replacement*, in the specified *searchRange* of the receiver.

```
public void replaceOccurrencesOfString(String target, String replacement, int opts,
     NSRange searchRange)
```

Discussion
Throws an `InvalidArgumentException` if any of the arguments are `null`. Throws a `RangeException` if any part of *searchRange* lies beyond the end of the receiver. This method treats the length of the string as a valid range value that returns an empty string. If *opts* is `BackwardsSearch`, the search is done from the end of the range. If *opts* is `AnchoredSearch`, only anchored (but potentially multiple) instances are replaced. `LiteralSearch` and `CaseInsensitiveSearch` also apply. Specify *searchRange* as `new NSRange(0, receiver.length())` to process the entire string.

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

setString

Replaces the characters of the receiver with those in *aString*.

```
public void setString(String aString)
```

Discussion
aString may not be `null`.

Replaces the characters of the receiver with those in *aStringReference*.

```
public void setString(NSStringReference aStringReference)
```

Discussion
aStringReference may not be `null`.

NSNamedValueSequence

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Key-Value Coding Programming Guide

Overview

NSNamedValueSequence manages a set of keys, allowing you to assign scalar and object values to them.

The class is similar to an NSMutableDictionary, but NSNamedValueSequence is created with a fixed number of elements, and it defines convenient methods such as `getFloatWithName` (page 367) and `setFloatWithName` (page 368). If you request a value for an undefined key, a default value of 0 or `null` is returned. After the object reaches its capacity of keys, attempts to define additional keys fail. NSNamedValueSequence objects cannot be resized, nor can keys be deleted.

Tasks

Constructors

`NSNamedValueSequence` (page 366)
Creates an empty NSNamedValueSequence object with the capacity *`size`*.

Getting Values

`getBooleanWithName` (page 367)
Returns the `boolean` value associated with *`key`*.

`getByteWithName` (page 367)
Returns the `byte` value associated with *`key`*.

`getCharWithName` (page 367)
Returns the `char` value associated with *`key`*.

`getDoubleWithName` (page 367)
Returns the `double` value associated with *`key`*.

`getFloatWithName` (page 367)
Returns the `float` value associated with *`key`*.

`getIntWithName` (page 367)
Returns the `int` value associated with *`key`*.

`getLongWithName` (page 367)
Returns the `long` value associated with *`key`*.

`getObjectWithName` (page 368)
Returns the `Object` value associated with *`key`*.

`getShortWithName` (page 368)
Returns the `short` value associated with *`key`*.

Setting Values

`setBooleanWithName` (page 368)
Sets the value of *`key`* to *`value`*.

`setByteWithName` (page 368)
Sets the value of *`key`* to *`value`*.

`setCharWithName` (page 368)
Sets the value of *`key`* to *`value`*.

`setDoubleWithName` (page 368)
Sets the value of *`key`* to *`value`*.

`setFloatWithName` (page 368)
Sets the value of *`key`* to *`value`*.

`setIntWithName` (page 368)
Sets the value of *`key`* to *`value`*.

`setLongWithName` (page 369)
Sets the value of *`key`* to *`value`*.

`setObjectWithName` (page 369)
Sets the value of *`key`* to *`value`*.

`setShortWithName` (page 369)
Sets the value of *`key`* to *`value`*.

Constructors

NSNamedValueSequence

Creates an empty NSNamedValueSequence object with the capacity *`size`*.

```
public NSNamedValueSequence(int size)
```

Discussion
An instance cannot be resized after being created, so make sure you allocate enough space for all the keys you expect to use.

Instance Methods

getBooleanWithName

Returns the `boolean` value associated with *key*.

```
public boolean getBooleanWithName(String key)
```

getByteWithName

Returns the `byte` value associated with *key*.

```
public byte getByteWithName(String key)
```

getCharWithName

Returns the `char` value associated with *key*.

```
public char getCharWithName(String key)
```

getDoubleWithName

Returns the `double` value associated with *key*.

```
public double getDoubleWithName(String key)
```

getFloatWithName

Returns the `float` value associated with *key*.

```
public float getFloatWithName(String key)
```

getIntWithName

Returns the `int` value associated with *key*.

```
public int getIntWithName(String key)
```

getLongWithName

Returns the `long` value associated with *key*.

```
public long getLongWithName(String key)
```

getObjectWithName

Returns the `Object` value associated with *key*.

```
public Object getObjectWithName(String key)
```

getShortWithName

Returns the `short` value associated with *key*.

```
public short getShortWithName(String key)
```

setBooleanWithName

Sets the value of *key* to *value*.

```
public void setBooleanWithName(boolean value, String key)
```

setByteWithName

Sets the value of *key* to *value*.

```
public void setByteWithName(byte value, String key)
```

setCharWithName

Sets the value of *key* to *value*.

```
public void setCharWithName(char value, String key)
```

setDoubleWithName

Sets the value of *key* to *value*.

```
public void setDoubleWithName(double value, String key)
```

setFloatWithName

Sets the value of *key* to *value*.

```
public void setFloatWithName(float value, String key)
```

setIntWithName

Sets the value of *key* to *value*.

```
public void setIntWithName(int value, String key)
```

setLongWithName

Sets the value of *key* to *value*.

```
public void setLongWithName(long value, String key)
```

setObjectWithName

Sets the value of *key* to *value*.

```
public void setObjectWithName(Object value, String key)
```

setShortWithName

Sets the value of *key* to *value*.

```
public void setShortWithName(short value, String key)
```

NSNameSpecifier

Inherits from	NSScriptObjectSpecifier : NSObject
Package:	com.apple.cocoa.foundation
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 NSNameSpecifiers 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 NSWhoseSpecifier that specifies the first object whose relevant `'pnam'` attribute matches the name is synthesized and evaluated. The NSWhoseSpecifier must search through all of the keyed elements in the container, looking for a match. The search is potentially very slow.

Tasks

Constructors

`NSNameSpecifier` (page 372)
Creates an NSNameSpecifier with no data.

Accessing a Name Specifier

`name` (page 373)
Returns the name encapsulated by the receiver for the specified object in the container.

`setName` (page 373)
Sets the name encapsulated with the receiver for the specified object in the container.

Constructors

NSNameSpecifier

Creates an NSNameSpecifier with no data.

```
public NSNameSpecifier()
```

Discussion
Do not use this constructor.

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

Returns a newly created unnamed NSNameSpecifier with container specifier *container* and key *property*.

```
public NSNameSpecifier(NSScriptObjectSpecifier container, String property)
```

Discussion
The class description of container is set automatically. Use `setName` (page 373) to assign a name to the returned object.

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

Creates an unnamed NSNameSpecifier initialized with container specifier *container*, key *property*, and the class description of the object specifier *classDescription*, derived from the value of the specifier's key.

```
public NSNameSpecifier(NSScriptClassDescription classDescription,
    NSScriptObjectSpecifier container, String property)
```

Discussion
The receiver's child specifier reference is set to `null`. Use `setName` (page 373) to assign a name to the returned object.

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

Creates an NSNameSpecifier named *name* initialized with container specifier *container*, key *property*, and the class description of the object specifier *classDescription*, derived from the value of the specifier's key.

```
public NSNameSpecifier(NSScriptClassDescription classDescription,
    NSScriptObjectSpecifier container, String property, String name)
```

Discussion
The receiver's child specifier reference is set to `null`.

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

Instance Methods

name

Returns the name encapsulated by the receiver for the specified object in the container.

```
public String name()
```

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

See Also
`setName` (page 373)

setName

Sets the name encapsulated with the receiver for the specified object in the container.

```
public void setName(String name)
```

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

See Also
`name` (page 373)

NSNetService

Inherits from	NSObject
Implements	NSObject
Package:	com.apple.cocoa.foundation
Availability	Available in Mac OS X v10.4 and later.
Companion guide	Bonjour Overview

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 380) 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 381) method to verify that the service is available and ready for your application. If it is, the `addresses` (page 378) 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

Constructors

`NSNetService` (page 378)
Creates an NSNetService object as a network service of the specified *type* at the socket location specified by *domain*, *name*, and *port*.

Managing Delegates

`delegate` (page 379)
Returns the receiver's delegate.

`setDelegate` (page 382)
Sets the receiver's delegate.

Maintaining Run Loops

`removeFromRunLoop` (page 380)
Removes the service from the specified run loop.

`scheduleInRunLoop` (page 381)
Adds the service to the specified run loop.

Getting Information About a Service

`addresses` (page 378)
Returns an NSArray containing NSData objects, each of which contains a socket address for the service.

`domain` (page 379)
Returns the domain name of the service.

`hostName` (page 379)
Returns the host name of the computer providing the service.

`name` (page 380)
Returns the name of the service.

`type` (page 383)
Returns the type of the service.

`protocolSpecificInformation` (page 380)
This method has been deprecated. Use `TXTRecordData` (page 383) instead.

`setProtocolSpecificInformation` (page 382)
This method has been deprecated. Use `setTXTRecordData` (page 382) instead.

`setTXTRecordData` (page 382)
Sets the TXT record for the receiver.

`TXTRecordData` (page 383)
Returns the TXT record for the receiver.

Working with a Service

`publish` (page 380)
Attempts to advertise the receiver's service on the network.

`resolve` (page 381)
This method has been deprecated. Use `resolveWithTimeout` (page 381) instead.

`resolveWithTimeout` (page 381)
Starts a resolve process of a finite duration for the receiver.

`stop` (page 383)
Halts a currently running attempt to publish or resolve a service.

Availability notifications

`netServiceDidNotPublish` (page 384) *delegate method*
Notifies the delegate that the service offered by *`sender`* could not be published.

`netServiceDidPublish` (page 384) *delegate method*
Notifies the delegate that the service offered by *`sender`* was successfully published.

`netServiceWillPublish` (page 385) *delegate method*
Notifies the delegate that the network is ready to publish the service.

Resolving services

`netServiceDidNotResolve` (page 384) *delegate method*
Informs the delegate that an error occurred during resolution of *`sender`*.

`netServiceDidResolveAddress` (page 384) *delegate method*
Informs the delegate that the address for *`sender`* was resolved.

`netServiceDidUpdateTXTRecordData` (page 385) *delegate method*
Notifies the delegate that the TXT record for *`sender`* has been updated.

`netServiceWillResolve` (page 385) *delegate method*
Notifies the delegate that the network is ready to resolve the service.

Stopping services

`netServiceDidStop` (page 385) *delegate method*
Informs the delegate that a `publish` (page 380) or `resolveWithTimeout` (page 381) request was stopped.

Constructors

NSNetService

```
public NSNetService()
```

Discussion
The default constructor for the NSNetService object.

```
public com.apple.cocoa.foundation.NSNetService(String domain, String type, String
    name)
```

Discussion
Creates an NSNetService object for a network service of the specified *type* and sets the initial *domain* and service *name*. After using this constructor, you can call `resolve` (page 381) to resolve the service location.

You cannot use this constructor to publish a service. The constructor passes an invalid port number, which prevents the service from being registered.

Creates an NSNetService object as a network service of the specified *type* at the socket location specified by *domain*, *name*, and *port*.

```
public NSNetService(java.lang.String domain, java.lang.String type, java.lang.String
    name, int port)
```

Discussion
You can use this constructor to create a service you wish to publish on the network.

When publishing a service, you must provide valid arguments to advertise your service correctly. The *name* parameter identifies your service to the network and must be unique. The *port* parameter must contain a port number acquired by your application for the service.

It is preferable to use a NSNetServiceBrowser object to obtain the local registration domain in which to publish your service. To use the default domain, simply pass an empty string to the domain parameter. 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.

The *type* parameter must contain both the service type and transport layer information. To ensure that the mDNS repsonder 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 pass the string "_http._tcp." to the *type* parameter. Note that the period character at the end of the string is required. It indicates that the domain name is an absolute name.

Instance Methods

addresses

Returns an NSArray containing NSData objects, each of which contains a socket address for the service.

```
public NSArray addresses()
```

Discussion
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.

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. If no addresses were resolved for the service, the returned NSArray contains zero elements.

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

See Also
`resolve` (page 381)

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

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

See Also
`setDelegate` (page 382)

domain

Returns the domain name of the service.

```
public String domain()
```

Discussion
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 v10.4 and later.

hostName

Returns the host name of the computer providing the service.

```
public String hostName()
```

Discussion
Returns `null` if a successful resolve has not occurred.

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

name

Returns the name of the service.

```
public String name()
```

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

protocolSpecificInformation

This method has been deprecated. Use `TXTRecordData` (page 383) instead.

```
public String protocolSpecificInformation()
```

Discussion
Returns any protocol-specific data associated with the service.

This method is provided for legacy support of older zeroconf-style clients and its use is discouraged.

Availability
Deprecated in Mac OS X v10.4.

See Also
`setProtocolSpecificInformation` (page 382)

publish

Attempts to advertise the receiver's service on the network.

```
public void publish()
```

Discussion
This method returns immediately, with success or failure indicated by the callbacks to the delegate.

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

See Also
`stop` (page 383)

removeFromRunLoop

Removes the service from the specified run loop.

```
public void removeFromRunLoop(NSRunLoop aRunLoop, String mode)
```

Discussion
You can use this method in conjunction with `scheduleInRunLoop` (page 381) to transfer the service to a different run loop. Although it is possible to remove an NSNetService completely from any run loop and then attempt actions on it, it is an error to do so.

Possible values for *mode* are discussed in the "Constants" (page 503) section of NSRunLoop.

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

See Also
`scheduleInRunLoop` (page 381)

resolve

This method has been deprecated. Use `resolveWithTimeout` (page 381) instead.

```
public 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 381) with a timeout value of 5.

Availability
Deprecated in Mac OS X v10.4.

See Also
`addresses` (page 378)
`stop` (page 383)
`resolveWithTimeout` (page 381)

resolveWithTimeout

Starts a resolve process of a finite duration for the receiver.

```
public void resolveWithTimeout(double timeout)
```

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

See Also
`addresses` (page 378)
`stop` (page 383)

scheduleInRunLoop

Adds the service to the specified run loop.

```
public void scheduleInRunLoop(NSRunLoop aRunLoop, String mode)
```

Discussion
You can use this method in conjunction with `removeFromRunLoop` (page 380) to transfer the service to a different run loop. You should not attempt to run the service on multiple run loops.

Possible values for *mode* are discussed in the "Constants" (page 503) section of NSRunLoop.

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

See Also
`removeFromRunLoop` (page 380)

setDelegate

Sets the receiver's delegate.

```
public void setDelegate(Object delegate)
```

Discussion
The delegate is not retained.

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

See Also
`delegate` (page 379)

setProtocolSpecificInformation

This method has been deprecated. Use `setTXTRecordData` (page 382) instead.

```
public void setProtocolSpecificInformation(String specificInformation)
```

Discussion
Attaches protocol-specific data to the service.

This method retains the string in *specificInformation* and releases the previous string. This method is provided for legacy support of older zeroconf-style clients and its use is discouraged.

Availability
Deprecated in Mac OS X v10.4.

See Also
`protocolSpecificInformation` (page 380)

setTXTRecordData

Sets the TXT record for the receiver.

```
public boolean setTXTRecordData(NSData recordData)
```

Discussion
Returns `true` when *recordData* is successfully set as the TXT record; otherwise, it returns `false`.

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

See Also
TXTRecordData (page 383)

stop

Halts a currently running attempt to publish or resolve a service.

```
public void stop()
```

Discussion
This method results in the sending of a netServiceDidStop (page 385) message to the delegate.

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

TXTRecordData

Returns the TXT record for the receiver.

```
public NSData TXTRecordData()
```

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

See Also
setTXTRecordData (page 382)

type

Returns the type of the service.

```
public String type()
```

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

Constants

When an error occurs, the delegate error-handling methods return a dictionary with the following keys.

Constant	Description
NSNetServicesErrorCode	This key identifies the error that occurred during the most recent operation.
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.

Delegate Methods

netServiceDidNotPublish

Notifies the delegate that the service offered by *sender* could not be published.

```
public abstract void netServiceDidNotPublish(NSNetService sender, NSDictionary
    errorDict)
```

Discussion
This method may be called long after a `netServiceWillPublish` (page 385) message has been delivered to the delegate. You can use the dictionary keys `NSNetServicesErrorCode` and `NSNetServicesErrorDomain` to retrieve the error information from *errorDict*.

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

netServiceDidNotResolve

Informs the delegate that an error occurred during resolution of *sender*.

```
public abstract void netServiceDidNotResolve(NSNetService sender, NSDictionary
    errorDict)
```

Discussion
You can use the dictionary keys `NSNetServicesErrorCode` and `NSNetServicesErrorDomain` to retrieve the error information from *errorDict*.

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 v10.4 and later.

netServiceDidPublish

Notifies the delegate that the service offered by *sender* was successfully published.

```
public abstract void netServiceDidPublish(NSNetService sender
```

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

netServiceDidResolveAddress

Informs the delegate that the address for *sender* was resolved.

```
public abstract void netServiceDidResolveAddress(NSNetService sender)
```

Discussion
The delegate can use the `addresses` (page 378) method to retrieve the service's address.

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

See Also
`addresses` (page 378)

netServiceDidStop

Informs the delegate that a `publish` (page 380) or `resolveWithTimeout` (page 381) request was stopped.

```
public abstract void netServiceDidStop(NSNetService sender)
```

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

See Also
`stop` (page 383)

netServiceDidUpdateTXTRecordData

Notifies the delegate that the TXT record for *sender* has been updated.

```
public abstract void netServiceDidUpdateTXTRecordData(NSNetService sender, NSData
    data)
```

Discussion
The *data* parameter contains the new TXT record.

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

netServiceWillPublish

Notifies the delegate that the network is ready to publish the service.

```
public abstract void netServiceWillPublish(NSNetService sender)
```

Discussion
Publication of the service proceeds asynchronously and may still generate a call to the delegate's `netServiceDidNotPublish` (page 384) method if an error occurs.

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

netServiceWillResolve

Notifies the delegate that the network is ready to resolve the service.

```
public abstract void netServiceWillResolve(NSNetService sender)
```

Discussion
Resolution of the service proceeds asynchronously and may still generate a call to the delegate's `netServiceDidNotResolve` (page 384) method if an error occurs.

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

NSNetServiceBrowser

Inherits from	NSObject
Implements	NSObject
Package:	com.apple.cocoa.foundation
Availability	Availabile in Mac OS X v10.4 and later.
Companion guide	Bonjour Overview

Overview

The NSNetServiceBrowser class defines an interface for finding published services on a network using multicast DNS. Services can range from standard services, such as HTTP and FTP, to custom services defined by other applications. You can use a NSNetServiceBrowser object in your code to obtain the list of accessible domains and then to obtain an instance of the NSNetService class for each discovered service. Each NSNetServiceBrowser object performs one search at a time, so if you want to perform multiple simultaneous searches, use multiple NSNetServiceBrowser objects.

A NSNetServiceBrowser object 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 sent a `stop` (page 392) message.

To use a NSNetServiceBrowser object to search for services, you allocate it, initialize it, and assign a delegate. (If you wish, you can also use the `scheduleInRunLoop` (page 390) and `removeFromRunLoop` (page 390) 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 391) or `searchForBrowsableDomains` (page 391) methods. From the list of returned domains, you can pick one and use the `searchForServicesOfType` (page 391) method to search for services in that domain.

NSNetServiceBrowser provides two ways to search for domains. In most cases, your client should use the `searchForRegistrationDomains` (page 391) 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

Constructors

`NSNetServiceBrowser` (page 389)

Managing Delegates

`delegate` (page 389)
Returns the receiver's delegate.

`setDelegate` (page 392)
Sets the receiver's delegate.

Maintaining Run Loops

`removeFromRunLoop` (page 390)
Removes the NSNetServiceBrowser from the specified run loop.

`scheduleInRunLoop` (page 390)
Adds the NSNetServiceBrowser to the specified run loop.

Working with a Service Browser

`searchForAllDomains` (page 390)
This method has been deprecated. Use `searchForBrowsableDomains` (page 391) or `searchForRegistrationDomains` (page 391).

`searchForBrowsableDomains` (page 391)
Initiates a search for domains visible to the host. This method returns immediately.

`searchForRegistrationDomains` (page 391)
Initiates a search for domains in which the host may register services.

`searchForServicesOfType` (page 391)
Starts a search for services of type *`type`* within a specific domain, *`domainString`*.

`stop` (page 392)
Halts a currently running search or resolution.

Searching

`netServiceBrowser` (page 393) *delegate method*
Sent by *`aNetServiceBrowser`* whenever an error prevented a search from occurring.

`netServiceBrowserDidStopSearch` (page 393) *delegate method*
Informs the delegate that the current search was stopped.

`netServiceBrowserWillSearch` (page 394) *delegate method*
Sent by *aNetServiceBrowser* to indicate a search is commencing.

Working with domains

`netServiceBrowserDidFindDomain` (page 393) *delegate method*
Sent by *aNetServiceBrowser* each time it finds a domain.

`netServiceDidRemoveDomain` (page 394) *delegate method*
Sent by *aNetServiceBrowser* whenever a domain disappears or becomes unavailable.

Working with services

`netServiceDidFindService` (page 394) *delegate method*
Sent by *aNetServiceBrowser* each time it finds a service.

`netServiceDidRemoveService` (page 395) *delegate method*
Sent by *aNetServiceBrowser* whenever a service disappears or becomes unavailable.

Constructors

NSNetServiceBrowser

```
public NSNetServiceBrowser()
```

Discussion
The constructor for the NSNetServiceBrowser object.

Instance Methods

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

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

See Also
`setDelegate` (page 392)

removeFromRunLoop

Removes the NSNetServiceBrowser from the specified run loop.

```
public void removeFromRunLoop(NSRunLoop aRunLoop, String mode)
```

Discussion
You can use this method in conjunction with `scheduleInRunLoop` (page 390) to transfer the service to a run loop other than the default one. Although it is possible to remove an NSNetService completely from any run loop and then attempt actions on it, it is an error to do so.

Possible values for *mode* are discussed in the "Constants" (page 503) section of NSRunLoop.

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

See Also
`scheduleInRunLoop` (page 390)

scheduleInRunLoop

Adds the NSNetServiceBrowser to the specified run loop.

```
public void scheduleInRunLoop(NSRunLoop aRunLoop, String mode)
```

Discussion
You can use this method in conjunction with `removeFromRunLoop` (page 390) to transfer the service to a run loop other than the default one. You should not attempt to run the service on multiple run loops.

Possible values for *mode* are discussed in the "Constants" (page 503) section of NSRunLoop.

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

See Also
`removeFromRunLoop` (page 390)

searchForAllDomains

This method has been deprecated. Use `searchForBrowsableDomains` (page 391) or `searchForRegistrationDomains` (page 391).

```
public void searchForAllDomains()
```

Discussion
Initiates a search for all domains that are visible to the host.

This method returns immediately, sending a `netServiceBrowserWillSearch` (page 394) message to the delegate if the network was ready to initiate the search. The delegate receives a subsequent `netServiceBrowserDidFindDomain` (page 393) 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 391)
`netServiceBrowserDidFindDomain` (page 393)

searchForBrowsableDomains

Initiates a search for domains visible to the host. This method returns immediately.

```
public void searchForBrowsableDomains()
```

Discussion
The delegate receives a `netServiceBrowserDidFindDomain` (page 393) message for each domain discovered.

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

See Also
`searchForRegistrationDomains` (page 391)

searchForRegistrationDomains

Initiates a search for domains in which the host may register services.

```
public void searchForRegistrationDomains()
```

Discussion
This method returns immediately, sending a `netServiceBrowserWillSearch` (page 394) message to the delegate if the network was ready to initiate the search. The delegate receives a subsequent `netServiceBrowserDidFindDomain` (page 393) message for each domain discovered.

Most NSNetService clients do not have to use this API—it is sufficient to publish your NSNetService with the empty string which will register it in any available registration domains automatically.

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

See Also
`searchForBrowsableDomains` (page 391)
`searchForServicesOfType` (page 391)
`netServiceBrowserDidFindDomain` (page 393)
`netServiceBrowserWillSearch` (page 394)

searchForServicesOfType

Starts a search for services of type *type* within a specific domain, *domainString*.

```
public void searchForServicesOfType(String type, String domainString)
```

Discussion
This method returns immediately, sending a `netServiceBrowserWillSearch` (page 394) message to the delegate if the network was ready to initiate the search.The delegate receives subsequent `netServiceDidFindService` (page 394) messages for each service discovered.

The *type* 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 *domainString* 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 domain. Usually, you pass in an empty string. Note that it is acceptable to use an empty string for the *domainString* argument when publishing or browsing a service, but do not rely on this for resolution.

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

See Also
`netServiceDidFindService` (page 394)
`netServiceBrowserWillSearch` (page 394)

setDelegate

Sets the receiver's delegate.

```
public void setDelegate(Object delegate)
```

Discussion
The delegate is not retained. You must specify a delegate. The NSNetServiceBrowser calls the methods of your delegate to receive information about discovered domains and services.

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

See Also
`delegate` (page 389)

stop

Halts a currently running search or resolution.

```
public void stop()
```

Discussion
Invoking this method sends a `netServiceBrowserDidStopSearch` (page 393) message to the delegate and causes the browser to discard any pending search results.

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

See Also
`netServiceBrowserDidStopSearch` (page 393)

Delegate Methods

netServiceBrowser

Sent by *aNetServiceBrowser* whenever an error prevented a search from occurring.

```
public abstract void netServiceBrowser(NSNetServiceBrowser aNetServiceBrowser,
    NSDictionary errorDict)
```

Discussion
You can use the dictionary keys `NSNetServicesErrorCode` and `NSNetServicesErrorDomain` to retrieve the error information from the dictionary.

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

See Also
`netServiceBrowserWillSearch` (page 394)

netServiceBrowserDidFindDomain

Sent by *aNetServiceBrowser* each time it finds a domain.

```
puublic abstract void netServiceBrowserDidFindDomain(NSNetServiceBrowser
    aNetServiceBrowser, String domainString, boolean moreComing)
```

Discussion
Use this message to accumulate the list of domain names. If *moreComing* is `true`, the browser is waiting to return additional domains. If your client displays a list of domains to the user, you should wait until this parameter is `false`, and then do a bulk update of your user interface elements.

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

See Also
`searchForBrowsableDomains` (page 391)
`searchForRegistrationDomains` (page 391)

netServiceBrowserDidStopSearch

Informs the delegate that the current search was stopped.

```
public abstract void netServiceBrowserDidStopSearch(NSNetServiceBrowser
    aNetServiceBrowser)
```

Discussion
The NSNetServiceBrowser object in *aNetServiceBrowser* sends this message to your delegate in response to receiving a `stop` (page 392) message from the browser client. Use this message to perform any necessary cleanup.

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

See Also
`stop` (page 392)

netServiceBrowserWillSearch

Sent by *aNetServiceBrowser* to indicate a search is commencing.

```
public abstract void netServiceBrowserWillSearch(NSNetServiceBrowser
    aNetServiceBrowser)
```

Discussion
This message is sent to the delegate only if the underlying network layer was in a state to begin a search. Your delegate can use this notification to prepare its data structures to receive data.

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

See Also
`netServiceBrowser` (page 393)

netServiceDidFindService

Sent by *aNetServiceBrowser* each time it finds a service.

```
public abstract void netServiceDidFindService(NSNetServiceBrowser aNetServiceBrowser,
     NSNetService aNetService,
```

Discussion
The *aNetService* argument contains the service that was discovered. You can use this object to connect to and start using the service. If *moreComing* is `true`, the browser is waiting to return additional service objects. If your client displays a list of services to the user, you should wait until this parameter is `false` and then do a bulk update of your user interface elements.

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

See Also
`searchForServicesOfType` (page 391)

netServiceDidRemoveDomain

Sent by *aNetServiceBrowser* whenever a domain disappears or becomes unavailable.

```
public abstract void netServiceDidRemoveDomain(NSNetServiceBrowser
    aNetServiceBrowser, String domainString, boolean moreComing)
```

Discussion
Use this message to update the list of available domains. If *moreComing* is `true`, the browser has additional domain names for your delegate to remove. If your client displays a list of the current domains to the user, you should wait until this parameter is `false`, and then do a bulk update of your user interface elements.

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

netServiceDidRemoveService

Sent by *aNetServiceBrowser* whenever a service disappears or becomes unavailable.

```
public abstract void netServiceDidRemoveService(NSNetServiceBrowser
    aNetServiceBrowser, NSNetService aNetService, boolean moreComing)
```

Discussion
Use this message to update the list of available services. If *moreComing* is `true`, the browser has additional services for your delegate to remove. If your client displays a list of the current services to the user, you should wait until this parameter is `false`, and then do a bulk update of your user interface elements.

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

NSNotification

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.foundation
Companion guide	Notification Programming Topics for Cocoa

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 constructor. However, you don't usually create your own notifications directly. The NSNotificationCenter method `postNotification` (page 403) allows you to conveniently post a notification without creating it first.

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.

Tasks

Constructors

`NSNotification` (page 398)
Throws an `IllegalArgumentException`.

Obtaining Information About a Notification

`name` (page 398)
Returns the name of the receiver.

`object` (page 399)
Returns the object associated with the receiver.

`userInfo` (page 399)
Returns the NSDictionary associated with the receiver or `null` if there is no such object.

Constructors

NSNotification

Throws an `IllegalArgumentException`.

```
public NSNotification()
```

Discussion
Use one of the other constructors to create an instance.

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

Creates a notification object that associates the name *`aName`* with the object *`anObject`*.

```
public NSNotification(String aName, Object anObject)
```

Discussion
`aName` may not be `null`.

Creates a notification object that associates the name *`aName`* with the object *`anObject`* and the dictionary of arbitrary data *`userInfo`*.

```
public NSNotification(String aName, Object anObject, NSDictionary userInfo)
```

Discussion
The dictionary *`userInfo`* may be `null`. *`aName`* may not be `null`.

See Also
`postNotification` (page 403) (NSNotificationCenter)

Instance Methods

name

Returns the name of the receiver.

```
public String name()
```

Discussion
Examples of this might be “PortIsInvalid” or “PhoneRinging.” Typically, you invoke this method on the notification object passed to your notification-handler method. (You specify a notification-handler method when you register to receive the notification.)

Notification names can be any string. To avoid name collisions, however, you might want to use a prefix that's specific to your application.

object

Returns the object associated with the receiver.

```
public Object object()
```

Discussion
This is often the object that posted this notification. It may be `null`.

Typically, you invoke this method on the notification object passed in to your notification-handler method. (You specify a notification-handler method when you register to receive the notification.)

userInfo

Returns the NSDictionary associated with the receiver or `null` if there is no such object.

```
public NSDictionary userInfo()
```

Discussion
The NSDictionary stores any additional objects that objects receiving the notification might use. For example, in the Application Kit, NSControl objects post the `ControlTextDidChangeNotification` whenever the field editor (an NSText object) changes text inside the NSControl. This notification provides both the NSControl object and the field editor to objects registered to receive them. The field editor is returned when you access the dictionary.

NSNotificationCenter

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Notification Programming Topics for Cocoa

Class at a Glance

NSNotificationCenter provides a way for objects that don't know about each other to communicate. It receives NSNotification objects and broadcasts them to all interested objects.

Principal Attributes

- A table containing objects that want to receive notifications, the notifications they want to receive, and the objects they want to receive these notifications from.

Each task has a default notification center. You typically don't create your own.

Commonly Used Methods

`defaultCenter` (page 403)
: Accesses the default notification center.

`addObserver` (page 403)
: Registers an object to receive a notification.

`postNotification` (page 403)
: Posts a notification.

`removeObserver` (page 404)
: Specifies that an object no longer wants to receive notifications.

Overview

An NSNotificationCenter object (or simply, notification center) is essentially a notification dispatch table. It notifies all observers of notifications meeting specific criteria. This information is encapsulated in NSNotification objects, also known as notifications. Client objects register themselves with the notification center as observers of specific notifications posted by other objects. When an event occurs, an object posts an appropriate

notification to the notification center. The notification center dispatches a message to each registered observer, 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.

An NSNotificationCenter object delivers notifications to observers synchronously. In other words the `postNotification` (page 403) method does not return until all observers have received and processed the notification. To send notifications asynchronously use NSNotificationQueue (page 405). 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.

An NSNotificationCenter object can only deliver notifications within a single task. If you want to post a notification to other tasks or receive notifications from other tasks, use NSDistributedNotificationCenter (page 159).

Tasks

Constructors

`NSNotificationCenter` (page 403)
Creates an empty NSNotificationCenter.

Accessing the Default Center

`defaultCenter` (page 403)
Returns the current task's notification center, which is used for system notifications.

Adding and Removing Observers

`addObserver` (page 403)
Registers *`anObserver`* to receive notifications with the name *`notificationName`* and/or containing *`anObject`*.

`removeObserver` (page 404)
Removes *`anObserver`* from all notification associations in the receiver.

Posting Notifications

`postNotification` (page 403)
Posts *`notification`* to the receiver.

Constructors

NSNotificationCenter

Creates an empty NSNotificationCenter.

```
public NSNotificationCenter()
```

Discussion
This center is not the default notification center. To obtain the default center, use `defaultCenter` (page 403).

Static Methods

defaultCenter

Returns the current task's notification center, which is used for system notifications.

```
public static NSNotificationCenter defaultCenter()
```

Instance Methods

addObserver

Registers *anObserver* to receive notifications with the name *notificationName* and/or containing *anObject*.

```
public void addObserver(Object anObserver, NSSelector aSelector, String
    notificationName, Object anObject)
```

Discussion
When a notification of name *notificationName* containing the object *anObject* is posted, *anObserver* receives an *aSelector* message with this notification as the argument. The method for the selector specified in *aSelector* must have one and only one argument. If *notificationName* is `null`, the notification center notifies the observer of all notifications with an object matching *anObject*. (The objects are matched by comparing their pointers.) If *anObject* is `null`, the notification center notifies the observer of all notifications with the name *notificationName*. *anObserver* may not be `null`.

postNotification

Posts *notification* to the receiver.

```
public void postNotification(NSNotification notification)
```

Discussion
You can create *notification* with the NSNotification constructor. An exception is thrown if *notification* is `null`.

Creates a notification with the name *notificationName*, associates it with the object *anObject*, and posts it to the notification center.

```
public void postNotification(String notificationName, Object anObject)
```

Discussion
anObject is typically the object posting the notification. It may be `null`.

Creates a notification with the name *notificationName*, associates it with the object *anObject* and dictionary *userInfo*, and posts it to the notification center.

```
public void postNotification(String notificationName, Object anObject, NSDictionary
     userInfo)
```

Discussion
This method is the preferred method for posting notifications. *anObject* is typically the object posting the notification. It may be `null`. *userInfo* also may be `null`.

removeObserver

Removes *anObserver* from all notification associations in the receiver.

```
public void removeObserver(Object anObserver)
```

Discussion
anObserver may not be `null`.

Removes *anObserver* as the observer of notifications with the name *notificationName* and object *anObject* from the receiver.

```
public void removeObserver(Object anObserver, String notificationName, Object
    anObject)
```

Discussion
anObserver may not be `null`.

If *notificationName* is `null`, *anObserver* is removed as an observer of all notifications containing *anObject*. If *anObject* is `null`, *anObserver* is removed as an observer of *notificationName* containing any object. For example, if you wanted to unregister `someObserver` from all notifications it had previously registered for, you would sent this message:

```
NSNotificationCenter.defaultCenter().removeObserver(someObserver,  null, null);
```

NSNotificationQueue

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Notification Programming Topics for Cocoa

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

Constructors

`NSNotificationQueue` (page 406)
Creates an NSNotificationQueue that uses the application's default notification center to post notifications.

Creating and Initializing Notification Queues

`defaultQueue` (page 406)
Returns the default NSNotificationQueue object for the current thread.

Inserting and Removing Notifications from a Queue

`dequeueMatchingNotifications` (page 406)
Removes all notifications from the receiver that match the attributes of *`notification`* as specified by *`coalesceMask`*.

`enqueueNotification` (page 407)
Puts *notification* in the receiver.

`enqueueNotificationWithCoalesceMaskForModes` (page 407)
Puts *notification* in the receiver.

Constructors

NSNotificationQueue

Creates an NSNotificationQueue that uses the application's default notification center to post notifications.

```
public NSNotificationQueue()
```

Creates an NSNotificationQueue that uses the notification center specified in *notificationCenter* to post notifications.

```
public NSNotificationQueue(NSNotificationCenter notificationCenter)
```

Static Methods

defaultQueue

Returns the default NSNotificationQueue object for the current thread.

```
public static NSNotificationQueue defaultQueue()
```

Discussion
This object always uses the default notification center object for the same task.

Instance Methods

dequeueMatchingNotifications

Removes all notifications from the receiver that match the attributes of *notification* as specified by *coalesceMask*.

```
public void dequeueMatchingNotifications(NSNotification notification, int
    coalesceMask)
```

Discussion
The mask is created by combining the constants `NotificationNoCoalescing`, `NotificationCoalescingOnName`, and `NotificationCoalescingOnSender`.

enqueueNotification

Puts *notification* in the receiver.

```
public void enqueueNotification(NSNotification notification, int postingStyle)
```

Discussion
The queue posts *notification* to the notification center at the time indicated by *postingStyle*. The notification queue posts in `NSRunLoop.DefaultRunLoopMode`, and it coalesces only notifications in the queue that match both the notification's name and object.

This method invokes `enqueueNotificationWithCoalesceMaskForModes` (page 407).

enqueueNotificationWithCoalesceMaskForModes

Puts *notification* in the receiver.

```
public void enqueueNotificationWithCoalesceMaskForModes(NSNotification notification,
     int postingStyle, int coalesceMask, NSArray modes)
```

Discussion
The queue posts *notification* to the notification center at the time indicated by *postingStyle*, but only if the run loop is in a mode identified by one of the string objects in the *modes* array. The notification queue coalesces related notifications in the queue as specified by *coalesceMask* (set using the constants `NotificationNoCoalescing`, `NotificationCoalescingOnName`, and `NotificationCoalescingOnSender`). If *modes* is `null`, the notification queue posts in `NSRunLoop.DefaultRunLoopMode`.

Constants

These constants specify how notifications are coalesced. They're used in the third argument of `enqueueNotificationWithCoalesceMaskForModes` (page 407). You can OR them together to specify more than one.

Constant	Description
`NotificationNoCoalescing`	Do not coalesce notifications in the queue.
`NotificationCoalescingOnName`	Coalesce notifications with the same name.
`NotificationCoalescingOnSender`	Coalesce notifications with the same object.

These constants specify when notifications are posted. They're used in both `enqueueNotification` (page 407) and `enqueueNotificationWithCoalesceMaskForModes` (page 407):

Constant	Description
`PostASAP`	The notification is posted at the end of the current notification callout or timer.
`PostWhenIdle`	The notification is posted when the run loop is idle.

Constant	Description
`PostNow`	The notification is posted immediately after coalescing.

NSNull

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.foundation
Companion guide	Number and Value Programming Topics for Cocoa

Overview

The NSNull class defines a unique object used to represent null values in collection objects (which don't allow `null` values).

Tasks

Constructors

`NSNull` (page 409)
Returns the unique instance of NSNull.

Obtaining an Instance

`nullValue` (page 410)
Returns the unique instance of NSNull.

Constructors

NSNull

Returns the unique instance of NSNull.

```
public NSNull()
```

Static Methods

nullValue

Returns the unique instance of NSNull.

```
public static NSNull nullValue()
```

NSNumberFormatter

Inherits from	NSFormatter : NSObject
Implements	NSCoding (NSNumberFormatter)
Package:	com.apple.cocoa.foundation
Companion guide	Data Formatting Programming Guide for Cocoa

Overview

Instances of NSNumberFormatter format the textual representation of cells that contain Numbers and convert textual representations of numeric values into Numbers. The representation encompasses integers, floats, and doubles; floats and doubles can be formatted to a specified decimal position. NSNumberFormatters can also impose ranges on the numeric values cells can accept.

Tasks

Constructors

`NSNumberFormatter` (page 413)
Creates an NSNumberFormatter with a format of "#,##0.00;0.00;–#,##0.00".

Setting and Getting Formats

`negativeFormat` (page 416)
Returns a String containing the format used by the receiver to display negative numbers.

`setNegativeFormat` (page 419)
Sets the format the receiver uses to display negative values to *aFormat*.

`positiveFormat` (page 416)
Returns a String containing the format used by the receiver to display positive numbers.

`setPositiveFormat` (page 419)
Sets the format the receiver uses to display positive values to *aFormat*.

`format` (page 415)
Returns a String containing the format being used by the receiver.

`setFormat` (page 418)
Sets the receiver's format to the string *aFormat*.

Setting and Getting Characteristics for Displaying Values

`textAttributesForNegativeValues` (page 421)
Returns an NSDictionary containing the text attributes that have been set for negative values.

`setTextAttributesForNegativeValues` (page 420)
Sets the text attributes to be used in displaying negative values to *newAttributes*.

`textAttributesForPositiveValues` (page 421)
Returns an NSDictionary containing the text attributes that have been set for positive values.

`setTextAttributesForPositiveValues` (page 420)
Sets the text attributes to be used in displaying positive values to *newAttributes*.

`attributedStringForZero` (page 415)
Returns the NSAttributedString used to display zero values.

`setAttributedStringForZero` (page 418)
Sets the NSAttributedString the receiver uses to display zero values to *newAttributedString*.

`attributedStringForNil` (page 414)
Returns the NSAttributedString used to display `null` values.

`setAttributedStringForNil` (page 417)
Sets the NSAttributedString the receiver uses to display `null` values to *newAttributedString*.

`attributedStringForNotANumber` (page 414)
Returns the NSAttributedString used to display “not a number” values.

`setAttributedStringForNotANumber` (page 417)
Sets the NSAttributedString the receiver uses to display “not a number” values to *newAttributedString*.

`attributedStringForObjectValue` (page 414)
Return an NSAttributedString if the string for display should have some attributes.

Setting and Getting Separators

`hasThousandSeparators` (page 415)
Returns `true` to indicate that the receiver’s format includes thousand separators, `false` otherwise.

`setHasThousandSeparators` (page 419)
Sets according to *flag* whether the receiver uses thousand separators.

`thousandSeparator` (page 421)
Returns a String containing the character the receiver uses to represent thousand separators.

`setThousandSeparator` (page 420)
Sets the character the receiver uses as a thousand separator to *newSeparator*.

`decimalSeparator` (page 415)
Returns a String containing the character the receiver uses to represent decimal separators.

`setDecimalSeparator` (page 418)
Sets the character the receiver uses as a decimal separator to *newSeparator*. If *newSeparator* contains multiple characters, only the first one is used. If you don’t have decimal separators enabled through another means (such as `setFormat` (page 418)), using this method enables them.

Enabling Localization

`localizesFormat` (page 416)
: Returns `true` to indicate that the receiver localizes formats, `false` otherwise.

`setLocalizesFormat` (page 419)
: Sets according to *`flag`* 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.

Setting and Getting Float Behavior

`allowsFloats` (page 414)
: Returns `true` if the receiver allows as input floating-point values (that is, values that include the period character [`.`]), `false` otherwise.

`setAllowsFloats` (page 417)
: Sets according to *`flag`* whether the receiver allows as input floating-point values (that is, values that include the period character [`.`]).

String Manipulation

`isPartialStringValid` (page 416)
: Since this method is invoked each time the user presses a key while the cursor is in the cell, it lets you verify the cell text as the user types it.

`objectValueForString` (page 416)
: Returns an object created from *`aString`*.

`replacementStringForString` (page 417)
: Checks whether *`aString`* is a valid string for the cell.

`stringForObjectValue` (page 420)
: Returns the string that textually represents the cell's object for display and for editing.

Constructors

NSNumberFormatter

Creates an NSNumberFormatter with a format of "#,##0.00;0.00;–#,##0.00".

```
public NSNumberFormatter()
```

See Also
`setFormat` (page 418)

Instance Methods

allowsFloats

Returns `true` if the receiver allows as input floating-point values (that is, values that include the period character [`.`]), `false` otherwise.

```
public boolean allowsFloats()
```

Discussion
When this method returns `false`, only integer values can be provided as input. The default is `true`.

See Also
`setAllowsFloats` (page 417)

attributedStringForNil

Returns the NSAttributedString used to display `null` values.

```
public NSAttributedString attributedStringForNil()
```

Discussion
By default `null` values are displayed as an empty string.

See Also
`setAttributedStringForNil` (page 417)

attributedStringForNotANumber

Returns the NSAttributedString used to display “not a number” values.

```
public NSAttributedString attributedStringForNotANumber()
```

Discussion
By default “not a number” values are displayed as the string “NaN”.

See Also
`setAttributedStringForNotANumber` (page 417)

attributedStringForObjectValue

Return an NSAttributedString if the string for display should have some attributes.

```
public NSAttributedString attributedStringForObjectValue(Object anObject,
    NSDictionary attributes)
```

Discussion
For instance, you might want negative values in a financial application to appear in red text. Invoke your implementation of `stringForObjectValue` (page 420) to get the nonattributed string. Then create an NSAttributedString with it. The default attributes for text in the cell are passed in with *`attributes`*; use this

NSDictionary to reset the attributes of the string when a change in value warrants it (for example, a negative value becomes positive). If an NSAttributedString cannot be created for *anObject*, an `NSFormatter.FormattingException` is thrown. For information on creating attributed strings, see the NSAttributedString (page 67) class.

attributedStringForZero

Returns the NSAttributedString used to display zero values.

```
public NSAttributedString attributedStringForZero()
```

Discussion
By default zero values are displayed according to the format specified for positive values; for more discussion of this subject see "Data Formatting".

See Also
`setAttributedStringForZero` (page 418)

decimalSeparator

Returns a String containing the character the receiver uses to represent decimal separators.

```
public String decimalSeparator()
```

Discussion
Note that the return value doesn't indicate whether decimal separators are enabled.

See Also
`setDecimalSeparator` (page 418)

format

Returns a String containing the format being used by the receiver.

```
public String format()
```

See Also
`setFormat` (page 418)

hasThousandSeparators

Returns `true` to indicate that the receiver's format includes thousand separators, `false` otherwise.

```
public boolean hasThousandSeparators()
```

Discussion
The default is `false`.

See Also
`setHasThousandSeparators` (page 419)

isPartialStringValid

Since this method is invoked each time the user presses a key while the cursor is in the cell, it lets you verify the cell text as the user types it.

```
public boolean isPartialStringValid(String partialString)
```

Discussion
partialString is the text currently in the cell. Return `true` if typed text is acceptable and `false` if it is not. If you return `false`, the cell displays *partialString* minus the last character typed.

See Also
`replacementStringForString` (page 417)

localizesFormat

Returns `true` to indicate that the receiver localizes formats, `false` otherwise.

```
public boolean localizesFormat()
```

Discussion
The default is `false`.

See Also
`setLocalizesFormat` (page 419)

negativeFormat

Returns a String containing the format used by the receiver to display negative numbers.

```
public String negativeFormat()
```

See Also
`setNegativeFormat` (page 419)
`setFormat` (page 418)

objectValueForString

Returns an object created from *aString*.

```
public Object objectValueForString(String aString)
```

Discussion
If an object cannot be created from *aString*, an `NSFormatter.ParsingException` is thrown.

See Also
`stringForObjectValue` (page 420)

positiveFormat

Returns a String containing the format used by the receiver to display positive numbers.

```
public String positiveFormat()
```

See Also
setPositiveFormat (page 419)
setFormat (page 418)

replacementStringForString

Checks whether *aString* is a valid string for the cell.

```
public String replacementStringForString(String aString)
```

Discussion
If it is, returns it unmodified. Otherwise, corrects it and returns the modified string. For example, this method might convert all lowercase letters to uppercase or insert separator characters in a telephone number.

See Also
isPartialStringValid (page 416)

setAllowsFloats

Sets according to *flag* whether the receiver allows as input floating-point values (that is, values that include the period character [.]).

```
public void setAllowsFloats(boolean flag)
```

Discussion
By default, floating point values are allowed as input.

See Also
allowsFloats (page 414)

setAttributedStringForNil

Sets the NSAttributedString the receiver uses to display `null` values to *newAttributedString*.

```
public void setAttributedStringForNil(NSAttributedString newAttributedString)
```

See Also
attributedStringForNil (page 414)

setAttributedStringForNotANumber

Sets the NSAttributedString the receiver uses to display "not a number" values to *newAttributedString*.

```
public void setAttributedStringForNotANumber(NSAttributedString newAttributedString)
```

See Also
attributedStringForNotANumber (page 414)

setAttributedStringForZero

Sets the NSAttributedString the receiver uses to display zero values to *newAttributedString*.

```
public void setAttributedStringForZero(NSAttributedString newAttributedString)
```

See Also
attributedStringForZero (page 415)

setDecimalSeparator

Sets the character the receiver uses as a decimal separator to *newSeparator*. If *newSeparator* contains multiple characters, only the first one is used. If you don't have decimal separators enabled through another means (such as setFormat (page 418)), using this method enables them.

```
public void setDecimalSeparator(String newSeparator)
```

See Also
decimalSeparator (page 415)

setFormat

Sets the receiver's format to the string *aFormat*.

```
public void setFormat(String aFormat)
```

Discussion
aFormat 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".

The following code excerpt shows the three different approaches for setting an NSNumberFormatter object's format using setFormat:

```
NSNumberFormatter numberFormatter = new NSNumberFormatter();

// 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)");
```

See Also
format (page 415)

setHasThousandSeparators

Sets according to *flag* whether the receiver uses thousand separators.

```
public void setHasThousandSeparators(boolean flag)
```

Discussion
When *flag* is `false`, thousand separators are disabled for both positive and negative formats (even if you've set them through another means, such as `setFormat` (page 418)). When *flag* is `true`, thousand separators are used. 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 `false` (though you in effect change this setting to `true` when you set thousand separators through any means, such as `setFormat` (page 418)).

See Also
`hasThousandSeparators` (page 415)

setLocalizesFormat

Sets according to *flag* 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.

```
public void setLocalizesFormat(boolean flag)
```

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.

See Also
`localizesFormat` (page 416)

setNegativeFormat

Sets the format the receiver uses to display negative values to *aFormat*.

```
public void setNegativeFormat(String aFormat)
```

See Also
`negativeFormat` (page 416)
`setFormat` (page 418)

setPositiveFormat

Sets the format the receiver uses to display positive values to *aFormat*.

```
public void setPositiveFormat(String aFormat)
```

See Also
positiveFormat (page 416)
setFormat (page 418)

setTextAttributesForNegativeValues

Sets the text attributes to be used in displaying negative values to *newAttributes*.

```
public void setTextAttributesForNegativeValues(NSDictionary newAttributes)
```

See Also
textAttributesForNegativeValues (page 421)

setTextAttributesForPositiveValues

Sets the text attributes to be used in displaying positive values to *newAttributes*.

```
public void setTextAttributesForPositiveValues(NSDictionary newAttributes)
```

See Also
textAttributesForPositiveValues (page 421)

setThousandSeparator

Sets the character the receiver uses as a thousand separator to *newSeparator*.

```
public void setThousandSeparator(String newSeparator)
```

Discussion
If *newSeparator* contains multiple characters, only the first one is used. If you don't have thousand separators enabled through any other means (such as setFormat (page 418)), using this method enables them.

See Also
thousandSeparator (page 421)

stringForObjectValue

Returns the string that textually represents the cell's object for display and for editing.

```
public String stringForObjectValue(Object anObject)
```

Discussion
First tests the passed-in object to see if it's of the correct class. If it isn't, returns `null`; if it is of the correct class, returns a properly formatted and, if necessary, localized string. If a string cannot be created for *anObject*, an `NSFormatter.FormattingException` is thrown.

See Also
attributedStringForObjectValue (page 414)

objectValueForString (page 416)

textAttributesForNegativeValues

Returns an NSDictionary containing the text attributes that have been set for negative values.

```
public NSDictionary textAttributesForNegativeValues()
```

See Also
setTextAttributesForNegativeValues (page 420)

textAttributesForPositiveValues

Returns an NSDictionary containing the text attributes that have been set for positive values.

```
public NSDictionary textAttributesForPositiveValues()
```

See Also
setTextAttributesForPositiveValues (page 420)

thousandSeparator

Returns a String containing the character the receiver uses to represent thousand separators.

```
public String thousandSeparator()
```

Discussion
By default this is the comma character (,). Note that the return value doesn't indicate whether thousand separators are enabled.

See Also
setThousandSeparator (page 420)

NSObject

Inherits from	Object
Implements	Cloneable NSKeyValueCoding java.io.Serializable
Package:	com.apple.cocoa.foundation
Companion guide	Cocoa Fundamentals Guide

Overview

NSObject is the root class of most Objective-C class hierarchies. NSObject, along with java.lang.Object, is the root class for all things Cocoa in Java.

Interfaces Implemented

NSKeyValueCoding

- `takeValueForKey` (page 689)
- `valueForKey` (page 690)

Tasks

Constructors

`NSObject` (page 424)
Creates an NSObject. You should create instances of concrete subclasses instead of NSObject.

Creating, Copying, and Deallocating Objects

`clone` (page 424)
Returns a new instance that's a copy of the receiver.

`mutableClone` (page 425)
Returns a new instance that's a mutable copy of the receiver. The copy returned is mutable whether the original is mutable or not.

Comparing Objects

`equals` (page 424)
Returns `true` if the receiver and *`anObject`* are equal, `false` otherwise.

`hashCode` (page 425)
Returns an integer that can be used as a table address in a hash table structure.

Describing Objects

`toString` (page 426)
Returns a string that represents the contents of the receiving class.

Key Value Coding

`takeValueForKey` (page 425)
Sets the value for the property identified by *`key`* to *`value`*.

`valueForKey` (page 426)
Returns the value for the property identified by *`key`*.

Constructors

NSObject

Creates an NSObject. You should create instances of concrete subclasses instead of NSObject.

```
public NSObject()
```

Instance Methods

clone

Returns a new instance that's a copy of the receiver.

```
public Object clone()
```

Discussion
The copy returned is immutable if applicable to the receiver; otherwise the exact nature of the copy is determined by the class. Throws `java.lang.CloneNotSupportedException`.

equals

Returns `true` if the receiver and *`anObject`* are equal, `false` otherwise.

```
public boolean equals(Object anObject)
```

Discussion
NSObject's implementation compares the `id` of *anObject* and the receiver to determine equality. Subclasses can override this method to redefine what it means for objects to be equal. For example, a container object might define two containers as equal if they contain the same contents. See the NSData (page 131), NSDictionary (page 153), NSArray (page 55), and `java.lang.String` class specifications for examples of the use of this method. If subclasses override `equals`, they must also override `hashCode` (page 425) to ensure that all objects that return `true` for `equals` also return the same value for `hashCode`. Note that equality as defined by this method is not necessarily reflexive. For example, A is equal to B does not imply B is equal to A, especially if B is a subclass of A.

hashCode

Returns an integer that can be used as a table address in a hash table structure.

```
public int hashCode()
```

Discussion
NSObject's implementation returns a value based on the object's `id`. If two objects are equal (as determined by the `equals` (page 424) method), they must return the same hash value. This last point is particularly important if you define `hashCode` in a subclass and intend to put instances of that subclass into a collection.

mutableClone

Returns a new instance that's a mutable copy of the receiver. The copy returned is mutable whether the original is mutable or not.

```
public Object mutableClone()
```

takeValueForKey

Sets the value for the property identified by *key* to *value*.

```
public void takeValueForKey(Object value, String key)
```

Discussion
The default implementation works as follows:

1. Searches for a public accessor method of the form `set`*Key*, invoking it if there is one.
2. If a public accessor method isn't found, searches for a private accessor method of the form `_set`*Key*, invoking it if there is one.
3. If an accessor method isn't found, `takeValueForKey` searches for an instance variable based on *key* and sets the *value* directly. If the *key* is "lastName", it searches for an instance variable named `_lastName` or `lastName`.

toString

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

```
public String toString()
```

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

valueForKey

Returns the value for the property identified by *key*.

```
public Object valueForKey(String key)
```

Discussion
The default implementation works as follows:

1. Searches for a public accessor method based on *key*. For example, with a *key* of "lastName", `valueForKey` looks for a method named `getLastName` or `lastName`.

2. If a public accessor method isn't found, searches for a private accessor method based on *key* (a method preceded by an underbar). For example, with a *key* of "lastName", `valueForKey` looks for a method named `_getLastName` or `_lastName`.

3. If an accessor method isn't found, `valueForKey` searches for an instance variable based on *key* and returns its value directly. For the *key* "lastName", this would be `_lastName` or `lastName`.

NSPathUtilities

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Low-Level File Management Programming Topics

Overview

The NSPathUtilities class provides numerous methods for manipulating file paths. The capabilities include dividing file paths into individual path components, combining individual path components into a full file path, obtaining paths to standard locations in the file system, and retrieving and setting file attributes.

Tasks

Constructors

`NSPathUtilities` (page 429)
Creates a new NSPathUtilities object.

Obtaining Standard Paths

`searchPathForDirectoriesInDomains` (page 432)
Creates a list of path strings for the specified directories in the specified domains.

`temporaryDirectory` (page 436)
Returns the path to the system's temporary directory.

Converting Between Paths and URLs

`URLWithPath` (page 436)
Returns a newly created URL referencing the file or directory at *path*.

`pathFromURL` (page 431)
Returns the path of *aURL* conforming to RFC 1808.

Getting and Setting File Attributes

`fileAttributes` (page 429)
Returns an NSDictionary containing various objects that represent the POSIX attributes of the file specified at *path*.

`setFileAttributes` (page 432)
Changes the attributes of the file or directory specified by *path*.

Manipulating Path Strings

`displayNameAtPath` (page 429)
Returns the name of the file or directory at *path* in a form appropriate for presentation to the user.

`isAbsolutePath` (page 430)
Interprets *aString* as a path, returning `true` if it represents an absolute path, `false` if it represents a relative path.

`lastPathComponent` (page 430)
Returns the last path component of *aString*.

`pathComponents` (page 430)
Interprets *aString* as a path, returning an array of strings containing, in order, each path component of *aString*.

`pathExtension` (page 431)
Interprets *aString* as a path, returning the extension of *aString*, if any (not including the extension divider).

`pathWithComponents` (page 431)
Returns a string built from the strings in *components*, by concatenating them with a path separator between each pair.

`pathsMatchingExtensions` (page 431)
Returns an array containing all the elements from the *pathNames* array that have filename extensions from the *filterTypes* array.

`stringByAbbreviatingWithTildeInPath` (page 432)
Returns a string representing *aString* as a path, with a tilde, "~", substituted for the full path to the current user's home directory, or "~user" for a user other than the current user.

`stringByAppendingPathComponent` (page 433)
Returns a string made by appending *aString1* with *aString2*, preceded if necessary by a path separator.

`stringByAppendingPathExtension` (page 433)
Returns a string made by appending to *aString1* an extension separator followed by *aString2*.

`stringByDeletingLastPathComponent` (page 434)
Returns a string made by deleting the last path component from *aString*, along with any final path separator.

`stringByDeletingPathExtension` (page 434)
Returns a string made by deleting the extension (if any, and only the last) from *aString*.

`stringByExpandingTildeInPath` (page 435)
Returns a string made by expanding the initial component of *aString*, if it begins with "~" or "~user", to its full path value.

`stringByResolvingSymlinksInPath` (page 435)
Expands an initial tilde expression in *aPath*, then resolves all symbolic links and references to current or parent directories if possible, returning a standardized path.

`stringByStandardizingPath` (page 435)
Returns a string representing the path of *aString*, with extraneous path components removed.

`stringsByAppendingPaths` (page 436)
Returns an array of strings made by separately appending each string in *paths* to *aString*, preceded if necessary by a path separator.

Constructors

NSPathUtilities

Creates a new NSPathUtilities object.

```
public NSPathUtilities()
```

Discussion
All the NSPathUtilities methods are static, so there is no need to create instances of NSPathUtilities.

Static Methods

displayNameAtPath

Returns the name of the file or directory at *path* in a form appropriate for presentation to the user.

```
public static String displayNameAtPath(String path)
```

fileAttributes

Returns an NSDictionary containing various objects that represent the POSIX attributes of the file specified at *path*.

```
public static NSDictionary fileAttributes(String path, boolean flag)
```

Discussion
You access these objects using the keys described in the "Constants" (page 437) section.

If *flag* is `true` and *path* is a symbolic link, the attributes of the linked-to file are returned; if the link points to a nonexistent file, this method returns `null`. If *flag* is `false`, the attributes of the symbolic link are returned.

See Also
`setFileAttributes` (page 432)

isAbsolutePath

Interprets *aString* as a path, returning `true` if it represents an absolute path, `false` if it represents a relative path.

```
public static boolean isAbsolutePath(String aString)
```

lastPathComponent

Returns the last path component of *aString*.

```
public static String lastPathComponent(String aString)
```

Discussion
The following table illustrates the effect of `lastPathComponent` on a variety of different paths:

Value of *aString*	String Returned
"/tmp/scratch.tiff"	"scratch.tiff"
"/tmp/scratch"	"scratch"
"/tmp/"	"tmp"
"scratch"	"scratch"
"/"	"/"

pathComponents

Interprets *aString* as a path, returning an array of strings containing, in order, each path component of *aString*.

```
public static NSArray pathComponents(String aString)
```

Discussion
The strings in the array appear in the order they did in *aString*. If *aString* begins or ends with the path separator, then the first or last component, respectively, contains the separator. Empty components (caused by consecutive path separators) are deleted.

If *aString* begins with a slash—for example, "/tmp/scratch"—the array has these contents:

Index	Path Component
0	"/"
1	"tmp"
2	"scratch"

If *aString* has no separators—for example, "scratch"—the array contains *aString* itself, in this case "scratch".

See Also
pathWithComponents (page 431)
stringByStandardizingPath (page 435)

pathExtension

Interprets *aString* as a path, returning the extension of *aString*, if any (not including the extension divider).

```
public static String pathExtension(String aString)
```

Discussion
The following table illustrates the effect of pathExtension on a variety of different paths:

Value of *aString*	String Returned
"/tmp/scratch.tiff"	"tiff"
"/tmp/scratch"	"" (an empty string)
"/tmp/"	"" (an empty string)
"/tmp/scratch..tiff"	"tiff"

pathFromURL

Returns the path of *aURL* conforming to RFC 1808.

```
public static String pathFromURL(java.net.URL aURL)
```

Discussion
If *aURL* does not conform to RFC 1808, returns null.

pathsMatchingExtensions

Returns an array containing all the elements from the *pathNames* array that have filename extensions from the *filterTypes* array.

```
public static NSArray pathsMatchingExtensions(NSArray pathNames, NSArray filterTypes)
```

Discussion
The extensions in *filterTypes* should not include the dot (".") character.

pathWithComponents

Returns a string built from the strings in *components*, by concatenating them with a path separator between each pair.

```
public static String pathWithComponents(NSArray components)
```

Discussion
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. This method doesn't clean up the path created; use `stringByStandardizingPath` (page 435) to resolve empty components, references to the parent directory, and so on.

See Also
`pathComponents` (page 430)

searchPathForDirectoriesInDomains

Creates a list of path strings for the specified directories in the specified domains.

```
public static NSArray searchPathForDirectoriesInDomains(int directory, int
    domainMask, boolean expandTilde)
```

Discussion
The list is in the order in which you should search the directories. The list of values for *`directory`* and *`domainMask`* is described in "Constants" (page 437). If *`expandTilde`* is `true`, tildes are expanded as described in `stringByExpandingTildeInPath` (page 435).

setFileAttributes

Changes the attributes of the file or directory specified by *`path`*.

```
public static boolean setFileAttributes(String path, NSDictionary attributes)
```

Discussion
The attributes that you can change include the owner, group, file permissions, and modification date. (The list of file attribute keys are given in the "Constants" (page 437) section; their values cannot all be changed, however.) 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. If all changes succeed, it returns `true`. If any change fails, the method returns `false`, but it is undefined whether any changes actually occurred.

The `FilePosixPermissions` value must be initialized with the code representing the POSIX file-permissions bit pattern. `FileHFSCreatorCode` and `FileHFSTypeCode` will only be heeded when *`path`* specifies a file.

You can change single attributes or any combination of attributes; you need not specify keys for all attributes.

See Also
`fileAttributes` (page 429)

stringByAbbreviatingWithTildeInPath

Returns a string representing *`aString`* as a path, with a tilde, "~", substituted for the full path to the current user's home directory, or "`~user`" for a user other than the current user.

```
public static String stringByAbbreviatingWithTildeInPath(String aString)
```

Discussion
Returns *`aString`* unaltered if it doesn't begin with the user's home directory.

See Also
stringByExpandingTildeInPath (page 435)

stringByAppendingPathComponent

Returns a string made by appending *aString1* with *aString2*, preceded if necessary by a path separator.

```
public static String stringByAppendingPathComponent(String aString1, String aString2)
```

Discussion
The following table illustrates the effect of this method on a variety of different paths, assuming that *aString2* is supplied as "scratch.tiff":

Value of *aString1*	Resulting String
"/tmp"	"/tmp/scratch.tiff"
"/tmp/"	"/tmp/scratch.tiff"
"/"	"/scratch.tiff"
"" (an empty string)	"scratch.tiff"

This method may fail to append if PATH_MAX is exceeded. Its failure mode is to return the original string passed to the method as *string1*, rather than a string with anything appended to it. The method also logs in console.log the fact that the attempt failed.

See Also
stringsByAppendingPaths (page 436)
stringByAppendingPathExtension (page 433)
stringByDeletingLastPathComponent (page 434)

stringByAppendingPathExtension

Returns a string made by appending to *aString1* an extension separator followed by *aString2*.

```
public static String stringByAppendingPathExtension(String aString1, String aString2)
```

Discussion
The following table illustrates the effect of this method on a variety of different paths, assuming that *aString2* is supplied as "tiff":

Value of *aString*	Resulting String
"/tmp/scratch.old"	"/tmp/scratch.old.tiff"
"/tmp/scratch."	"/tmp/scratch..tiff"
"/tmp/"	"/tmp/.tiff"
"scratch"	"scratch.tiff"

See Also
stringByAppendingPathComponent (page 433)
stringByDeletingPathExtension (page 434)

stringByDeletingLastPathComponent

Returns a string made by deleting the last path component from *aString*, along with any final path separator.

```
public static String stringByDeletingLastPathComponent(String aString)
```

Discussion
If *aString* represents the root path, however, it's returned unaltered. The following table illustrates the effect of this method on a variety of different paths:

Value of *aString*	Resulting String
"/tmp/scratch.tiff"	"/tmp"
"/tmp/lock/"	"/tmp"
"/tmp/"	"/"
"/tmp"	"/"
"/"	"/"
"scratch.tiff"	"" (an empty string)

See Also
stringByDeletingPathExtension (page 434)
stringByAppendingPathComponent (page 433)

stringByDeletingPathExtension

Returns a string made by deleting the extension (if any, and only the last) from *aString*.

```
public static String stringByDeletingPathExtension(String aString)
```

Discussion
Strips any trailing path separator before checking for an extension. If *aString* represents the root path, however, it's returned unaltered. The following table illustrates the effect of this method on a variety of different paths:

Value of *aString*	Resulting String
"/tmp/scratch.tiff"	"/tmp/scratch"
"/tmp/"	"/tmp"
"scratch.bundle/"	"scratch"

Value of *aString*	Resulting String
"scratch..tiff"	"scratch."
".tiff"	"" (an empty string)
"/"	"/"

See Also
pathExtension (page 431)
stringByDeletingLastPathComponent (page 434)

stringByExpandingTildeInPath

Returns a string made by expanding the initial component of *aString*, if it begins with "~" or "~user", to its full path value.

```
public static String stringByExpandingTildeInPath(String aString)
```

Discussion
Returns *aString* unaltered if that component can't be expanded.

See Also
stringByAbbreviatingWithTildeInPath (page 432)

stringByResolvingSymlinksInPath

Expands an initial tilde expression in *aPath*, then resolves all symbolic links and references to current or parent directories if possible, returning a standardized path.

```
public static String stringByResolvingSymlinksInPath(String aPath)
```

Discussion
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 `this` if an error occurs.

If the name of *aPath* begins with `/private`, this method strips off the `/private` designator, provided the result is the name of an existing file.

See Also
stringByStandardizingPath (page 435)
stringByExpandingTildeInPath (page 435)

stringByStandardizingPath

Returns a string representing the path of *aString*, with extraneous path components removed.

```
public static String stringByStandardizingPath(String aString)
```

Discussion
If `stringByStandardizingPath` detects symbolic links in a pathname, the `stringByResolvingSymlinksInPath` (page 435) 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), `this` is returned.

This method can make the following changes in the provided string:

- Expand an initial tilde expression using `stringByExpandingTildeInPath` (page 435).
- 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 435), 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).

See Also
`stringByExpandingTildeInPath` (page 435)
`stringByResolvingSymlinksInPath` (page 435)

stringsByAppendingPaths

Returns an array of strings made by separately appending each string in *`paths`* to *`aString`*, preceded if necessary by a path separator.

```
public static NSArray stringsByAppendingPaths(String aString, NSArray paths)
```

Discussion
See `stringByAppendingPathComponent` (page 433) for an individual example.

temporaryDirectory

Returns the path to the system's temporary directory.

```
public static String temporaryDirectory()
```

URLWithPath

Returns a newly created URL referencing the file or directory at *`path`*.

```
public static java.net.URL URLWithPath(String path)
```

Constants

The following constants for directory locations are provided by NSPathUtilities:

Constant	Description
`AdminApplicationDirectory`	System and network administration applications.
`AllApplicationsDirectory`	All directories where applications can occur.
`AllLibrariesDirectory`	All directories where resources can occur.
`ApplicationDirectory`	Supported applications (`/Applications`).
`DemoApplicationDirectory`	Unsupported applications and demonstration versions.
`DeveloperApplicationDirectory`	Developer applications (`/Developer/Applications`).
`DeveloperDirectory`	Developer resources (`/Developer`).
`DocumentationDirectory`	Documentation.
`LibraryDirectory`	Various user-visible documentation, support, and configuration files (`/Library`).
`UserDirectory`	User home directories (`/Users`).

The following masks are provided for working with paths:

Constant	Description
`AllDomainsMask`	All domains. Includes all of the below and future items.
`LocalDomainMask`	Local to the current machine—the place to install items available to everyone on this machine (`/Local`).
`NetworkDomainMask`	Publicly available location in the local are a network—the place to install items available on the network (`/Network`).
`SystemDomainMask`	Provided by Apple—can't be modified (`/System`).
`UserDomainMask`	The user's home directory—the place to install user's personal items (~).

The following are keys used to access attributes of a mounted file system:

Key	Value Type
`FileSystemSize` (in an appropriate unit, usually `bytes`)	`int`
`FileSystemFreeSize` (in an appropriate unit, usually bytes)	`int`
`FileSystemNodes`	`int`

Key	Value Type
FileSystemFreeNodes	int
FileSystemNumber	int

The following are keys used to access file attributes contained in the attribute dictionaries returned from fileAttributes (page 429) and passed to setFileAttributes (page 432):

Key	Value Type
FileSize (in bytes)	int
FileModificationDate	NSDate
FileOwnerAccountName	String
FileGroupOwnerAccountName	String
FileReferenceCount (number of hard links)	int
FileDeviceIdentifier	int
FilePosixPermissions	int
FileType	String (see below for possible values)
FileExtensionHidden	boolean
FileHFSCreatorCode	int (see “HFS File Types”)
FileHFSTypeCode	int (see “HFS File Types”)
FileSystemFileNumber	int
FileImmutable	boolean
FileAppendOnly	boolean
FileCreationDate	NSDate
FileOwnerAccountID	int
FileGroupOwnerAccountID	int

The following constants are the possible values for the FileType attribute key:

Constant	Description
FileTypeDirectory	Directory

Constant	Description
`FileTypeRegular`	Regular file
`FileTypeSymbolicLink`	Symbolic link
`FileTypeSocket`	Socket
`FileTypeCharacterSpecial`	Character special file
`FileTypeBlockSpecial`	Block special file
`FileTypeUnknown`	Unknown

NSPoint

Inherits from	Object
Implements	Cloneable
Package:	com.apple.cocoa.foundation
Companion guide	Geometry and Range Utilities

Overview

An NSPoint represents a location in a coordinate system. It has two primary values, an x-coordinate (horizontal) and a y-coordinate (vertical). The methods of NSPoint give access to these values, convert between an NSPoint and its string representation, measure the distance between two NSPoints, and compare two NSPoints for equality.

Tasks

Constructors

`NSPoint` (page 442)

Accessing Coordinate Values

`x` (page 444)
Returns the x-coordinate of the receiver.

`y` (page 444)
Returns the y-coordinate of the receiver.

Converting Points

`hashCode` (page 443)
Provide an appropriate hash code useful for storing the receiver in a hash-based data structure.

`fromString` (page 443)

`toAWTPoint` (page 444)
Returns the receiver as a AWT Point object.

`toString` (page 444)
Returns the receiver as converted to a string object in the form of "{x, y}" where x is the float representation of the x-coordinate value and y is the float representation of the y-coordinate value.

Finding the Distance Between Points

`distanceToPoint` (page 443)
Computes and returns the distance between *aPoint* and the receiver.

Testing Points

`equals` (page 443)
Returns whether *otherObject* is an NSPoint and has the same x-coordinate and y-coordinate values as the receiver.

`isEqualToPoint` (page 444)
Returns whether *aPoint* has the same x-coordinate and y-coordinate as the receiver.

Copying

`clone` (page 443)
Creates and returns a copy of the receiver.

Constructors

NSPoint

```
public NSPoint()
```

Discussion
Initializes both x and y coordinates to zero.

```
public NSPoint(float x, float y)
```

Discussion
Initializes the NSPoint with the horizontal coordinate *x* and the vertical coordinate *y*.

```
public NSPoint(NSPoint aPoint)
```

Discussion
Initializes the new NSPoint with the coordinate values of existing NSPoint *aPoint*; this constructor is used in cloning the receiver.

```
public NSPoint(java.awt.Point javaPoint)
```

Discussion
Initializes the NSPoint with the values extracted from an AWT Point object.

Static Methods

fromString

```
public static NSPoint fromString(String pointAsString)
```

Discussion
Creates an NSPoint from the string *pointAsString*, which must be of the form "{x , y}" where x is a float representation of the x-coordinate and y is a float representation of the y-coordinate. Throws an `IllegalArgumentException` if the string is improperly formatted.

See Also
`toString` (page 444)

Instance Methods

clone

Creates and returns a copy of the receiver.

```
public Object clone()
```

distanceToPoint

Computes and returns the distance between *aPoint* and the receiver.

```
public float distanceToPoint(NSPoint aPoint)
```

equals

Returns whether *otherObject* is an NSPoint and has the same x-coordinate and y-coordinate values as the receiver.

```
public boolean equals(Object otherObject)
```

See Also
`isEqualToPoint` (page 444)

hashCode

Provide an appropriate hash code useful for storing the receiver in a hash-based data structure.

```
public int hashCode()
```

Discussion
This value is the sum of the receiver's x-coordinate and y-coordinate, rounded to the nearest integer.

isEqualToPoint

Returns whether *aPoint* has the same x-coordinate and y-coordinate as the receiver.

```
public boolean isEqualToPoint(NSPoint aPoint)
```

See Also
equals (page 443)

toAWTPoint

Returns the receiver as a AWT Point object.

```
public java.awt.Point toAWTPoint()
```

Discussion
The float values of the x-coordinate and the y-coordinate are rounded down to the nearest integer (that is, the point is moved "down" and to the "left" to the nearest integer).

See Also
toString (page 444)

toString

Returns the receiver as converted to a string object in the form of "{x, y}" where x is the float representation of the x-coordinate value and y is the float representation of the y-coordinate value.

```
public String toString()
```

See Also
fromString (page 443)

x

Returns the x-coordinate of the receiver.

```
public float x()
```

y

Returns the y-coordinate of the receiver.

```
public float y()
```

Constants

NSPoint provides the following constant as a convenience; you can use it to compare values returned by some NSPoint methods:

Constant	Description
`ZeroPoint`	An NSPoint with both x and y coordinates set to zero.

NSPort

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Run Loops

Overview

NSPort is an abstract class that represents a communication channel to or from another NSPort, which typically resides in a different thread or task.

To receive incoming messages, NSPorts must be added to an NSRunLoop as input sources.

Tasks

Constructors

`NSPort` (page 448)
Creates a new NSPort object capable of both sending and receiving messages.

Validation

`invalidate` (page 448)
Marks the receiver as invalid.

`isValid` (page 448)
Returns `false` if the receiver is known to be invalid, `true` otherwise (an NSPort only notes that it has become invalid when it tries to send or receive a message).

Setting the Delegate

`setDelegate` (page 449)

`delegate` (page 448)
Returns the receiver's delegate.

Constructors

NSPort

Creates a new NSPort object capable of both sending and receiving messages.

```
public NSPort()
```

Creates a newly allocated NSPort object to use the Mach port *machPort*.

```
public NSPort(int machPort)
```

Discussion
Depending on the access rights for *machPort*, the new NSPort may only be able to send messages.

Instance Methods

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

See Also
setDelegate (page 449)

invalidate

Marks the receiver as invalid.

```
public void invalidate()
```

See Also
isValid (page 448)

isValid

Returns false if the receiver is known to be invalid, true otherwise (an NSPort only notes that it has become invalid when it tries to send or receive a message).

```
public boolean isValid()
```

Discussion
An NSPort becomes invalid when its underlying communication resource, which is operating system dependent, is closed or damaged.

See Also
invalidate (page 448)

setDelegate

```
public void setDelegate(Object anObject)
```

Discussion
Sets the receiver's delegate to *anObject*.

See Also
delegate (page 448)

NSPositionalSpecifier

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
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. NSPositionalSpecifiers 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. For example, the following snippet of Objective-C code illustrates the object specifiers and the positional specifier created to represent the script statement `move word 1 after word 5`.

```
    NSPositionalSpecifier *posSpec;
    NSScriptObjectSpecifier *spec;
    NSScriptCommand *command;
    spec = [[[NSIndexSpecifier allocWithZone:[self zone]]
            initWithContainerClassDescription:nil containerSpecifier:nil  key:
 @"orderedDocuments" index:0]
            autorelease];
    spec = [[[NSPropertySpecifier allocWithZone:[self zone]]
            initWithContainerClassDescription:nil containerSpecifier:spec
key:@"textStorage"]
            autorelease];
    spec = [[[NSIndexSpecifier allocWithZone:[self zone]]
            initWithContainerClassDescription:nil containerSpecifier:spec
key:@"words" index: 0]
            autorelease];
    command = [moveCommandDef createCommandInstanceWithZone:[self  zone]];
    [command setReceiversSpecifier:spec];
    spec = [[[NSIndexSpecifier allocWithZone:[self zone]]
            initWithContainerClassDescription:nil containerSpecifier:nil  key:
 @"orderedDocuments" index:0]
            autorelease];
    spec = [[[NSPropertySpecifier allocWithZone:[self zone]]
            initWithContainerClassDescription:nil containerSpecifier:spec
key:@"textStorage"]
            autorelease];
    spec = [[[NSIndexSpecifier allocWithZone:[self zone]]
            initWithContainerClassDescription:nil containerSpecifier:spec
key:@"words" index: 4]
            autorelease];
/* Here the positional specifier is created, encapsulating the 'word  5' object
```

```
 * specifier, and set as the argument of the command
*/
    posSpec = [[[NSPositionalSpecifier allocWithZone:[self zone]]
            initWithPosition:NSPositionAfter objectSpecifier:spec]
            autorelease];
    [command setArguments:[NSDictionary dictionaryWithObjectsAndKeys:posSpec,
 @"ToLocation",  nil]];
```

The public interface of NSPositionalSpecifier allows you to access the container of the object specified in the argument (`insertionContainer` (page 453)) as well as the insertion key of the object (`insertionKey` (page 453)) and the position of the object in the insertion key of this container (`insertionIndex` (page 453)). Invoking any one of these messages causes the NSPositionalSpecifier to be evaluated if it hasn't been already.

You don't normally subclass NSPositionalSpecifier.

Tasks

Constructors

`NSPositionalSpecifier` (page 453)
Returns an NSPositionalSpecifier with no data.

Accessing Information About a Positional Specifier

`insertionContainer` (page 453)
Returns the container identified by the receiver.

`insertionIndex` (page 453)
Returns the index identified by the receiver.

`insertionKey` (page 453)
Returns the key identified by the receiver.

`insertionReplaces` (page 454)
Returns `true` 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, or `false` otherwise. If this object has never been evaluated, evaluation is attempted.

`setInsertionClassDescription` (page 454)
Sets the class description for the object or objects to be inserted. This message can be sent at any time after object initialization, but must be sent before evaluation to have any effect.

Evaluating a Positional Specifier

`evaluate` (page 453)
Causes the receiver to evaluate its position.

Constructors

NSPositionalSpecifier

Returns an NSPositionalSpecifier with no data.

```
public NSPositionalSpecifier()
```

Discussion
Do not use this constructor.

Returns an NSPositionalSpecifier with position specified by *position* and object specifier set to *specifier*.

```
public NSPositionalSpecifier(int position, NSScriptObjectSpecifier specifier)
```

Discussion
See "Constants" (page 454) for possible values for *position*.

Instance Methods

evaluate

Causes the receiver to evaluate its position.

```
public void evaluate()
```

Discussion
Calling `insertionContainer` (page 453), `insertionKey` (page 453), `insertionIndex` (page 453), or `insertionReplaces` (page 454) also causes the receiver to be evaluated, if it hasn't already been evaluated.

insertionContainer

Returns the container identified by the receiver.

```
public Object insertionContainer()
```

insertionIndex

Returns the index identified by the receiver.

```
public int insertionIndex()
```

insertionKey

Returns the key identified by the receiver.

```
public String insertionKey()
```

insertionReplaces

Returns `true` 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, or `false` otherwise. If this object has never been evaluated, evaluation is attempted.

```
public boolean insertionReplaces()
```

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

setInsertionClassDescription

Sets the class description for the object or objects to be inserted. This message can be sent at any time after object initialization, but must be sent before evaluation to have any effect.

```
public void setInsertionClassDescription(NSScriptClassDescription classDescription)
```

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

Constants

The following constants are defined by NSPositionalSpecifier:

Constant	Description
`PositionAfter`	Specifies a position after another object.
`PositionBefore`	Specifies a position before another object.
`PositionBeginning`	Specifies a position at the beginning of a collection.
`PositionEnd`	Specifies a position at the end of a collection.
`PositionReplace`	Specifies a position in the place of another object.

NSPredicate

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.foundation
Availability	Available in Mac OS X v10.4 and later.
Companion guide	Predicate Programming Guide

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. You often create predicates from a format string which is parsed by the class methods on NSPredicate; examples 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")`

You can create predicates for relationships, such as:

- `group.name matches "work*"`
- `ALL children.age > 12`
- `ANY children.age > 12`

You can also create predicates for operations, such as `@sum.items.price < 1000`.

In general, the order for `like` and `matches` predicates is `propertyName OPERATOR pattern`.

It is also common to create predicates directly—this is more efficient, more (type-)safe, and it avoids localization issues.

You can also create predicates that include variables, so that they can be pre-defined (in a tool) before substituting concrete values at runtime. For predicates that use variables, evaluation is a two step process (see `predicateWithSubstitutionVariables` (page 458) and `evaluateWithObject` (page 457)).

Tasks

Constructors

`NSPredicate` (page 456)

Creating Predicates

`predicateWithFormat` (page 457)
Returns a new predicate by substituting the values in *`arguments`* into *`predicateFormat`* in the order they appear, and parsing the result.

`predicateWithSubstitutionVariables` (page 458)
Returns a copy of the receiver with the receiver's variables substituted by values specified in the substitution variables dictionary.

`predicateWithValue` (page 457)

Evaluating

`evaluateWithObject` (page 457)
Returns `true` if *`object`* matches the conditions specified by the receiver; otherwise `false`

Getting Format Information

`predicateFormat` (page 457)

Constructors

NSPredicate

```
public NSPredicate()
```

Discussion
Creates an empty NSPredicate object.

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

Static Methods

predicateWithFormat

Returns a new predicate by substituting the values in *arguments* into *predicateFormat* in the order they appear, and parsing the result.

```
public static NSPredicate predicateWithFormat(String predicateFormat, NSArray
    arguments)
```

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

predicateWithValue

```
public static NSPredicate predicateWithValue(boolean value)
```

Discussion
Returns a predicate that always evaluates to *value*.

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

Instance Methods

evaluateWithObject

Returns `true` if *object* matches the conditions specified by the receiver; otherwise `false`

```
public boolean evaluateWithObject(Object object)
```

Discussion
.

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

predicateFormat

```
public String predicateFormat()
```

Discussion
Returns the receiver's format string.

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

predicateWithSubstitutionVariables

Returns a copy of the receiver with the receiver's variables substituted by values specified in the substitution variables dictionary.

```
public NSPredicate predicateWithSubstitutionVariables(NSDictionary variables)
```

Discussion
The substitution variables dictionary is specified by *variables* which must contain key-value pairs for all variables in the predicate.

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.

NSPropertyListSerialization

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guides	Archives and Serializations Programming Guide for Cocoa Property List Programming Guide

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 number objects.

Tasks

Constructors

`NSPropertyListSerialization` (page 460)
Creates a new NSPropertyListSerialization object.

Serializing Property Lists

`XMLDataFromPropertyList` (page 462)
Creates a data object, serializes *`aPropertyList`* into it in XML format, and returns the data object.

`dataFromPropertyList` (page 460)
Creates a data object, serializes *`aPropertyList`* into it, and returns the data object.

`stringFromPropertyList` (page 462)
Creates an old-style string representation (non-XML) of *`aPropertyList`*.

Testing Property Lists

`propertyListIsValidForFormat` (page 462)
Returns `true` if *`plist`* is a valid property list in format *`format`*, `false` otherwise.

Deserializing Property Lists

`propertyListFromData` (page 461)
Returns a property list object corresponding to the representation in *`serialization`* or `null` if *`serialization`* does not represent a property list.

`propertyListFromString` (page 461)
Returns a property list object corresponding to the representation in *`serialization`* or `null` if *`serialization`* does not represent a property list.

`propertyListFromXMLData` (page 462)
Returns a property list object corresponding to the XML representation in *`serialization`* or `null` if *`serialization`* does not represent a property list.

Constructors

NSPropertyListSerialization

Creates a new NSPropertyListSerialization object.

```
public NSPropertyListSerialization()
```

Discussion
All the NSPropertyListSerialization methods are static, so there is no need to create instances of NSPropertyListSerialization.

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

Static Methods

dataFromPropertyList

Creates a data object, serializes *`aPropertyList`* into it, and returns the data object.

```
public static NSData dataFromPropertyList(Object aPropertyList)
```

Discussion
`aPropertyList` must be a kind of NSData, string, NSArray, or NSDictionary. Container objects must also contain only these kinds of objects.

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

See Also
`propertyListFromData` (page 461)

Returns an NSData object in the property list format specified by *`plist`* or `null` if *`plist`* does not represent a valid property list.

```
public static NSData dataFromPropertyList(Object plist, int format, String[]
    errorString)
```

Discussion
plist must be a kind of NSData, string, number, NSDate, NSArray, or NSDictionary. Container objects must also contain only these kinds of objects. On return, if the conversion is successful, *errorString* is `null`. If the conversion fails, *errorString* points to a string describing the nature of the error. Possible values for *format* are described in "Constants" (page 463).

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

See Also
`propertyListFromData` (page 461)

propertyListFromData

Returns a property list object corresponding to the representation in *serialization* or `null` if *serialization* does not represent a property list.

```
public static Object propertyListFromData(NSData serialization)
```

Discussion
If the property list object is a dictionary or an array, the recomposed object is made immutable.

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

See Also
`dataFromPropertyList` (page 460)

Returns a property list object corresponding to the representation in *data*, or `null` if *data* does not represent a valid property list in a supported format.

```
public static Object propertyListFromData(NSData data, int opt, int[] format,
    String[] errorString)
```

Discussion
The mutability option *opt* determines whether the property list's containers and/or leaves are mutable. If the property list is valid, the format is returned in the first element of the array *format*. Possible values for *opt* and *format* are described in "Constants" (page 463).

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

See Also
`dataFromPropertyList` (page 460)

propertyListFromString

Returns a property list object corresponding to the representation in *serialization* or `null` if *serialization* does not represent a property list.

```
public static Object propertyListFromString(String serialization)
```

Discussion
If the property list object is a dictionary or an array, the recomposed object is made immutable. This method can parse both XML and non-XML style property list strings.

See Also
`stringFromPropertyList` (page 462)

propertyListFromXMLData

Returns a property list object corresponding to the XML representation in *serialization* or `null` if *serialization* does not represent a property list.

```
public static Object propertyListFromXMLData(NSData serialization)
```

Discussion
If the property list object is a dictionary or an array, the recomposed object is made immutable.

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

See Also
`XMLDataFromPropertyList` (page 462)

propertyListIsValidForFormat

Returns `true` if *plist* is a valid property list in format *format*, `false` otherwise.

```
public static boolean propertyListIsValidForFormat(Object plist, int format)
```

Discussion
Possible values for format are listed in "Constants" (page 463).

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

stringFromPropertyList

Creates an old-style string representation (non-XML) of *aPropertyList*.

```
public static String stringFromPropertyList(Object aPropertyList)
```

Discussion
aPropertyList must be a kind of NSData, NSString, NSDate, number class, NSArray, or NSDictionary. Container objects must also contain only these same kinds of objects. When read back by `propertyListFromString` (page 461), NSDate and numerical values are interpreted as strings.

XMLDataFromPropertyList

Creates a data object, serializes *aPropertyList* into it in XML format, and returns the data object.

```
public static NSData XMLDataFromPropertyList(Object aPropertyList)
```

Discussion
aPropertyList must be a kind of NSData, string, NSDate, number, NSArray, or NSDictionary. Container objects must also contain only these same kinds of objects.

Note that XML can handle more data types than the binary serialization created by `dataFromPropertyList` (page 460).

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

See Also
`propertyListFromXMLData` (page 462)

Constants

The following table describes the constants used to specify mutability options in property lists:

Constant	Description
`PropertyListImmutable`	Causes the returned property list to contain immutable objects.
`PropertyListMutableContainers`	Causes the returned property list to have mutable containers but immutable leaves.
`PropertyListMutable-ContainersAndLeaves`	Causes the returned property list to have mutable containers and leaves.

The following table describes the constants used to specify property list serialization format:

Constant	Description
`PropertyListOpenStepFormat`	Specifies the old-style ASCII property list format inherited from the OpenStep APIs.
`PropertyListXMLFormat`	Specifies the XML property list format.
`PropertyListBinaryFormat`	Specifies the binary property list format.

NSPropertySpecifier

Inherits from	NSScriptObjectSpecifier : NSObject
Package:	com.apple.cocoa.foundation
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.

Tasks

Constructors

`NSPropertySpecifier` (page 465)
Returns an NSPropertySpecifier with no data.

Constructors

NSPropertySpecifier

Returns an NSPropertySpecifier with no data.

```
public NSPropertySpecifier()
```

Discussion
Do not use this constructor.

Returns an NSPropertySpecifier 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.

```
public NSPropertySpecifier(NSScriptClassDescription classDescription,
    NSScriptObjectSpecifier specifier, String key)
```

Discussion
The receiver's child specifier reference is set to `null`.

Returns an NSPropertySpecifier initialized with container specifier *specifier* and key *key*.

```
public NSPropertySpecifier(NSScriptObjectSpecifier specifier, String key)
```

Discussion
The class description of the container is set automatically.

NSQuitCommand

Inherits from	NSScriptCommand : NSObject
Package:	com.apple.cocoa.foundation
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).

NSQuitCommand is part of Cocoa's built-in scripting support. Most applications don't need to subclass NSQuitCommand or call its methods.

Tasks

Constructors

`NSQuitCommand` (page 467)
Returns an NSQuitCommand with no data.

Accessing Options

`saveOptions` (page 468)
Returns a constant indicating how to deal with closing any modified documents.

Constructors

NSQuitCommand

Returns an NSQuitCommand with no data.

```
public NSQuitCommand()
```

Discussion
Do not use this constructor.

Initializes an instance of NSQuitCommand with the command description supplied by *commandDescription*.

```
public NSQuitCommand(NSScriptCommandDescription commandDescription)
```

Discussion
Note that such an instance has no receiver specifier, arguments, or direct parameter and is not a fully functional command.

Instance Methods

saveOptions

Returns a constant indicating how to deal with closing any modified documents.

```
public int saveOptions()
```

Discussion
The default value returned is `SaveOptionsAsk`. See "Constants" (page 112) in NSCloseCommand for a list of possible return values.

NSRandomSpecifier

Inherits from	NSScriptObjectSpecifier
Package:	com.apple.cocoa.foundation
Companion guide	Cocoa Scripting Guide

Overview

Specifies an arbitrary object in a collection or, if not a one-to-many relationship, the sole object.

Tasks

Constructors

`NSRandomSpecifier` (page 469)
Returns an NSRandomSpecifier with no data.

Constructors

NSRandomSpecifier

Returns an NSRandomSpecifier with no data.

```
public NSRandomSpecifier()
```

Discussion
Do not use this constructor.

Returns an NSRandomSpecifier 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.

```
public NSRandomSpecifier(NSScriptClassDescription classDescription,
    NSScriptObjectSpecifier specifier, String key)
```

Discussion
The receiver's child specifier reference is set to `null`.

Returns an NSRandomSpecifier initialized with container specifier *specifier* and key *key*.

```
public NSRandomSpecifier(NSScriptObjectSpecifier specifier, String key)
```

Discussion
The class description of the container is set automatically.

NSRange

Inherits from	Object
Implements	Cloneable
Package:	com.apple.cocoa.foundation
Companion guide	Geometry and Range Utilities

Overview

An NSRange represents a range, a measurement of a segment of something linear, such as a byte stream. An NSRange has two primary values, a location and a length. The methods of NSRange give access to these values; convert between NSRanges and their string representations; test and compare ranges; and create ranges based on operations involving the union, intersection, and subtraction of two ranges.

Tasks

Constructors

`NSRange` (page 472)

Accessing Range Elements

`length` (page 474)
Returns the length of the receiver from its starting location.

`location` (page 475)
Returns the starting location of the receiver.

`locationInRange` (page 475)
Returns whether the location *aLocation* comes after or matches the starting location and comes before the ending location of the receiver.

Manipulating Ranges

`clone` (page 473)
Creates and returns a copy of the receiver.

`rangeByIntersectingRange` (page 475)
Returns an NSRange that is the intersection of *aRange* and the receiver.

`rangeByUnioningRange` (page 476)
Returns an NSRange that is the union of *aRange* and the receiver (a range constructed from the lowest starting location and the highest ending location of both NSRanges).

`subtractRange` (page 476)
Returns the ranges resulting from the subtraction of *otherRange* from the receiver by modifying the mutable ranges *resultRange1* and *resultRange2* (provided by the caller).

Testing Ranges

`equals` (page 473)
Returns whether *otherObject* is an NSRange and is equal in location and length to the receiver.

`hashCode` (page 473)
Provides an appropriate hash code useful for storing the receiver in a hash-based data structure.

`intersectsRange` (page 474)
Returns whether the range *aRange* intersects the receiver.

`isEmpty` (page 474)
Returns whether the length of the receiver is 0.

`isEqualToRange` (page 474)
Returns whether the range *aRange* is equal in both location and length to the receiver.

`isSubrangeOfRange` (page 474)
Returns whether the receiver's endpoints match or fall within those of range *aRange*.

`maxRange` (page 475)
Returns the extent of the receiver (its starting location plus its length).

Converting Between Strings and NSRanges

`fromString` (page 473)

`toString` (page 476)
Returns a String representing the receiver's starting location and length.

Constructors

NSRange

```
public NSRange()
```

Discussion
Returns an empty NSRange.

```
public NSRange(int location, int length)
```

Discussion
Initializes the NSRange with the range elements of `location` and `length`; it throws an `IllegalArgumentException` if either integer is negative.

```
public NSRange(NSRange aRange)
```

Discussion
Initializes the new NSRange with the location and length values of `aRange`; this constructor is used in cloning the receiver.

Static Methods

fromString

```
public static NSRange fromString(String rangeAsString)
```

Discussion
Creates an NSRange from the string `rangeAsString`, which must be of the form "{*loc*, *len*}" where *loc* is a number representing the starting location of the range and *len* is the range's length. Throws an `IllegalArgumentException` if the string is improperly formatted.

See Also
`toString` (page 476)

Instance Methods

clone

Creates and returns a copy of the receiver.

```
public Object clone()
```

equals

Returns whether `otherObject` is an NSRange and is equal in location and length to the receiver.

```
public boolean equals(Object otherObject)
```

See Also
`isEqualToRange` (page 474)
`isSubrangeOfRange` (page 474)

hashCode

Provides an appropriate hash code useful for storing the receiver in a hash-based data structure.

```
public int hashCode()
```

Discussion
This value is the sum of the receiver's location and length.

intersectsRange

Returns whether the range *aRange* intersects the receiver.

```
public boolean intersectsRange(NSRange aRange)
```

See Also
rangeByIntersectingRange (page 475)

isEmpty

Returns whether the length of the receiver is 0.

```
public boolean isEmpty()
```

See Also
maxRange (page 475)

isEqualToRange

Returns whether the range *aRange* is equal in both location and length to the receiver.

```
public boolean isEqualToRange(NSRange aRange)
```

See Also
equals (page 473)
isSubrangeOfRange (page 474)

isSubrangeOfRange

Returns whether the receiver's endpoints match or fall within those of range *aRange*.

```
public boolean isSubrangeOfRange(NSRange aRange)
```

See Also
intersectsRange (page 474)

length

Returns the length of the receiver from its starting location.

```
public int length()
```

See Also
location (page 475)

location

Returns the starting location of the receiver.

```
public int location()
```

See Also
length (page 474)

locationInRange

Returns whether the location *aLocation* comes after or matches the starting location and comes before the ending location of the receiver.

```
public boolean locationInRange(int aLocation)
```

See Also
intersectsRange (page 474)
location (page 475)

maxRange

Returns the extent of the receiver (its starting location plus its length).

```
public int maxRange()
```

Discussion
This number is 1 greater than the last location in the range.

See Also
isEmpty (page 474)
length (page 474)
location (page 475)

rangeByIntersectingRange

Returns an NSRange that is the intersection of *aRange* and the receiver.

```
public NSRange rangeByIntersectingRange(NSRange aRange)
```

Discussion
If the ranges do not intersect, returns an empty range (see isEmpty (page 474)).

See Also
rangeByUnioningRange (page 476)
subtractRange (page 476)

rangeByUnioningRange

Returns an NSRange that is the union of *aRange* and the receiver (a range constructed from the lowest starting location and the highest ending location of both NSRanges).

```
public NSRange rangeByUnioningRange(NSRange aRange)
```

See Also
rangeByIntersectingRange (page 475)
subtractRange (page 476)

subtractRange

Returns the ranges resulting from the subtraction of *otherRange* from the receiver by modifying the mutable ranges *resultRange1* and *resultRange2* (provided by the caller).

```
public void subtractRange(NSRange otherRange, NSMutableRange resultRange1,
    NSMutableRange resultRange2)
```

Discussion
Either or both of the result ranges might be empty when this method returns.

See Also
isSubrangeOfRange (page 474)

toString

Returns a String representing the receiver's starting location and length.

```
public String toString()
```

Discussion
The string has the form "{*loc*, *len*}", where *loc* is the starting location of the range and *len* is its length.

See Also
fromString (page 473)

Constants

NSRange provides the following constant as a convenience; you can use it to compare values returned by some NSRange methods:

Constant	Description
ZeroRange	An NSRange set to 0 in location and length

NSRangeSpecifier

Inherits from	NSScriptObjectSpecifier : NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Cocoa Scripting Guide

Overview

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). These specifiers are evaluated in the context of the same container object as the range specifier itself.

You don't normally subclass NSRangeSpecifier.

Tasks

Constructors

`NSRangeSpecifier` (page 478)
Returns an NSRangeSpecifier with no data.

Accessing a Range Specifier

`endSpecifier` (page 478)
Returns the object specifier representing the last object of the range.

`setEndSpecifier` (page 479)
Sets the object specifier representing the last object of the range to *endSpec*.

`setStartSpecifier` (page 479)
Sets the object specifier representing the first object of the range to *startSpec*.

`startSpecifier` (page 479)
Returns the object specifier representing the first object of the range.

Constructors

NSRangeSpecifier

Returns an NSRangeSpecifier with no data.

```
public NSRangeSpecifier()
```

Discussion
Do not use this constructor.

Returns an NSRangeSpecifier 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.

```
public NSRangeSpecifier(NSScriptClassDescription classDescription,
    NSScriptObjectSpecifier specifier, String key)
```

Discussion
The receiver's child specifier reference is set to `null`. You use the `setStartSpecifier` (page 479) and `setEndSpecifier` (page 479) methods to set the start and end specifiers for the range.

Returns an NSRangeSpecifier initialized with container specifier *specifier* and key *key*.

```
public NSRangeSpecifier(NSScriptObjectSpecifier specifier, String key)
```

Discussion
The class description of the container is set automatically. You use the `setStartSpecifier` (page 479) and `setEndSpecifier` (page 479) methods to set the start and end specifiers for the range.

Returns an NSRangeSpecifier 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. The start and end specifiers are set based on the *startSpec* and *endSpec* parameters.

```
public NSRangeSpecifier(NSScriptClassDescription classDescription,
    NSScriptObjectSpecifier specifier, String key, NSScriptObjectSpecifier startSpec,
     NSScriptObjectSpecifier endSpec)
```

Discussion
The receiver's child specifier reference is set to `null`.

Instance Methods

endSpecifier

Returns the object specifier representing the last object of the range.

```
public NSScriptObjectSpecifier endSpecifier()
```

setEndSpecifier

Sets the object specifier representing the last object of the range to *endSpec*.

```
public void setEndSpecifier(NSScriptObjectSpecifier endSpec)
```

setStartSpecifier

Sets the object specifier representing the first object of the range to *startSpec*.

```
public void setStartSpecifier(NSScriptObjectSpecifier startSpec)
```

startSpecifier

Returns the object specifier representing the first object of the range.

```
public NSScriptObjectSpecifier startSpecifier()
```

NSRect

Inherits from	Object
Implements	Cloneable
Package:	com.apple.cocoa.foundation
Companion guide	Geometry and Range Utilities

Overview

An NSRect object represents a rectangle. The elemental attributes of a rectangle are its origin (its starting x coordinate and y coordinate) and its size (width and height as measured from the origin). An NSRect with height or width of 0 or less is considered an "empty" rectangle. The methods of NSRect give you access to these values and to computed values, and they allow you to compare rectangles and perform various tests, such as determining whether a point falls in a rectangle. They also convert NSRects between string objects and AWT rectangles, and they yield new NSRects based on intersection, union, inset, offset, and other operations.

Tasks

Constructors

`NSRect` (page 483)

Accessing Elemental Values

`height` (page 485)
Returns the height dimension of the receiver.

`origin` (page 487)
Returns the origin of the receiver, the point from which the dimensions of the receiver are measured.

`size` (page 489)
Returns the size of the receiver.

`width` (page 490)
Returns the width of the receiver.

`x` (page 490)
Returns the origin x coordinate of the receiver.

`y` (page 490)
Returns the origin y coordinate of the receiver.

Accessing Computed Values

`maxX` (page 486)
Returns the farthest extent of the receiver along the x axis in the current coordinate system; or, to put it another way, the x coordinate value of the receiver's right edge.

`maxY` (page 487)
Returns the farthest extent of the receiver along the y axis in the current coordinate system; or, to put it another way, the y coordinate value of the receiver's top edge (or bottom, if the coordinate system is flipped).

`midX` (page 487)
Returns the extent of the receiver along the x axis halfway between its origin x coordinate and the x coordinate of its right edge.

`midY` (page 487)
Returns the extent of the receiver along the y axis halfway between its origin y coordinate and the y coordinate of its upper edge (or bottom edge, if used in a flipped coordinate system).

Testing and Comparing Rectangles

`containsPoint` (page 484)
Returns whether the receiver contains the point *`aPoint`*.

`equals` (page 485)
Returns whether *`otherObject`* is an NSRect and is equal in origin and size to the receiver.

`hashCode` (page 485)
Provides an appropriate hash code useful for storing the receiver in any hash-based data structure.

`intersectsRect` (page 485)
Returns whether the receiver intersects rectangle *`aRect`*.

`isEmpty` (page 486)
Returns whether either dimension (width or height) of the receiver is 0.

`isEqualToRect` (page 486)
Returns whether the NSRect *`aRect`* is equal in origin and size to the receiver.

`isSubrectOfRect` (page 486)
Returns whether the receiver is entirely enclosed by rectangle *`aRect`*.

Deriving Rectangles

`rectByInsettingRect` (page 488)
Returns an NSRect that is the result of insetting the receiver *`horizInset`* units horizontally and *`vertInset`* units vertically from all edges.

`rectByIntersectingRect` (page 488)
Returns an NSRect that is the result of the intersection of the receiver with *`otherRect`*.

`rectByMakingIntegral` (page 488)
Returns an NSRect that is the result of rounding down the receiver's x coordinate and y coordinate to the nearest integer and rounding up the receiver's height and width to the nearest integer.

`rectByOffsettingRect` (page 488)
Returns an NSRect that is the result of offsetting the receiver *`horizInset`* units horizontally and *`vertInset`* units vertically.

`rectByUnioningRect` (page 488)
Returns an NSRect that is the result of the union of the receiver with *`otherRect`*, which is a rectangle that contains the two NSRects combined.

`sliceRect` (page 489)
Makes two smaller rectangles from the receiver and returns them by modifying two mutable rectangles passed in as arguments.

Transforming NSRects

`fromString` (page 484)

`toString` (page 490)
Returns the receiver as converted to a string object.

`toAWTRectangle` (page 490)
Returns the receiver as a AWT Rectangle object.

Copying

`clone` (page 484)
Creates and returns a copy of the receiver.

Constructors

NSRect

```
public NSRect()
```

Discussion
Initializes an empty rectangle (that is, a rectangle with at least one dimension of 0).

```
public NSRect(float x, float y, float w, float h)
```

Discussion
Initializes an NSRect from a starting x coordinate (*x*), a starting y coordinate (*y*), a width value (*w*), and a height value (*h*). If either width or height is 0, initializes an empty rectangle. Throws an `IllegalArgumentException` if any argument is not a valid float value (NaN).

```
public NSRect(NSPoint aPoint, NSSize aSize)
```

Discussion
Initializes an NSRect from an NSPoint object, *aPoint*, and an NSSize object, *aSize*.

```
public NSRect(NSPoint pointOne, NSPoint pointTwo)
```

Discussion
Initializes an NSRect from two NSPoint objects, *pointOne* and *pointTwo*, creating the smallest rectangle whose opposite corners touch the two points.

```
public NSRect(java.awt.Rectangle aRectangle)
```

Discussion
Initializes an NSRect from an AWT Rectangle object, *aRectangle*.

```
public NSRect(NSRect aRectangle)
```

Discussion
Initializes an NSRect from another NSRect object, *aRectangle*; this constructor is used in cloning the receiver.

Static Methods

fromString

```
public static NSRect fromString(String rectAsString)
```

Discussion
Creates an NSRect from the string *rectAsString*, which must be of the form "{{*x*, *y*}, {*w*, *h*}}", where *x* is a float representation of the origin x coordinate, *y* is a float representation of the origin y coordinate, *w* is a float representation of the width, and *h* is a float representation of the height. Throws an `IllegalArgumentException` if the string is improperly formatted.

See Also
`toString` (page 490)

Instance Methods

clone

Creates and returns a copy of the receiver.

```
public Object clone()
```

containsPoint

Returns whether the receiver contains the point *aPoint*.

```
public boolean containsPoint(NSPoint aPoint, boolean isFlipped)
```

Discussion
If *isFlipped* is `true`, the measurements occur in a flipped coordinate system (where the y axis extends down instead of up). If *aPoint* occurs inside or coincides with the receiver's x axis or y axis and falls within the opposite edges of the rectangle, it is considered to be contained by the rectangle.

See Also
`intersectsRect` (page 485)
`isSubrectOfRect` (page 486)

equals

Returns whether *otherObject* is an NSRect and is equal in origin and size to the receiver.

```
public boolean equals(Object otherObject)
```

See Also
`isEqualToRect` (page 486)

hashCode

Provides an appropriate hash code useful for storing the receiver in any hash-based data structure.

```
public int hashCode()
```

Discussion
This value is the sum of the receiver's origin x coordinate, its origin y coordinate, half of its width, and half of its height, rounded to the nearest integer.

height

Returns the height dimension of the receiver.

```
public float height()
```

See Also
`maxY` (page 487)
`midY` (page 487)
`size` (page 489)
`width` (page 490)

intersectsRect

Returns whether the receiver intersects rectangle *aRect*.

```
public boolean intersectsRect(NSRect aRect)
```

Discussion
Alignment of outer edges is not considered an intersection. Returns `false` if either NSRect is an empty rectangle.

See Also
`containsPoint` (page 484)
`isSubrectOfRect` (page 486)
`rectByIntersectingRect` (page 488)

isEmpty

Returns whether either dimension (width or height) of the receiver is 0.

```
public boolean isEmpty()
```

isEqualToRect

Returns whether the NSRect *aRect* is equal in origin and size to the receiver.

```
public boolean isEqualToRect(NSRect aRect)
```

See Also
`equals` (page 485)

isSubrectOfRect

Returns whether the receiver is entirely enclosed by rectangle *aRect*.

```
public boolean isSubrectOfRect(NSRect aRect)
```

Discussion
Coincidence of x axes and y axes or the edges opposite the x axis or y axis are acceptable. Returns `false` if either NSRect is an empty rectangle. By reversing receiver and argument, you can use this method as a virtual "`containsRect`".

See Also
`containsPoint` (page 484)
`intersectsRect` (page 485)

maxX

Returns the farthest extent of the receiver along the x axis in the current coordinate system; or, to put it another way, the x coordinate value of the receiver's right edge.

```
public float maxX()
```

Discussion
This value is the sum of the receiver's origin x coordinate and its width.

See Also
maxY (page 487)

maxY

Returns the farthest extent of the receiver along the y axis in the current coordinate system; or, to put it another way, the y coordinate value of the receiver's top edge (or bottom, if the coordinate system is flipped).

```
public float maxY()
```

Discussion
This value is the sum of the receiver's origin y coordinate and its height.

See Also
maxX (page 486)

midX

Returns the extent of the receiver along the x axis halfway between its origin x coordinate and the x coordinate of its right edge.

```
public float midX()
```

See Also
midY (page 487)

midY

Returns the extent of the receiver along the y axis halfway between its origin y coordinate and the y coordinate of its upper edge (or bottom edge, if used in a flipped coordinate system).

```
public float midY()
```

See Also
midX (page 487)

origin

Returns the origin of the receiver, the point from which the dimensions of the receiver are measured.

```
public NSPoint origin()
```

See Also
x (page 490)
y (page 490)

rectByInsettingRect

Returns an NSRect that is the result of insetting the receiver *horizInset* units horizontally and *vertInset* units vertically from all edges.

```
public NSRect rectByInsettingRect(float horizInset, float vertInset)
```

See Also
rectByOffsettingRect (page 488)

rectByIntersectingRect

Returns an NSRect that is the result of the intersection of the receiver with *otherRect*.

```
public NSRect rectByIntersectingRect(NSRect otherRect)
```

Discussion
Returns ZeroRect if either receiver or *otherRect* is an empty rectangle or the two NSRects do not intersect at all.

See Also
rectByUnioningRect (page 488)

rectByMakingIntegral

Returns an NSRect that is the result of rounding down the receiver's x coordinate and y coordinate to the nearest integer and rounding up the receiver's height and width to the nearest integer.

```
public NSRect rectByMakingIntegral()
```

Discussion
The resulting rectangle thus completely encloses the original. Returns ZeroRect if the receiver is an empty rectangle.

See Also
toAWTRectangle (page 490)

rectByOffsettingRect

Returns an NSRect that is the result of offsetting the receiver *horizInset* units horizontally and *vertInset* units vertically.

```
public NSRect rectByOffsettingRect(float horizOffset, float vertOffset)
```

Discussion
This method affects the origin only, while rectByInsettingRect (page 488) affects all edges.

rectByUnioningRect

Returns an NSRect that is the result of the union of the receiver with *otherRect*, which is a rectangle that contains the two NSRects combined.

```
public NSRect rectByUnioningRect(NSRect otherRect)
```

Discussion
Returns `ZeroRect` if both receiver and *`otherRect`* are empty rectangles. If one of the two rectangles, the receiver and *`otherRect`*, is an empty rectangle, a copy of the other rectangle is returned.

See Also
`rectByIntersectingRect` (page 488)

size

Returns the size of the receiver.

```
public NSSize size()
```

Discussion
The NSSize object encapsulates the width and height of the receiver.

See Also
`height` (page 485)
`origin` (page 487)
`width` (page 490)

sliceRect

Makes two smaller rectangles from the receiver and returns them by modifying two mutable rectangles passed in as arguments.

```
public void sliceRect(float thickness, int edge, NSMutableRect rect1, NSMutableRect
    rect2)
```

Discussion
One of these NSMutableRects has the width or height thickness as determined by the constant *`edge`*. One modified rectangle is returned in *`rect1`* and the other in *`rect2`*; which rectangle goes where is also determined by *`edge`*. Here is a summary of how the *`edge`* constant and the other arguments interact:

Constant	Description
`MinXEdge`	The rectangle is sliced vertically, and the rectangle with the width of *`thickness`* is placed in *`rect1`*.
`MaxXEdge`	The rectangle is sliced horizontally, and the rectangle with the height of *`thickness`* is placed in *`rect2`*.
`MinYEdge`	The rectangle is sliced horizontally, and the rectangle with the height of *`thickness`* is placed in *`rect1`*.
`MaxYEdge`	The rectangle is sliced vertically, and the rectangle with the width of *`thickness`* is placed in *`rect2`*.

If *`thickness`* is greater than the width or height of the receiver (as determined by *`edge`*), it is made the same width or height as the receiver. This method does not affect the receiver.

toAWTRectangle

Returns the receiver as a AWT Rectangle object.

```
public java.awt.Rectangle toAWTRectangle()
```

Discussion
This method calls `rectByMakingIntegral` (page 488) to round the receiver's float values to appropriate integers.

See Also
`toString` (page 490)

toString

Returns the receiver as converted to a string object.

```
public String toString()
```

Discussion
The string has the form of "{{*x*, *y*}, {*w*, *h*}}", where *x* is the origin x coordinate, *y* is the origin y coordinate, *w* is the width, and *h* is the height.

See Also
`fromString` (page 484)

width

Returns the width of the receiver.

```
public float width()
```

See Also
`height` (page 485)
`size` (page 489)

x

Returns the origin x coordinate of the receiver.

```
public float x()
```

See Also
`origin` (page 487)
`y` (page 490)

y

Returns the origin y coordinate of the receiver.

```
public float y()
```

See Also
origin (page 487)
x (page 490)

Constants

NSRect provides the following constant as a convenience; you can use it to compare values returned by many NSRect methods:

Constant	Description
ZeroRect	An NSRect set to 0 in width and height.

The following constants can be used to specify window edges:

Constant	Description
MinXEdge	The left edge of a window.
MinYEdge	The bottom edge of a window.
MaxXEdge	The right edge of a window.
MaxYEdge	The top edge of a window.

NSRelativeSpecifier

Inherits from	NSScriptObjectSpecifier : NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Cocoa Scripting Guide

Overview

Specifies an object in a collection by its position relative to another object. You don't normally subclass NSRelativeSpecifier.

Tasks

Constructors

`NSRelativeSpecifier` (page 494)
Returns an NSRelativeSpecifier with no data.

Accessing a Relative Specifier

`baseSpecifier` (page 494)
Returns a specifier for the base object—the object to which the relative specifier is related.

`relativePosition` (page 495)
Returns the relative position encapsulated by the receiver.

`setBaseSpecifier` (page 495)
Sets the specifier for the base object, *`baseSpecifier`*—the object to which the relative specifier is related.

`setRelativePosition` (page 495)
Sets the relative position encapsulated by the receiver.

Constructors

NSRelativeSpecifier

Returns an NSRelativeSpecifier with no data.

```
public NSRelativeSpecifier()
```

Discussion
Do not use this constructor.

Returns an NSRelativeSpecifier 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.

```
public NSRelativeSpecifier(NSScriptClassDescription classDescription,
    NSScriptObjectSpecifier specifier, String key)
```

Discussion
The receiver's child specifier reference is set to `null`. You use the `setBaseSpecifier` (page 495) method to set the base specifier and the `setRelativePosition` (page 495) method to set the relative position encapsulated by the object.

Returns an NSRelativeSpecifier initialized with container specifier *specifier* and key *key*.

```
public NSRelativeSpecifier(NSScriptObjectSpecifier specifier, String key)
```

Discussion
The class description of the container is set automatically. You use the `setBaseSpecifier` (page 495) method to set the base specifier and the `setRelativePosition` (page 495) method to set the relative position encapsulated by the object.

Returns an NSRelativeSpecifier 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. The relative position is set according to the *relPos* parameter and the base specifier according to the *baseSpecifier* parameter.

```
public NSRelativeSpecifier(NSScriptClassDescription classDescription,
    NSScriptObjectSpecifier specifier, String key, int relPos,
    NSScriptObjectSpecifier baseSpecifier)
```

Discussion
The receiver's child specifier reference is set to `null`.

Instance Methods

baseSpecifier

Returns a specifier for the base object—the object to which the relative specifier is related.

```
public NSScriptObjectSpecifier baseSpecifier()
```

relativePosition

Returns the relative position encapsulated by the receiver.

```
public int relativePosition()
```

Discussion
See "Constants" (page 495) for a list of possible return values.

setBaseSpecifier

Sets the specifier for the base object, *baseSpecifier*—the object to which the relative specifier is related.

```
public void setBaseSpecifier(NSScriptObjectSpecifier baseSpecifier)
```

setRelativePosition

Sets the relative position encapsulated by the receiver.

```
public void setRelativePosition(int relPos)
```

Discussion
See "Constants" (page 495) for a list of possible values for *relPos*.

Constants

The following constants are defined by NSRelativeSpecifier and are used by `relativePosition` (page 495) and `setRelativePosition` (page 495):

Constant	Description
`RelativeAfter`	Specifies a position after another object.
`RelativeBefore`	Specifies a position before another object.

NSRunLoop

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Run Loops

Overview

The NSRunLoop class declares the programmatic interface to objects that manage input sources. An NSRunLoop processes input for sources such as mouse and keyboard events from the window system, NSPorts, NSTimers, and NSConnections.

In general, your application does not need to either create or explicitly manage NSRunLoop objects. Each NSThread, including the application's main thread, has an NSRunLoop object automatically created for it. If you need to access the current thread's default run loop, you do so with the class method `currentRunLoop` (page 499).

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

Constructors

`NSRunLoop` (page 499)
Creates and returns a new NSRunLoop instance.

Accessing the Current Run Loop

`allModes` (page 500)
Returns an array of strings of all the modes defined in the current run loop.

`currentRunLoop` (page 499)
Returns the NSRunLoop for the current thread.

`currentMode` (page 501)
Returns the current input mode.

`limitDateForMode` (page 501)
Performs one pass through the run loop in mode *mode* and returns the date at which the next timer is scheduled to fire.

Managing Timers

`addTimerForMode` (page 500)
Registers the timer *aTimer* with input mode *mode*.

`containsTimerForMode` (page 500)
Returns `true` if *aTimer* is a member of input mode *mode*.

`removeTimerForMode` (page 502)
Removes *aTimer* from input mode *mode*.

`timersForMode` (page 503)
Returns an empty array.

Managing Ports

`addPortForMode` (page 499)
Adds *aPort* to be monitored by the receiver in the input mode *mode*.

`containsPortForMode` (page 500)
Returns `true` if *aPort* is a member of input mode *mode*.

`portsForMode` (page 502)
Returns an empty array.

`removePortForMode` (page 502)
Removes *aPort* from the list of ports being monitored by the receiver in input mode *mode*.

Running a Loop

`run` (page 502)
Runs the loop in `DefaultRunLoopMode` by repeatedly invoking `runModeBeforeDate` (page 502) until all input sources have been removed.

`runModeBeforeDate` (page 502)
Runs the loop once, blocking for input in mode *mode* until *limitDate*.

`runModeUntilDate` (page 503)
Runs the loop in mode *mode* by repeatedly invoking `runModeBeforeDate` (page 502) until *limitDate* or until all input sources have been removed.

`acceptInputForMode` (page 499)
Runs the loop once, blocking for input in mode *mode* until *limitDate*.

Sending Messages

`performSelectorWithOrder` (page 501)
Schedules the sending of an *aSelector* message.

`cancelPerformSelectorWithOrder` (page 500)
Cancels the sending of a message previously scheduled using `performSelectorWithOrder` (page 501).

Constructors

NSRunLoop

Creates and returns a new NSRunLoop instance.

```
public NSRunLoop()
```

Discussion
Only one run loop is allowed per thread, so if a run loop already exists in the current thread, this method returns `null`. Use `currentRunLoop` (page 499) instead.

Static Methods

currentRunLoop

Returns the NSRunLoop for the current thread.

```
public static NSRunLoop currentRunLoop()
```

Discussion
If a run loop does not exist in the thread, one is created and returned.

See Also
`currentMode` (page 501)

Instance Methods

acceptInputForMode

Runs the loop once, blocking for input in mode *mode* until *limitDate*.

```
public void acceptInputForMode(String mode, NSDate limitDate)
```

See Also
`runModeBeforeDate` (page 502)

addPortForMode

Adds *aPort* to be monitored by the receiver in the input mode *mode*.

```
public void addPortForMode(NSPort aPort, String mode)
```

Discussion
The receiver maintains a count of the number of ports added, and the same number must be removed.

See Also
`removePortForMode` (page 502)

addTimerForMode

Registers the timer *aTimer* with input mode *mode*.

```
public void addTimerForMode(NSTimer aTimer, String mode)
```

Discussion
The run loop causes the timer to fire on or after its scheduled fire date. Timers have a message associated with them. When a timer fires, it sends its message to the appropriate object. To remove a timer from a mode, send the `invalidate` (page 612) message to the timer.

allModes

Returns an array of strings of all the modes defined in the current run loop.

```
public NSArray allModes()
```

cancelPerformSelectorWithOrder

Cancels the sending of a message previously scheduled using `performSelectorWithOrder` (page 501).

```
public void cancelPerformSelectorWithOrder(NSSelector aSelector, Object target,
    Object anArgument)
```

Discussion
The *aSelector* message with argument *anArgument* will not be sent to *target*.

containsPortForMode

Returns `true` if *aPort* is a member of input mode *mode*.

```
public boolean containsPortForMode(NSPort aPort, String mode)
```

Discussion
Returns `false` otherwise.

containsTimerForMode

Returns `true` if *aTimer* is a member of input mode *mode*.

```
public boolean containsTimerForMode(NSTimer aTimer, String mode)
```

Discussion
Returns `false` otherwise.

currentMode

Returns the current input mode.

```
public String currentMode()
```

Discussion
The current mode is set by the methods that run the run loop, such as `acceptInputForMode` (page 499) and `runModeBeforeDate` (page 502).

The `currentMode` method returns the current input mode ONLY while the receiver is running. Otherwise, `currentMode` returns `null`.

See Also
`currentRunLoop` (page 499)
`limitDateForMode` (page 501)
`run` (page 502)
`runModeUntilDate` (page 503)

limitDateForMode

Performs one pass through the run loop in mode *mode* and returns the date at which the next timer is scheduled to fire.

```
public NSDate limitDateForMode(String mode)
```

Discussion
Returns `null` if there are no input sources for this mode. 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.

performSelectorWithOrder

Schedules the sending of an *aSelector* message.

```
public void performSelectorWithOrder(NSSelector aSelector, Object target, Object
    anArgument, int order, NSArray modes)
```

Discussion
The *aSelector* message is sent to *target* with argument *anArgument* at the start of the next run loop iteration in any of the input modes specified in *modes*. *order* assigns a priority to the messages. If multiple messages are scheduled to be sent, the messages with a lower order value are sent before messages with a higher order value.

This method returns before the *aSelector* message is sent. The *aSelector* method should not have a significant return value and should take a single argument of type Object.

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.

See Also
cancelPerformSelectorWithOrder (page 500)

portsForMode

Returns an empty array.

```
public NSArray portsForMode(String aString)
```

removePortForMode

Removes *aPort* from the list of ports being monitored by the receiver in input mode *mode*.

```
public void removePortForMode(NSPort aPort, String mode)
```

Discussion
The receiver maintains a count of the ports added, and the same number of ports must be removed. Ports are automatically removed from input modes if they are detected to be invalid.

See Also
addPortForMode (page 499)

removeTimerForMode

Removes *aTimer* from input mode *mode*.

```
public void removeTimerForMode(NSTimer aTimer, String mode)
```

See Also
addTimerForMode (page 500)

run

Runs the loop in DefaultRunLoopMode by repeatedly invoking runModeBeforeDate (page 502) until all input sources have been removed.

```
public void run()
```

Discussion
If there are no input sources in the run loop, it exits immediately.

See Also
runModeUntilDate (page 503)

runModeBeforeDate

Runs the loop once, blocking for input in mode *mode* until *limitDate*.

```
public boolean runModeBeforeDate(String mode, NSDate limitDate)
```

Discussion
The method returns after either the first input is processed or *limitDate* is reached. Returns `false` without starting the run loop if there are no input sources in *mode*; otherwise returns `true`.

See Also
`run` (page 502)
`runModeUntilDate` (page 503)

runModeUntilDate

Runs the loop in mode *mode* by repeatedly invoking `runModeBeforeDate` (page 502) until *limitDate* or until all input sources have been removed.

```
public boolean runModeUntilDate(String mode, NSDate limitDate)
```

Discussion
If there are no input sources in the run loop, it exits immediately, returning `false`.

See Also
`run` (page 502)

timersForMode

Returns an empty array.

```
public NSArray timersForMode(String mode)
```

Constants

Cocoa defines the following run loop modes:

Input mode	Description
`DefaultRunLoopMode`	Use this mode to deal with input sources other than NSConnections. This mode is the most commonly used run-loop mode.
`NSApplication.Modal-PanelRunLoopMode`	Use this mode when waiting for input from a modal panel, such as NSSavePanel or NSOpenPanel.
`NSApplication.Event-TrackingRunLoopMode`	Use this mode for event-tracking loops.

NSRuntime

Inherits from	Object
Package:	com.apple.cocoa.foundation
Companion guide	Code Loading Programming Topics for Cocoa

Overview

The NSRuntime class provides methods to load Objective-C dynamic libraries into Java applications. It manages a list of directories that are searched when you attempt to load a library without providing its absolute path.

Libraries must have an initialization function named *basename*`Initialization` where *basename* is the name of the library without the standard `lib` and `.dylib` prefix and suffix. For example, when creating a library named `libMyCode.dylib`, create a function named `MyCodeInitialization` to initialize the library when it gets loaded. The function takes no arguments.

Tasks

Working with Libraries

`addPathToLibrarySearchPaths` (page 505)
Adds *path* to the list of directories searched when loading libraries with `loadLibrary` (page 506).

`librarySearchPaths` (page 506)
Returns the list of directories searched by `loadLibrary` (page 506).

`loadLibrary` (page 506)
Loads the Objective-C dynamic library *name* into the application.

`nextRootPath` (page 506)
Returns the value of the `NEXT_ROOT` environment variable.

Static Methods

addPathToLibrarySearchPaths

Adds *path* to the list of directories searched when loading libraries with `loadLibrary` (page 506).

```
public static void addPathToLibrarySearchPaths(String path)
```

Discussion
path must be an absolute path.

librarySearchPaths

Returns the list of directories searched by `loadLibrary` (page 506).

```
public static String[] librarySearchPaths()
```

Discussion
The initial list includes the `Resources` directory of the application's bundle, `/usr/local/lib/java`, and `/usr/lib/java`. Additional paths are added to the list using `addPathToLibrarySearchPaths` (page 505).

loadLibrary

Loads the Objective-C dynamic library *name* into the application.

```
public static void loadLibrary(String name)
```

Discussion
name can be either the absolute path to the library or just the library name. If just the library name is given, either with or without the standard prefix `lib` or suffix `.dylib`, NSRuntime searches through the directories returned by `librarySearchPaths` (page 506) until it finds the library. After the library is loaded, NSRuntime initializes the library by calling the function *basename*`Initialization` where *basename* is the library's name with the prefix and suffix stripped off.

If the library is not found or if the library lacks an initialization function, the application exits with an `UnsatisfiedLinkError` error.

nextRootPath

Returns the value of the `NEXT_ROOT` environment variable.

```
public static String nextRootPath()
```

Discussion
This variable is no longer used.

NSScriptClassDescription

Inherits from	NSClassDescription : NSObject
Package:	com.apple.cocoa.foundation
Companion guides	Cocoa Scripting Guide Key-Value Coding Programming Guide

Overview

NSScriptClassDescription is one of the classes in Cocoa's scripting support that encapsulates an application's scripting information.

Encapsulating Scripting Information

Scripting information includes descriptions of the attributes and relationships of the scriptable objects in an application and of the commands the application supports. This information specifies the operations a scripter can perform to control the application from a script. For example, a scriptable document object for a drawing application might support attributes (one of each) such as `filename` and `file type`, and relationships (possibly many of each) such as collections of `circles`, `rectangles`, and `lines`.

Scripting information is provided in one of two standard formats: the script suite format or the sdef (scripting definition) format. For more information on these formats, see NSScriptSuiteRegistry (page 549).

Scripting information is collected automatically by an instance of NSScriptSuiteRegistry, which creates an NSScriptClassDescription for each class description it finds and caches these objects in memory.

As with many of the classes in Cocoa's built-in scripting support, your application may never need to directly access NSScriptClassDescription. One case where you might need access to a class description is if you override `objectSpecifier` in a scriptable class. For more information on how to do this, see the examples in the Class Description for NSScriptObjectSpecifier (page 539).

Another case where your application may need access to class description information is if you override `indicesOfObjectsByEvaluatingWithContainer` in a specifier class.

Although you can subclass NSScriptClassDescription, it is unlikely that you would need to.

Specifying Subcontainers Implicitly

Cocoa Scripting provides support for **implicitly specified subcontainers**. Without this feature, an AppleScript writer would have to write statements like the following

```
fourth word of text of front document
```

to refer to that specific word in, for example, a TextEdit document. Developers of scripter-friendly applications will want to save scripters the trouble of writing full references to objects in situations where part of the reference can be safely assumed. To continue the example, the reference

```
fourth word of front document
```

could be a perfectly acceptable reference to the same word in a TextEdit document, since `text` is the obvious container of `words` in a document. That is, it's appropriate to allow `of text` to be implicitly specified by context, instead of explicitly specified in the script.

To support implicitly specified subcontainers, a new `DefaultSubcontainerAttribute` entry is now allowed in class dictionaries within `.scriptSuite` property list files. The value of the entry must be the key of one of the entries in the Attributes or ToOneRelationships dictionary of the class. For example, the TextEdit application's `TextEdit.scriptSuite` file includes a new entry in its document dictionary to support implicitly specified text storage. The following is an excerpt from that file:

```
{
    [...]
    "Classes" = {
        [...]
        "Document" = {
            "Superclass" = "NSCoreSuite.NSDocument";
            "AppleEventCode" = "docu";
            "DefaultSubcontainerAttribute" = "textStorage";
            "ToOneRelationships" = {
                "textStorage" = {
                    "Type" = "NSTextStorage";
                    "AppleEventCode" = "ctxt";
                };
            };
        };
    };
}
```

If an attribute or to-one relation is declared to be the default subcontainer for a container class, an appropriate NSPropertySpecifier will be inserted into the object specifier containment chain when a reference form using that container class would otherwise be invalid.

Tasks

Constructors

`NSScriptClassDescription` (page 510)
Returns an NSScriptClassDescription with no data.

Getting an NSScriptClassDescription

`classDescriptionForKey` (page 511)
Returns the NSScriptClassDescription for the type of the attribute or relationship identified by *key*.

`superclassDescription` (page 513)
Returns the NSScriptClassDescription object for the superclass of the receiver's class or `null` if the class has no superclass.

Getting Basic Information

`className` (page 511)
Returns the class name associated with the receiver.

`defaultSubcontainerAttributeKey` (page 511)
Returns the value of the DefaultSubcontainerAttribute entry of the class dictionary from which the NSScriptClassDescription instance was instantiated.

`isLocationRequiredToCreateForKey` (page 511)

`suiteName` (page 513)
Returns the name of the receiver's suite.

Getting and Comparing Apple Event Codes

`appleEventCode` (page 510)
Returns the four-character Apple event code associated with the receiver.

`appleEventCodeForKey` (page 510)
Returns the four-character Apple event code associated with the attribute or relationship identified by *`key`* in the receiver or, if no such attribute or relationship exists in the receiver, in the class description for the receiver's superclass.

`matchesAppleEventCode` (page 512)
Returns a Boolean value that indicates whether the receiver's primary four-character Apple event code or any of its secondary codes (`"synonyms"`) matches *`code`*.

Getting Attribute and Relationship Information

`isReadOnlyKey` (page 512)
Returns a Boolean value that indicates whether the attribute, one-to-one relationship, or one-to-many relationship identified by *`key`* is read-only.

`keyWithAppleEventCode` (page 512)
Returns the key of an attribute, one-to-one relationship, or one-to-many relationship identified by the given four-character Apple event code, *`code`*.

`typeForKey` (page 513)
Returns the type of the attribute or relationship identified by *`key`*, such as "NSString".

Getting Command Information

`selectorForCommand` (page 513)
Returns the selector associated with command description *`commandDesc`*.

`supportsCommand` (page 513)
Returns a Boolean value that indicates whether the receiver or the NSScriptClassDescription of any superclass of the receiver's class supports the command described by *`commandDesc`* among its supported commands.

Constructors

NSScriptClassDescription

Returns an NSScriptClassDescription with no data.

```
public NSScriptClassDescription()
```

Discussion
Do not use this constructor.

Returns an NSScriptClassDescription object initialized for suite *`suiteName`* and class *`className`* and encapsulating the information in *`descriptions`*.

```
public NSScriptClassDescription(String suiteName, String className, NSDictionary
    descriptions)
```

Discussion
This information includes attributes, relationships of various kinds, and Apple event code synonyms. Registers `this` with the application's NSScriptSuiteRegistry by Apple event code and also registers each key from the *`descriptions`* dictionary by Apple event code. Returns `null` if the event code value for the class description itself is missing or is not a string. Also returns `null` if the superclass name or any of the subdictionaries of descriptions are not of the right type.

Instance Methods

appleEventCode

Returns the four-character Apple event code associated with the receiver.

```
public int appleEventCode()
```

See Also
`appleEventCodeForKey` (page 510)
`matchesAppleEventCode` (page 512)

appleEventCodeForKey

Returns the four-character Apple event code associated with the attribute or relationship identified by *`key`* in the receiver or, if no such attribute or relationship exists in the receiver, in the class description for the receiver's superclass.

```
public int appleEventCodeForKey(String key)
```

Discussion
Returns 0 if the attribute or relationship doesn't exist in the class description of any superclass.

See Also
`appleEventCode` (page 510)
`matchesAppleEventCode` (page 512)

classDescriptionForKey

Returns the NSScriptClassDescription for the type of the attribute or relationship identified by *key*.

```
public NSScriptClassDescription classDescriptionForKey(String key)
```

Discussion
Returns `null` if no type is assigned or if no NSScriptClassDescription object exists for the type.

See Also
`superclassDescription` (page 513)

className

Returns the class name associated with the receiver.

```
public String className()
```

See Also
`suiteName` (page 513)

defaultSubcontainerAttributeKey

Returns the value of the DefaultSubcontainerAttribute entry of the class dictionary from which the NSScriptClassDescription instance was instantiated.

```
public String defaultSubcontainerAttributeKey()
```

Discussion
Returns `null` if the there was no such entry.

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

isLocationRequiredToCreateForKey

```
public boolean isLocationRequiredToCreateForKey(String toManyRelationshipKey)
```

Discussion
Returns `true` if creation of objects for the relationship specified by the key, in containers of the class described by the NSScriptClassDescription, requires an explicitly specified insertion location. For example, NSMakeCommand uses this method to determine whether or not a specific Make command must have an `at` parameter.

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

isReadOnlyKey

Returns a Boolean value that indicates whether the attribute, one-to-one relationship, or one-to-many relationship identified by *key* is read-only.

```
public boolean isReadOnlyKey(String key)
```

Discussion
If the receiver does not have an attribute or relationship for *key*, the method queries the superclass's NSScriptClassDescription (if such exists) for this value. Returns `false` if the receiver or any superclass NSScriptClassDescription does not have a property for *key*.

See Also
`keyWithAppleEventCode` (page 512)
`typeForKey` (page 513)

keyWithAppleEventCode

Returns the key of an attribute, one-to-one relationship, or one-to-many relationship identified by the given four-character Apple event code, *code*.

```
public String keyWithAppleEventCode(int code)
```

Discussion
If it cannot find a matching key, it checks its superclasses. Returns `null` if it cannot find any such attribute or relationship.

See Also
`isReadOnlyKey` (page 512)
`typeForKey` (page 513)

matchesAppleEventCode

Returns a Boolean value that indicates whether the receiver's primary four-character Apple event code or any of its secondary codes (`"synonyms"`) matches *code*.

```
public boolean matchesAppleEventCode(int code)
```

See Also
`appleEventCode` (page 510)
`appleEventCodeForKey` (page 510)

selectorForCommand

Returns the selector associated with command description *commandDesc*.

```
public NSSelector selectorForCommand(NSScriptCommandDescription commandDesc)
```

Discussion
If the receiver does not have a selector for *commandDesc*, the method checks if the any of the superclass NSScriptClassDescriptions (if such exist) has a selector. Returns `NULL` if no selector is found for the receiver or any superclass NSScriptClassDescription.

See Also
`supportsCommand` (page 513)

suiteName

Returns the name of the receiver's suite.

```
public String suiteName()
```

See Also
`className` (page 511)

superclassDescription

Returns the NSScriptClassDescription object for the superclass of the receiver's class or `null` if the class has no superclass.

```
public NSScriptClassDescription superclassDescription()
```

Discussion
The superclass's NSScriptClassDescription can be in the same suite as the receiver or in a different suite.

See Also
`classDescriptionForKey` (page 511)

supportsCommand

Returns a Boolean value that indicates whether the receiver or the NSScriptClassDescription of any superclass of the receiver's class supports the command described by *commandDesc* among its supported commands.

```
public boolean supportsCommand(NSScriptCommandDescription commandDesc)
```

See Also
`selectorForCommand` (page 513)

typeForKey

Returns the type of the attribute or relationship identified by *key*, such as "NSString".

```
public String typeForKey(String key)
```

Discussion
The attributes and relationships of the receiver are searched first, then the attributes and relationships of the NSScriptClassDescription of any superclass (if such exist). Returns `null` if no type is found.

See Also
`isReadOnlyKey` (page 512)
`keyWithAppleEventCode` (page 512)

NSScriptCoercionHandler

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Cocoa Scripting Guide

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

Constructors

`NSScriptCoercionHandler` (page 516)
Returns an NSScriptCoercionHandler with no data.

Accessing the Application's Handler

`sharedCoercionHandler` (page 516)
Returns the shared NSScriptCoercionHandler for the application.

Working with Handlers

`coerceValueToClass` (page 516)
Returns an object of the class *`toClass`* representing the value specified by *`value`*.

`registerCoercer` (page 516)
Registers *`coercer`* to handle coercions (conversions) from class *`fromClass`* to class *`toClass`*.

Constructors

NSScriptCoercionHandler

Returns an NSScriptCoercionHandler with no data.

```
public NSScriptCoercionHandler()
```

Discussion
Do not use this constructor.

Static Methods

sharedCoercionHandler

Returns the shared NSScriptCoercionHandler for the application.

```
public static NSScriptCoercionHandler sharedCoercionHandler()
```

Instance Methods

coerceValueToClass

Returns an object of the class *toClass* representing the value specified by *value*.

```
public Object coerceValueToClass(Object value, Class toClass)
```

Discussion
Returns `null` if an error occurs. The object is autoreleased, so you must retain it until you are finished with it.

registerCoercer

Registers *coercer* to handle coercions (conversions) from class *fromClass* to class *toClass*.

```
public void registerCoercer(Object coercer, NSSelector selector, Class fromClass,
     Class toClass)
```

Discussion
selector should take two arguments. The first is the value to be converted. The second is the class to convert it to. *coercer* should typically be a class object and *selector* a factory method.

NSScriptCommand

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Cocoa Scripting Guide

Overview

An 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 NSScriptCommand) 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 523)).

The process of transforming an Apple event into a script command depends on use of the application's scripting information, which is specified by either an sdef (script definition) file or one or more script suites. In its scripting information, the application describes the scripting commands, properties, and classes it supports. The documentation for NSScriptCommandDescription (page 529) and NSScriptSuiteRegistry (page 549) provides more information about how scripting information is used.

If the Apple event has any arguments, the script command encapsulates them. The receivers and arguments can initially be NSScriptObjectSpecifiers, in which case they are evaluated, and the command uses the resulting object or objects. (In the example above, `word 1 of the second document` evaluates to an argument, and `word 5 of the front document` evaluates to the receiver of the command.)

An NSScriptCommand also ensures that the receivers, the selector, and the arguments exist and are valid before executing the command. It consults NSScriptClassDescription (page 507) for this "sanity check." When it executes a command, it checks whether the receiver has designated a method to handle this command in the `SupportedCommands` section of the receiver's suite definition. If so, the command invokes this method; otherwise, it invokes its own `performDefaultImplementation` (page 523) method.

As part of Cocoa's standard scripting implementation, NSScriptCommand and its subclasses can handle the default command set in the Core (or Standard) suite for most applications without any subclassing. The Core suite includes the `copy`, `count`, `create`, `delete`, `exists`, `get`, `move`, `set`, `open`, `close`, and `print` commands. However, if your scriptable application, framework, or bundle has special requirements for any of these commands, you can do one of two things: supply a method to handle the command and describe the command in the scripting information you supply for the application; or create a subclass of NSScriptCommand or one of its subclasses and override `performDefaultImplementation` (page 523). For more information on working with script commands, see "Script Commands in Cocoa Scripting".

Your application most likely calls methods of NSScriptCommand when a command handler method needs to extract the command arguments.

Tasks

Constructors

`NSScriptCommand` (page 520)
Returns an NSScriptCommand with no data.

Obtaining the Current Command

`currentCommand` (page 520)
If a command is being executed in the current thread by Cocoa Scripting's built-in Apple event handling, return the command.

Obtaining the Apple Event

`appleEvent` (page 521)
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 Commands

`executeCommand` (page 522)
Executes the command if it is valid and returns the result, if any.

`performDefaultImplementation` (page 523)
Overridden by subclasses to provide a default implementation for the command represented by the receiver.

Getting and Setting Receivers

`evaluatedReceivers` (page 522)
Returns the object or objects to which the command is to be sent (called both the “receivers” or “targets” of script commands).

`receiversSpecifier` (page 523)
Returns the object specifier that, when evaluated, yields the receiver or receivers of the command.

`setReceiversSpecifier` (page 525)
Sets the object specifier to *`receiversSpec`* that, when evaluated, indicates the receiver or receivers of the command.

Getting and Setting Arguments

`arguments` (page 521)
Returns the arguments of the command.

`evaluatedArguments` (page 522)
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 525)
Sets the arguments of the command to *`args`*.

Getting and Setting Parameters

`directParameter` (page 521)
Returns the object that corresponds to the `keyDirectObject` parameter of the Apple event from which the receiver derives, or `null` if the event doesn't contain a direct parameter.

`setDirectParameter` (page 525)
Sets the object to *`directParameter`* that corresponds to the `keyDirectObject` parameter of the Apple event from which the receiver derives. You don't normally override this method.

Getting Information About the Command

`commandDescription` (page 521)
Returns the command description for the command.

`isWellFormed` (page 523)
Returns a Boolean value that indicates whether the receiver is well formed according to its command description.

Managing Script Execution Problems

`scriptErrorNumber` (page 524)
Returns the script error number, if any, associated with execution of the command.

`setScriptErrorNumber` (page 526)
Sets a script error number to *`errorNumber`* that is associated with the execution of the command.

`scriptErrorString` (page 524)
Returns the script error string, if any, associated with execution of the command.

`setScriptErrorString` (page 526)
Sets a script error string to *`errorString`* that is associated with execution of the command.

Suspending and Resuming Commands

`suspendExecution` (page 526)
Suspends the execution of the receiver, 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()`).

`resumeExecutionWithResult` (page 524)
If a successful, unmatched, invocation of `suspendExecution` (page 526) has been made, resume the execution of the command.

Constructors

NSScriptCommand

Returns an NSScriptCommand with no data.

```
public NSScriptCommand()
```

Discussion
Do not use this constructor.

Returns an NSScriptCommand object initialized with the command description *commandDesc*.

```
public NSScriptCommand(NSScriptCommandDescription commandDesc)
```

Discussion
To make this command object usable, you must set its receiving objects and arguments (if any) after invoking this method.

See Also
`setArguments` (page 525)
`setReceiversSpecifier` (page 525)

Static Methods

currentCommand

If a command is being executed in the current thread by Cocoa Scripting's built-in Apple event handling, return the command.

```
public static 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 522) message at this instant as the result of the dispatch of an Apple event. Returns `null` otherwise. `setScriptErrorNumber` (page 526) and `setScriptErrorString` (page 526) 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 `null`.

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

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.

```
public NSAppleEventDescriptor appleEvent()
```

Discussion
The effects of mutating this descriptor are undefined, although it may be copied.

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

arguments

Returns the arguments of the command.

```
public NSDictionary arguments()
```

Discussion
If there are no arguments, returns an empty NSDictionary. 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 522) instead.

See Also
`setArguments` (page 525)

commandDescription

Returns the command description for the command.

```
public NSScriptCommandDescription commandDescription()
```

Discussion
Once a command is created, its command description is immutable.

See Also
`isWellFormed` (page 523)

directParameter

Returns the object that corresponds to the `keyDirectObject` parameter of the Apple event from which the receiver derives, or `null` if the event doesn't contain a direct parameter.

```
public Object directParameter()
```

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 523).

See Also
`setDirectParameter` (page 525)

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.

```
public 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 `null` if the command is not well formed. Also returns `null` if an object specifier does not evaluate to an object or if there is no type defined for the argument in the command description.

See Also
`isWellFormed` (page 523)
`arguments` (page 521)
`setArguments` (page 525)

evaluatedReceivers

Returns the object or objects to which the command is to be sent (called both the “receivers” or “targets” of script commands).

```
public Object 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 `null`.

See Also
`receiversSpecifier` (page 523)
`setReceiversSpecifier` (page 525)

executeCommand

Executes the command if it is valid and returns the result, if any.

```
public Object executeCommand()
```

Discussion
Before this method executes the command, 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 523).

See Also
`evaluatedArguments` (page 522)
`evaluatedReceivers` (page 522)

isWellFormed

Returns a Boolean value that indicates whether the receiver is well formed according to its command description.

```
public boolean isWellFormed()
```

Discussion
The method ensures that there is a description of the command and that the number of arguments and the types of nonspecifier arguments conform to the command description.

See Also
`commandDescription` (page 521)

performDefaultImplementation

Overridden by subclasses to provide a default implementation for the command represented by the receiver.

```
public Object performDefaultImplementation()
```

Discussion
Do not invoke this method directly. `executeCommand` (page 522) invokes this method when the command being executed is not supported by the class of the objects receiving the command. The default implementation returns `null`.

You need to create a subclass of NSScriptCommand only if you need to provide a default implementation of a command.

receiversSpecifier

Returns the object specifier that, when evaluated, yields the receiver or receivers of the command.

```
public 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.

See Also
`evaluatedReceivers` (page 522)
`setReceiversSpecifier` (page 525)

resumeExecutionWithResult

If a successful, unmatched, invocation of `suspendExecution` (page 526) has been made, resume the execution of the command.

```
public void resumeExecutionWithResult(Object result)
```

Discussion
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 526) 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 523) 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 526) 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.

scriptErrorNumber

Returns the script error number, if any, associated with execution of the command.

```
public 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.

See Also
`setScriptErrorNumber` (page 526)

scriptErrorString

Returns the script error string, if any, associated with execution of the command.

```
public String scriptErrorString()
```

Discussion
When you subclass NSScriptCommand or one of its subclasses, you shouldn't need to override this method.

See Also
`setScriptErrorString` (page 526)

setArguments

Sets the arguments of the command to *args*.

```
public 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.

See Also
`arguments` (page 521)
`evaluatedArguments` (page 522)

setDirectParameter

Sets the object to *directParameter* that corresponds to the `keyDirectObject` parameter of the Apple event from which the receiver derives. You don't normally override this method.

```
public void setDirectParameter(Object directParameter)
```

See Also
`directParameter` (page 521)

setReceiversSpecifier

Sets the object specifier to *receiversSpec* that, when evaluated, indicates the receiver or receivers of the command.

```
public 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`.

See Also
`evaluatedReceivers` (page 522)
`receiversSpecifier` (page 523)

setScriptErrorNumber

Sets a script error number to *errorNumber* that is associated with the execution of the command.

```
public void setScriptErrorNumber(int errorNumber)
```

Discussion
If you override `performDefaultImplementation` (page 523) 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.

See Also
`scriptErrorNumber` (page 524)

setScriptErrorString

Sets a script error string to *errorString* that is associated with execution of the command.

```
public void setScriptErrorString(String errorString)
```

Discussion
If you override `performDefaultImplementation` (page 523) 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.

See Also
`scriptErrorString` (page 524)

suspendExecution

Suspends the execution of the receiver, 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()`).

```
public void suspendExecution()
```

Discussion
Otherwise, does nothing. A matching invocation of `resumeExecutionWithResult` (page 524) 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 524). 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.

Constants

NSScriptCommand uses the following error codes for general command execution problems:

Constant	Description
`NoScriptError`	No error.
`ReceiverEvaluationScriptError`	The object or objects specified by the direct parameter to a command could not be found.
`KeySpecifierEvaluationScriptError`	The object or objects specified by a key (for commands that support key specifiers) could not be found.
`ArgumentEvaluationScriptError`	The object specified by an argument could not be found.
`ReceiversCantHandle-CommandScriptError`	The receivers don't support the command sent to them.
`RequiredArguments-MissingScriptError`	An argument (or more than one argument) is missing.
`ArgumentsWrongScriptError`	An argument (or more than one argument) is of the wrong type or is otherwise invalid.
`UnknownKeyScriptError`	An unidentified error occurred; indicates an error in the scripting support of your application.
`InternalScriptError`	An unidentified internal error occurred; indicates an error in the scripting support of your application.
`OperationNotSupportedForKey-ScriptError`	The implementation of a scripting command signaled an error.
`CannotCreateScriptCommandError`	Could not create the script command; an invalid or unrecognized Apple event was received.

NSScriptCommandDescription

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Cocoa Scripting Guide

Overview

An instance of NSScriptCommandDescription describes a script command that a Cocoa application, framework, or bundle supports.

For example, commands in AppleScript's Core suite include `clone`, `count`, `create`, `delete`, `exists`, `get`, `move`, `set`, `open`, `close`, and `save`. An NSScriptCommandDescription describes the name, class, argument types, and return type of a supported command.

Scripting information is provided in one of two standard formats: the script suite format or the sdef (scripting definition) format. For more information on these formats, see Sdef Scriptability Guide for Cocoa.

Cocoa's scripting support automatically creates an NSScriptSuiteRegistry (page 549) when an application needs it, and this object loads the scripting information for the application. It creates an NSScriptCommandDescription for each command description, registers the Apple events that correspond to each command, and caches them in memory.

The API of NSScriptCommandDescription is primarily used by Cocoa's built-in scripting support to encapsulate information about script command definitions. Although you can subclass NSScriptCommandDescription, it is unlikely that you would need to.

Tasks

Constructors

`NSScriptCommandDescription` (page 530)
Returns an NSScriptCommandDescription with no data.

Getting Basic Information

`appleEventClassCode` (page 531)
Returns the four-character code for the Apple event class associated with the receiver.

`appleEventCode` (page 531)
Returns the four-character Apple event code value associated with the receiver.

`commandClassName` (page 532)
Returns the name of the class of the command (for example, “NSGetCommand”).

`commandName` (page 532)
Returns the name of the command.

`suiteName` (page 533)
Returns the name of the suite that defines the command described by the receiver.

Creating Commands

`createCommandInstance` (page 532)
Creates and returns an NSScriptCommand (page 517) object representing the command defined by the receiver.

Getting Argument Information

`appleEventCodeForArgumentWithName` (page 531)
Returns the four-character Apple event code for the command argument *`argument`*.

`argumentNames` (page 532)
Returns the names (or keys) for all arguments of the receiver’s command.

`isOptionalArgumentWithName` (page 533)
Returns a Boolean value that indicates whether the command argument identified by the key *`argument`* is an optional argument.

`typeForArgumentWithName` (page 533)
Returns the type of the command argument identified by the key *`name`*.

Getting Return-type Information

`appleEventCodeForReturnType` (page 532)
Returns the four-character Apple event code value that identifies the command’s return type.

`returnType` (page 533)
Returns the return type of the command (for example, “NSNumber” or “NSDictionary”).

Constructors

NSScriptCommandDescription

Returns an NSScriptCommandDescription with no data.

```
public NSScriptCommandDescription()
```

Discussion
Do not use this constructor.

Returns an NSScriptCommandDescription object initialized with suite *suiteName* and command *commandName* and encapsulating the information in *commandDescriptions*.

```
public NSScriptCommandDescription(String suiteName, String commandName, NSDictionary
    commandDescriptions)
```

Discussion
Registers `this` with the registry (that is, the application's single instance of NSScriptSuiteRegistry) by Apple event code and also registers all arguments with the registry. Returns `null` if the event code value or class name for the command description is missing; also returns `null` if the return type or argument values are of the wrong type.

Instance Methods

appleEventClassCode

Returns the four-character code for the Apple event class associated with the receiver.

```
public int appleEventClassCode()
```

Discussion
For example, commands in AppleScript's Core suite, such as `Clone`, `Count`, and `Create`, have a four-character code of `'core'`. This code and the event ID code returned by `appleEventCode` (page 531) together specify the necessary information for identifying and dispatching an Apple event.

appleEventCode

Returns the four-character Apple event code value associated with the receiver.

```
public int appleEventCode()
```

Discussion
This value is the event ID and, together with the event class code returned by `appleEventClassCode` (page 531), specifies the necessary information for identifying and dispatching an Apple event.

See Also
`appleEventCodeForArgumentWithName` (page 531)
`appleEventCodeForReturnType` (page 532)

appleEventCodeForArgumentWithName

Returns the four-character Apple event code for the command argument *argument*.

```
public int appleEventCodeForArgumentWithName(String argument)
```

See Also
`argumentNames` (page 532)

appleEventCodeForReturnType

Returns the four-character Apple event code value that identifies the command's return type.

```
public int appleEventCodeForReturnType()
```

See Also
`appleEventCodeForArgumentWithName` (page 531)
`returnType` (page 533)

argumentNames

Returns the names (or keys) for all arguments of the receiver's command.

```
public NSArray argumentNames()
```

Discussion
If there are no arguments for the command, returns an empty array.

commandClassName

Returns the name of the class of the command (for example, "NSGetCommand").

```
public String commandClassName()
```

Discussion
This class is always a subclass of NSScriptCommand (page 517) or NSScriptCommand itself.

See Also
`commandName` (page 532)

commandName

Returns the name of the command.

```
public String commandName()
```

Discussion
This is the command name as it appears in the script suite and may not be the same as it appears to the scripter. For example, Cocoa's core suite defines the Create command to implement what a scripter calls `make`.

See Also
`commandClassName` (page 532)

createCommandInstance

Creates and returns an NSScriptCommand (page 517) object representing the command defined by the receiver.

```
public NSScriptCommand createCommandInstance()
```

Discussion
This object is instantiated from the appropriate subclass of NSScriptCommand or from NSScriptCommand itself.

isOptionalArgumentWithName

Returns a Boolean value that indicates whether the command argument identified by the key *argument* is an optional argument.

```
public boolean isOptionalArgumentWithName(String argument)
```

Discussion
Returns `false` if there is no argument by that name.

See Also
`argumentNames` (page 532)

returnType

Returns the return type of the command (for example, “NSNumber” or “NSDictionary”).

```
public String returnType()
```

See Also
`appleEventCodeForReturnType` (page 532)

suiteName

Returns the name of the suite that defines the command described by the receiver.

```
public String suiteName()
```

See Also
`appleEventCode` (page 531)

typeForArgumentWithName

Returns the type of the command argument identified by the key *name*.

```
public String typeForArgumentWithName(String name)
```

Discussion
Returns `null` if there is no such argument.

NSScriptExecutionContext

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Cocoa Scripting Guide

Overview

An NSScriptExecutionContext 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.

In most cases, the top-level container for a complete series of nested object specifiers is automatically set to the application object (`NSApplication.sharedApplication()`), and you can get this object with the `topLevelObject` (page 538) method. But you can also set this top-level container to something else (using `setTopLevelObject` (page 538)) if the situation warrants it. There are also occasions for two special kinds of top-level container:

- An object that is being tested inside a "whose" qualifier (an instance of NSSpecifierTest). The NSWhoseSpecifier object containing this qualifier invokes `setObjectBeingTested` (page 537) for each object in the collection involved in the test. The NSSpecifierTest then invokes `objectBeingTested` (page 537) to get this object.
- An object specifier that is being tested inside a range specifier (NSRangeSpecifier). An NSRangeSpecifier contains a "start" and an "end" specifier to mark off the range; these specifiers are evaluated against the top-level range container.

In object-specifier evaluation, NSScriptObjectSpecifier's `containerIsObjectBeingTested` (page 544) and `containerIsRangeContainerObject` (page 544) are invoked to determine whether `objectBeingTested` (page 537), `rangeContainerObject` (page 537), or `topLevelObject` (page 538) should be invoked to obtain the top-level container for the current evaluation.

It is unlikely that you will need to subclass NSScriptExecutionContext.

Tasks

Constructors

`NSScriptExecutionContext` (page 536)
Returns an NSScriptCommandExecutionContext with no data.

Getting the Current Context

`sharedScriptExecutionContext` (page 537)
Returns the shared NSScriptExecutionContext instance, creating it first if it doesn't exist.

Getting and Setting the Container Object

`topLevelObject` (page 538)
Returns the top-level object for an object-specifier evaluation.

`setTopLevelObject` (page 538)
Sets the top-level object for an object-specifier evaluation.

`objectBeingTested` (page 537)
Returns the top-level container object currently being tested in a "whose" qualifier.

`setObjectBeingTested` (page 537)
Sets the top-level container object currently being tested in a "whose" qualifier to *`object`*.

`rangeContainerObject` (page 537)
Returns the top-level container object for an object specifier (encapsulated in an NSRangeSpecifier) that represents the first or last element in a range of elements.

`setRangeContainerObject` (page 537)
Sets the top-level container object for a range-specifier evaluation to *`container`*.

Constructors

NSScriptExecutionContext

Returns an NSScriptCommandExecutionContext with no data.

```
public NSScriptExecutionContext()
```

Discussion
Do not use this constructor.

Static Methods

sharedScriptExecutionContext

Returns the shared NSScriptExecutionContext instance, creating it first if it doesn't exist.

```
public static NSScriptExecutionContext sharedScriptExecutionContext()
```

Instance Methods

objectBeingTested

Returns the top-level container object currently being tested in a "whose" qualifier.

```
public Object objectBeingTested()
```

Discussion
Returns `null` if such an object does not exist.

See Also
`setObjectBeingTested` (page 537)
`containerIsObjectBeingTested` (page 544) (NSScriptObjectSpecifier)

rangeContainerObject

Returns the top-level container object for an object specifier (encapsulated in an NSRangeSpecifier) that represents the first or last element in a range of elements.

```
public Object rangeContainerObject()
```

See Also
`setObjectBeingTested` (page 537)
`containerIsRangeContainerObject` (page 544) (NSScriptObjectSpecifier)

setObjectBeingTested

Sets the top-level container object currently being tested in a "whose" qualifier to *`object`*.

```
public void setObjectBeingTested(Object object)
```

See Also
`objectBeingTested` (page 537)

setRangeContainerObject

Sets the top-level container object for a range-specifier evaluation to *`container`*.

```
public void setRangeContainerObject(Object container)
```

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*.

See Also
rangeContainerObject (page 537)

setTopLevelObject

Sets the top-level object for an object-specifier evaluation.

```
public void setTopLevelObject(Object anObject)
```

See Also
topLevelObject (page 538)

topLevelObject

Returns the top-level object for an object-specifier evaluation.

```
public Object topLevelObject()
```

Discussion
For applications, this object is automatically set to the application object, but can be set to some other container object.

See Also
setTopLevelObject (page 538)

NSScriptObjectSpecifier

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Cocoa Scripting Guide

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 path through the object hierarchy of an application (a document-based application in this case), though each scripting object need not correspond to a separate application object. At the end of this path is the object or objects that are the target (or "receiver") of the `get` command. When the command is executed, the requested values are extracted from the receiver—in this case, an array of blue words (if any exist in paragraph 10)—and are returned. Reference forms (and object specifiers) can indicate command arguments as well as command receivers, for example:

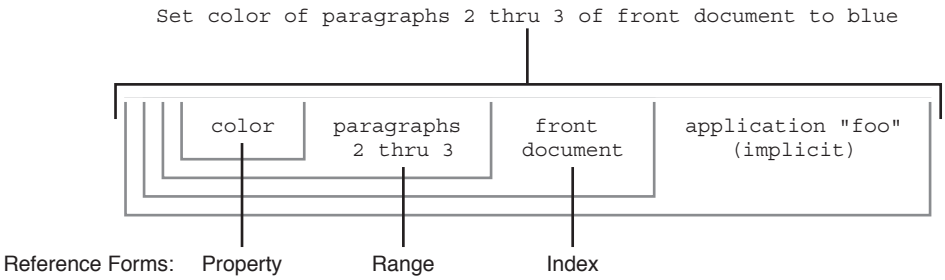
```
set first word of paragraphs 2 through last of front document to  font of first
 word of front  document
```

In this case, the reference forms up to `front document` evaluate to the receiver of the command and the reference forms starting with `font of first word` evaluate to the command's argument.

A script command (NSScriptCommand) arrives in a scriptable application with the receiver or receivers set as object specifiers and any arguments possibly set as object specifiers (arguments can be actual objects as well). To represent a series of reference forms, each object specifier is nested inside its "container" object specifier; the innermost object specifier indicates the final object to be evaluated, while the topmost object is usually the application itself. An object specifier keeps references not only to its container (or "parent") specifier but also to its "child" specifier. Although the chain of references is bidirectional, the child specifier "owns" its parent container.

For an example of nested references, take the statement `set color of paragraphs 2 thru 3 of front document to blue`; as container specifiers mapped to reference forms, this statement could be depicted as the following:

Figure 88-1 Reference forms and nested object specifiers



These reference forms are represented by instances of the appropriate NSScriptObjectSpecifier subclasses: NSPropertySpecifier (page 465), NSRangeSpecifier (page 477), and NSIndexSpecifier (page 223). Most subclasses add an appropriate instance variable (such as an index number or an NSRange value) and then implement accessor methods and an initializer for that instance variable. Each of these object specifiers except for NSPropertySpecifier deals with identifying objects in collections (NSArray).

The keys to an attribute or relationship are often not the same words expressed by the corresponding reference forms. For example, the key for an array of document objects is `orderedDocuments`, but the actual scripting term used is `document`. The mapping between key name and script name is done through the information contained in the scripting information provided for scriptable classes used in an application. When the Apple event translator converts an Apple event into an NSScriptCommand, it consults the appropriate terminology for the mapping between the scripting name and the key of the related class, attribute, or relationship. Armed with this key, it can then locate the language-independent information (specifically, class and command descriptions) needed to compose the NSScriptCommand, including its arguments and receivers.

In the normal course of script-command execution, an application invokes `evaluatedReceivers` (page 522) on an NSScriptCommand to get the receiver or receivers of the command and invokes `evaluatedArguments` (page 522) to get any arguments of the command. These methods in turn invoke `objectsByEvaluatingSpecifier` (page 545) on the object specifiers representing command arguments or receivers. The object specifier receiving the message is the innermost specifier as nested in its containers (`color` in the above example). The `objectsByEvaluatingSpecifier` (page 545) method goes up the chain of nested containers by asking each specifier for its container until it comes to the top-level object specifier, which has no container. The top-level object is usually the application object (`NSApplication.sharedApplication()`), but it can be an object specifier involved in a whose clause (NSWhoseSpecifier) or the container for a range evaluation. The method then invokes `objectsByEvaluatingWithContainers` (page 546) on this top-level specifier, which then proceeds down the chain of nested specifiers, evaluating each through key-value coding and using the evaluated object as the basis for the next evaluation. Evaluating the innermost specifier yields the real command receiver or receivers or any object used as a command argument.

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` (page 545) 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 constructor, and initializes these variables.

Other Classes Used in Object-Specifier Evaluation

In addition to the concrete subclasses of NSScriptObjectSpecifier mentioned in the previous section, Cocoa frameworks include a handful of other classes that assist the object-specifier classes in evaluation. Instances of these classes help to indicate relative position (NSPositionalSpecifier (page 451)) and represent Boolean and logical expressions in which object specifiers are involved.

Boolean Expressions and Logical Operations

A script statement can contain filter reference forms, which identify objects in containers based on the conditions specified in Boolean expressions. These expressions can be linked together by logical operators (AND, OR, NOT) and return the appropriate combined true or false value. Filter reference forms begin with the words `whose` or `where`, as in `get words where color is blue or color is red of front document`. These reference forms can contain phrases such as `is`, `is equal to` or `is greater than` as well as their symbolic equivalents (such as = and >).

Instances of the NSWhoseSpecifier class represent filter reference forms in Cocoa (see the class description of NSWhoseSpecifier (page 675) for more information). These instances hold a "test" instance variable, which is an NSScriptWhoseTest object. For more information, see the class description for NSScriptWhoseTest (page 557) and the descriptions in NSComparisonMethods (page 685).

You shouldn't need to subclass NSScriptObjectSpecifier, and you should rarely need to subclass any of its subclasses.

Tasks

Constructors

`NSScriptObjectSpecifier` (page 543)
Returns an NSScriptObjectSpecifier with no data.

Evaluating an Object Specifier

`indicesOfObjectsByEvaluatingWithContainer` (page 545)

`objectsByEvaluatingSpecifier` (page 545)

`objectsByEvaluatingWithContainers` (page 546)

Getting, Testing, and Setting Containers

`containerClassDescription` (page 543)

`containerIsObjectBeingTested` (page 544)
: If the receiver's container specifier is `null`, returns a Boolean value that indicates whether the receiver's container is the object involved in a specifier test.

`containerIsRangeContainerObject` (page 544)
: If the receiver's container specifier is `null`, returns a Boolean value that indicates whether the container for the receiver contains the range of elements represented by an NSRangeSpecifier.

`containerSpecifier` (page 544)

`setContainerClassDescription` (page 546)

`setContainerIsObjectBeingTested` (page 547)

`setContainerSpecifier` (page 547)

`setContainerIsRangeContainerObject` (page 547)

Getting and Setting Child References

`childSpecifier` (page 543)

`setChildSpecifier` (page 546)

Getting and Setting Object Keys

`key` (page 545)

`keyClassDescription` (page 545)

`setKey` (page 548)

Getting Evaluation Errors

`evaluationErrorSpecifier` (page 545)

`evaluationErrorNumber` (page 544)

`setEvaluationErrorNumber` (page 547)

Constructors

NSScriptObjectSpecifier

Returns an NSScriptObjectSpecifier with no data.

```
public NSScriptObjectSpecifier()
```

Discussion
Do not use this constructor.

Returns an NSScriptObjectSpecifier 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.

```
public NSScriptObjectSpecifier(NSScriptClassDescription classDescription,
    NSScriptObjectSpecifier specifier, String key)
```

Discussion
The receiver's child specifier reference is set to `null`.

Returns a newly created NSScriptObjectSpecifier with container specifier *specifier* and key *key*.

```
public NSScriptObjectSpecifier(NSScriptObjectSpecifier specifier, String key)
```

Discussion
The class description of the container is set automatically.

Instance Methods

childSpecifier

```
public NSScriptObjectSpecifier childSpecifier()
```

Discussion
Returns the receiver's child reference, that is, the object specifier evaluating to the object or objects that the receiver contains.

See Also
`setChildSpecifier` (page 546)

containerClassDescription

```
public NSScriptClassDescription containerClassDescription()
```

Discussion
Returns the class description of the object indicated by the receiver's container specifier.

See Also
`setContainerClassDescription` (page 546)

containerIsObjectBeingTested

If the receiver's container specifier is `null`, returns a Boolean value that indicates whether the receiver's container is the object involved in a specifier test.

```
public boolean containerIsObjectBeingTested()
```

Discussion
An example of a specifier test is `whose color is blue`). If the returned value is `true`, then the top-level object is the object being tested (that is, the specifier has no container specifier).

See Also
`objectBeingTested` (page 537) (NSScriptExecutionContext)

containerIsRangeContainerObject

If the receiver's container specifier is `null`, returns a Boolean value that indicates whether the container for the receiver contains the range of elements represented by an NSRangeSpecifier.

```
public boolean containerIsRangeContainerObject()
```

See Also
`setContainerIsRangeContainerObject` (page 547)

containerSpecifier

```
public NSScriptObjectSpecifier containerSpecifier()
```

Discussion
Returns the receiver's container specifier, which is the object specifier that must be evaluated to provide a context for the evaluation of the receiver.

See Also
`childSpecifier` (page 543)
`containerClassDescription` (page 543)
`setContainerSpecifier` (page 547)

evaluationErrorNumber

```
public int evaluationErrorNumber()
```

Discussion
Returns the constant identifying the type of error that caused evaluation to fail. This error code could be associated with the receiver or any container specifier "above" the receiver. Possible return values are defined in "Constants" (page 548).

See Also
`evaluationErrorSpecifier` (page 545)

evaluationErrorSpecifier

```
public NSScriptObjectSpecifier evaluationErrorSpecifier()
```

Discussion
Returns the object specifier in which an evaluation error occurred. The object specifier failing to evaluate could be the receiver or any container specifier "above" the receiver.

See Also
`evaluationErrorNumber` (page 544)

indicesOfObjectsByEvaluatingWithContainer

```
public int[] indicesOfObjectsByEvaluatingWithContainer(Object specifier)
```

Discussion
Returns an array of indices identifying objects in the key of the container *specifier* 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 –1 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 546). The default implementation returns –1.

key

```
public String key()
```

Discussion
Returns the key of the receiver.

See Also
`keyClassDescription` (page 545)
`setKey` (page 548)

keyClassDescription

```
public NSScriptClassDescription keyClassDescription()
```

Discussion
Returns the class description of the objects specified by the receiver.

See Also
`key` (page 545)
`setKey` (page 548)

objectsByEvaluatingSpecifier

```
public Object objectsByEvaluatingSpecifier()
```

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 546)) 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 `null` if a container specifier could not be evaluated or if no top-level container specifier could be found. Thus `null` can be a valid value or can indicate an error; you can use `evaluationErrorNumber` (page 544) to determine if and which error occurred and `evaluationErrorSpecifier` (page 545) to find the container specifier responsible for the error. In the normal course of command processing, this method is invoked by NSScriptCommand's `evaluatedArguments` (page 522) and `evaluatedReceivers` (page 522), which take as message receiver the innermost object specifier.

See Also
`indicesOfObjectsByEvaluatingWithContainer` (page 545)

objectsByEvaluatingWithContainers

```
public Object objectsByEvaluatingWithContainers(Object containers)
```

Discussion
Returns the actual object or objects specified by the receiver as evaluated in the context of its container object or objects (*containers*). Invokes `indicesOfObjectsByEvaluatingWithContainer` (page 545) on `this` 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 `null` if there are no matching values in containers. If there are multiple matching values, they are returned in an NSArray; if matching values are `null`, NSNulls 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.

See Also
`objectsByEvaluatingSpecifier` (page 545)

setChildSpecifier

```
public void setChildSpecifier(NSScriptObjectSpecifier child)
```

Discussion
Sets the receiver's child reference to *child*. Do not invoke this method directly; it is automatically invoked by `setContainerSpecifier` (page 547).

See Also
`childSpecifier` (page 543)

setContainerClassDescription

```
public void setContainerClassDescription(NSScriptClassDescription classDescription)
```

Discussion
Sets the class description of the receiver's container specifier to *classDescription*.

See Also
containerClassDescription (page 543)

setContainerIsObjectBeingTested

```
public void setContainerIsObjectBeingTested(boolean flag)
```

Discussion
If the receiver's container specifier is `null` and *flag* is `true`, 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 `null` and *flag* is `false`, sets the receiver's container to be the top-level object.

If this method is invoked with an argument of `true` setContainerIsRangeContainerObject (page 547) should not also be invoked with an argument of `true`.

See Also
containerIsObjectBeingTested (page 544)

setContainerIsRangeContainerObject

```
public void setContainerIsRangeContainerObject(boolean flag)
```

Discussion
If the receiver's container specifier is `null` and *flag* is `true`, sets the receiver's container to be the container for a range specifier. If the receiver's container specifier is `null` and *flag* is `false`, sets the receiver's container to be the top-level object.If this method is invoked with an argument of `true`, setContainerIsObjectBeingTested (page 547) should not also be invoked with an argument of `true`.

See Also
containerIsRangeContainerObject (page 544)

setContainerSpecifier

```
public void setContainerSpecifier(NSScriptObjectSpecifier objSpecifier)
```

Discussion
Sets the container specifier of the receiver to *objSpecifier*.

See Also
containerSpecifier (page 544)

setEvaluationErrorNumber

```
public void setEvaluationErrorNumber(int error)
```

Discussion
Sets the value of the evaluation error to *error*.

See Also
evaluationErrorNumber (page 544)

setKey

```
public void setKey(String key)
```

Discussion
Sets the key of the receiver to *key*.

See Also
`key` (page 545)
`keyClassDescription` (page 545)

Constants

NSScriptObjectSpecifier provides the following constants for error codes for specific problems evaluating specifiers:

Constant	Description
`NoSpecifierError`	No error encountered.
`NoTopLevelContainersSpecifierError`	Someone called `evaluate` with `null`.
`ContainerSpecifierError`	Error evaluating container specifier.
`UnknownKeySpecifierError`	Receivers do not understand the key.
`InvalidIndexSpecifierError`	Index out of bounds.
`InternalSpecifierError`	Other internal error.
`OperationNotSupportedForKey-SpecifierError`	Attempt made to perform an unsupported operation on some key.

NSScriptSuiteRegistry

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Cocoa Scripting Guide

Overview

NSScriptSuiteRegistry functions as the top-level repository of scripting information for an application at runtime.

NSScriptSuiteRegistry is a key player in Cocoa's built-in scripting support. To take advantage of that support, each application supplies scripting information (or metadata) in one of two standard formats: the script suite format or the sdef (scripting definition) format. For information on these formats, and on how to create script suite and sdef files, see Sdef Scriptability Guide for Cocoa.

An application's scripting information specifies what a scripter can do with the application, including which objects, properties, and commands are available. When a user executes a script, the appropriate scripting component (most commonly the AppleScript component) converts command lines from the script (such as `close the first document`) into Apple events that are sent to the targeted application. Using the information stored in NSScriptSuiteRegistry, Cocoa automatically converts incoming Apple events into script commands (based on NSScriptCommand (page 517) or a subclass) that manipulate objects in the application.

One instance of NSScriptSuiteRegistry is shared among the objects of an application. Scripting information is collected when an application first needs to respond to an Apple event other than the core events (`open documents`, `print documents`, `open application`, `print documents`, `reopen`, and `quit`). An NSScriptSuiteRegistry is automatically created to load the scripting information:

- For applications that supply scripting information in script suites, the NSScriptSuiteRegistry loads the script suite files (matching pairs of files with the extensions `.scriptSuite` and `.scriptTerminology`) for the application, for all imported frameworks, and for all loaded bundles.
- For applications in Mac OS version 10.4 that supply scripting information in an sdef file, the NSScriptSuiteRegistry loads information from the specified file.

The scripting information provided by an application has two main parts: one for "class descriptions" and the other for "command descriptions." A class description defines the attributes and relationships of a scriptable object; it also lists the script commands supported by the object. A command description defines each command in this suite, including its class (NSScriptCommand or subclass), return type, and argument names and types. Scripting information may contain additional declarations, such as enumerations.

When an NSScriptSuiteRegistry loads scripting information, it creates NSScriptClassDescriptions and NSScriptCommandDescriptions from the contents. It also caches the corresponding four-character Apple event codes. It registers class descriptions with the central class registry maintained by NSClassDescription,

and it registers to handle incoming Apple events that represent the defined commands. An application's scripting information also maps scripting terminology—the English-like words and phrases a scripter can use in a script, such as `the first word in the first paragraph`—to the class and command descriptions used to implement scripting support in the application.

The public methods of NSScriptSuiteRegistry are used primarily by Cocoa's built-in scripting support to access scripting information from all loaded suites. You should rarely need to create a subclass of NSScriptSuiteRegistry.

Tasks

Constructors

`NSScriptSuiteRegistry` (page 551)
Returns an NSScriptSuiteRegistry with no data.

Getting and Setting the Shared Instance

`setSharedScriptSuiteRegistry` (page 552)
Sets the single, shared instance of NSScriptSuiteRegistry to *registry*.

`sharedScriptSuiteRegistry` (page 552)
Returns the single, shared instance of NSScriptSuiteRegistry, creating it first if it doesn't exist.

Getting Suite Information

`suiteForAppleEventCode` (page 555)
Returns the name of the suite definition associated with the given four-character Apple event code, *code*.

`suiteNames` (page 555)
Returns the names of the suite definitions currently loaded by the application.

Getting and Registering Class Descriptions

`classDescriptionsInSuite` (page 553)
Returns the class descriptions contained in the suite identified by *suiteName*.

`classDescriptionWithAppleEventCode` (page 553)
Returns the class description associated with the given four-character Apple event code, *code*.

`registerClassDescription` (page 555)
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 553)
Returns the command descriptions contained in the suite identified by *suiteName*.

`commandDescriptionWithAppleEventCodes` (page 554)
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 555)
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 552)
Returns an NSData object that contains data in 'aete' resource format describing the suites currently known to the application.

`appleEventCodeForSuite` (page 552)
Returns the Apple event code associated with the suite named *suiteName*, such as 'core' for the Core suite.

`bundleForSuite` (page 553)
Returns the bundle containing the suite-definition property list (extension `.scriptSuite`) identified by *suiteName*.

Loading Suites

`loadSuiteWithDictionary` (page 554)
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 554)
Loads the suite definitions in bundle *aBundle*, invoking `loadSuiteWithDictionary` (page 554) for each suite found.

Constructors

NSScriptSuiteRegistry

Returns an NSScriptSuiteRegistry with no data.

```
public NSScriptSuiteRegistry()
```

Discussion
Do not use this constructor.

Static Methods

setSharedScriptSuiteRegistry

Sets the single, shared instance of NSScriptSuiteRegistry to *registry*.

```
public static void setSharedScriptSuiteRegistry(NSScriptSuiteRegistry registry)
```

sharedScriptSuiteRegistry

Returns the single, shared instance of NSScriptSuiteRegistry, creating it first if it doesn't exist.

```
public static NSScriptSuiteRegistry sharedScriptSuiteRegistry()
```

Discussion
If it creates an instance, the method loads suite definitions in all frameworks and other bundles that the application currently imports or includes. If in reading a `.scriptSuite` property list an exception is `thrown` because of parsing errors, it handles the exception by printing a line of information to the console.

See Also
`loadSuiteWithDictionary` (page 554)

Instance Methods

aeteResource

Returns an NSData object that contains data in `'aete'` resource format describing the suites currently known to the application.

```
public NSData aeteResource(String languageName)
```

Discussion
This method is typically invoked to implement the Get AETE Apple event. 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'` unless this method is called.

See Also
`appleEventCodeForSuite` (page 552)

appleEventCodeForSuite

Returns the Apple event code associated with the suite named *suiteName*, such as `'core'` for the Core suite.

```
public int appleEventCodeForSuite(String suiteName)
```

See Also
suiteForAppleEventCode (page 555)

bundleForSuite

Returns the bundle containing the suite-definition property list (extension .scriptSuite) identified by *suiteName*.

```
public NSBundle bundleForSuite(String suiteName)
```

classDescriptionsInSuite

Returns the class descriptions contained in the suite identified by *suiteName*.

```
public NSDictionary classDescriptionsInSuite(String suiteName)
```

Discussion
Each class description (instance of NSScriptClassDescription) in the returned dictionary is identified by class name.

See Also
classDescriptionWithAppleEventCode (page 553)
registerClassDescription (page 555)

classDescriptionWithAppleEventCode

Returns the class description associated with the given four-character Apple event code, *code*.

```
public NSScriptClassDescription classDescriptionWithAppleEventCode(int 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.

See Also
classDescriptionsInSuite (page 553)
registerClassDescription (page 555)

commandDescriptionsInSuite

Returns the command descriptions contained in the suite identified by *suiteName*.

```
public NSDictionary commandDescriptionsInSuite(String suiteName)
```

Discussion
Each command description (instance of NSScriptCommandDescription) in the returned dictionary is identified by command name.

See Also
commandDescriptionWithAppleEventCodes (page 554)
registerCommandDescription (page 555)

commandDescriptionWithAppleEventCodes

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*).

```
public NSScriptCommandDescription commandDescriptionWithAppleEventCodes(int
    eventClass, int commandCode)
```

See Also
commandDescriptionsInSuite (page 553)
registerCommandDescription (page 555)

loadSuitesFromBundle

Loads the suite definitions in bundle *aBundle*, invoking loadSuiteWithDictionary (page 554) for each suite found.

```
public void loadSuitesFromBundle(NSBundle aBundle)
```

Discussion
If errors occur while method is parsing a suite-definition file, the method logs error messages to the console.

loadSuiteWithDictionary

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*.

```
public void loadSuiteWithDictionary(NSDictionary dictionary, 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 NSClassDescription's class method registerClassDescription (page 107). 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.

See Also
loadSuitesFromBundle (page 554)
registerClassDescription (page 555)
registerCommandDescription (page 555)
sharedScriptSuiteRegistry (page 552)

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.

```
public void registerClassDescription(NSScriptClassDescription classDescription)
```

See Also
loadSuiteWithDictionary (page 554)
registerCommandDescription (page 555)

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.

```
public void registerCommandDescription(NSScriptCommandDescription commandDesc)
```

See Also
loadSuiteWithDictionary (page 554)
registerClassDescription (page 555)

suiteForAppleEventCode

Returns the name of the suite definition associated with the given four-character Apple event code, *code*.

```
public String suiteForAppleEventCode(int code)
```

See Also
suiteNames (page 555)

suiteNames

Returns the names of the suite definitions currently loaded by the application.

```
public NSArray suiteNames()
```

See Also
suiteForAppleEventCode (page 555)

NSScriptWhoseTest

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Cocoa Scripting Guide

Overview

NSScriptWhoseTest is an abstract class whose sole method is `isTrue` (page 558). 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 (page 585) and NSLogicalTest (page 265). In evaluating itself, an NSWhoseSpecifier invokes the `isTrue` (page 558) method of its "test" object.

You shouldn't need to subclass NSScriptWhoseTest, and you should rarely need to subclass one of its subclasses.

Tasks

Constructors

`NSScriptWhoseTest` (page 557)
Returns an NSScriptWhoseTest with no data.

Evaluating a Test

`isTrue` (page 558)
Returns `true` if the test represented by the receiver evaluates to `true`.

Constructors

NSScriptWhoseTest

Returns an NSScriptWhoseTest with no data.

```
public NSScriptWhoseTest()
```

Discussion
Do not use this constructor.

Instance Methods

isTrue

Returns `true` if the test represented by the receiver evaluates to `true`.

```
public boolean isTrue()
```

NSSelector

Inherits from	Object
Package:	com.apple.cocoa.foundation
Companion guide	Cocoa-Java Integration Guide

Overview

An NSSelector object (also called a selector) specifies a method signature, which is a method's name and parameter list. You can later apply a selector on any object, and it performs the method that matches the selector, if there is one.

NSSelector is the Java implementation of the Objective-C type `SEL`, described in the "Selectors" section of *The Objective-C Programming Language*.

For details on using NSSelector, see "Method Selectors" and "Using NSSelector".

Tasks

Constructors

`NSSelector` (page 560)
Deprecated.

Invoking a Selector

`invoke` (page 560)

`invoke` (page 562)

Accessing Information About a Selector

`equals` (page 561)
Returns whether *`anObject`* equals the method that matches the receiver.

`implementedByClass` (page 561)
: Returns whether the class *`targetClass`* implements a method that matches the receiver.

`implementedByObject` (page 561)
: Returns whether the object *`target`* implements a method that matches the receiver.

`methodOnClass` (page 563)
: Returns the method on the class *`targetClass`* that matches the receiver. If *`targetClass`* has no method that matches the receiver, this method throws `NoSuchMethodException`.

`methodOnObject` (page 563)
: Returns the method on the object *`target`* that matches the receiver. If *`target`* has no method that matches the receiver, this method throws `NoSuchMethodException`.

`name` (page 563)
: Returns the name of the method specified by the receiver.

`parameterTypes` (page 563)
: Copies and returns the array of parameter types specified by the receiver.

`toString` (page 563)
: Returns a string containing the receiver's class ("NSSelector") and the name of the method the receiver specifies.

Constructors

NSSelector

Deprecated.

```
public NSSelector(String methodName)
```

Creates a selector for the method that's named *`methodName`* and takes parameters *`parameterTypes`*.

```
public NSSelector(String methodName, Class[] parameterTypes)
```

Discussion
To create a selector for a method that takes no arguments, use `null` for *`parameterTypes`*. For an example, see "Using NSSelector".

Static Methods

invoke

```
public static Object invoke(String methodName, Class[] parameterTypes, Object
    target, Object[] arguments)
```

Discussion
Creates and applies a selector that has any number of arguments. This method creates a selector with *`methodName`* and the parameter types in the array *`parameterTypes`*, applies that selector to *`target`* with the arguments in the array *`arguments`*, and returns the result. To apply a method that takes no arguments,

use `null` for the arrays *parameterTypes* and *arguments*. As part of its implementation, this method uses the NSSelector constructor and the instance method `invoke` (page 562). For more information, see those method descriptions.

```
public static Object invoke(String methodName, Class parameterType, Object target,
     Object argument)
```

Discussion
Creates and applies a selector that has one argument. This method creates a selector with *methodName* and *parameterType*, applies that selector to *target* with *argument*, and returns the result. As part of its implementation, this method uses the NSSelector constructor and the instance method `invoke` (page 562). For more information, see those method descriptions.

```
public static Object invoke(String methodName, Class parameterType1, Class
    parameterType2, Object target, Object argument1, Object argument2)
```

Discussion
Creates and applies a selector that has two arguments. This method creates a selector with *methodName* and the parameter types *parameterType1* and *parameterType2*, applies that selector to *target* with the arguments *argument1* and *argument2*, and returns the result. As part of its implementation, this method uses the NSSelector constructor and the instance method `invoke` (page 562). For more information, see those method descriptions.

Instance Methods

equals

Returns whether *anObject* equals the method that matches the receiver.

```
public boolean equals(Object anObject)
```

implementedByClass

Returns whether the class *targetClass* implements a method that matches the receiver.

```
public boolean implementedByClass(Class targetClass)
```

implementedByObject

Returns whether the object *target* implements a method that matches the receiver.

```
public boolean implementedByObject(Object target)
```

Discussion
As part of its implementation, this method uses `implementedByClass` (page 561).

invoke

```
public Object invoke(Object target, Object[] arguments)
```

Discussion
Invokes the method specified by the receiver on *target* with *arguments* and returns the result. If that method is `void`, it returns `null`. Note that the method may be a static or instance method.

The `invoke` method can't handle arguments or return values of primitive types (such as `boolean`, `int`, or `float`). If the method matching the selector returns a value of a primitive type, `invoke` returns the value in an object of the corresponding wrapper type (such as Boolean, Integer, or Float). To pass an argument of a primitive type to `invoke`, use an object of the corresponding wrapper class. `invoke` converts the object back to the primitive type when it invokes the method.

`invoke` throws an exception in the following cases:

- If *target* has no method that matches the selector, it throws `NoSuchMethodException`.
- If a method matches the selector but is inaccessible to *target*, it throws `IllegalAccessException`.
- If it can't convert an argument to the type specified in the selector, it throws `IllegalArgumentException`.
- If the invoked method throws an exception, it wraps that exception in a `java.lang.reflect.InvocationTargetException` and throws the new exception without completing.

As part of its implementation, this method uses `methodOnClass` (page 563).

For an example, see "Using NSSelector".

```
public Object invoke(Object target)
```

Discussion
Invokes the method specified by the selector on *target* with no arguments and returns the result. If that method is `void`, it returns `null`. Note that the method may be a static or instance method.

As part of its implementation, this method calls the `invoke` (page 562) instance method which takes an array of arguments. For more information, see that method's description.

```
public Object invoke(Object target, Object argument)
```

Discussion
Invokes the method specified by the receiver on *target* with one argument (*argument*) and returns the result. If that method is `void`, it returns `null`. Note that the method may be a static or instance method.

As part of its implementation, this method calls the `invoke` (page 562) instance method, which takes an array of arguments. For more information, see that method's description.

```
public Object invoke(Object target, Object argument1, Object argument2)
```

Discussion
Invokes the method specified by the receiver on *target* with two arguments (*argument1* and *argument2*) and returns the result. If that method is `void`, it returns `null`. Note that the method may be a static or instance method.

As part of its implementation, this method calls the `invoke` (page 562) instance method, which takes an array of arguments. For more information, see that method's description.

methodOnClass

Returns the method on the class *targetClass* that matches the receiver. If *targetClass* has no method that matches the receiver, this method throws `NoSuchMethodException`.

```
public java.lang.reflect.Method methodOnClass(Class targetClass)
```

methodOnObject

Returns the method on the object *target* that matches the receiver. If *target* has no method that matches the receiver, this method throws `NoSuchMethodException`.

```
public java.lang.reflect.Method methodOnObject(Object target)
```

name

Returns the name of the method specified by the receiver.

```
public String name()
```

parameterTypes

Copies and returns the array of parameter types specified by the receiver.

```
public Class[] parameterTypes()
```

toString

Returns a string containing the receiver's class ("NSSelector") and the name of the method the receiver specifies.

```
public String toString()
```

NSSet

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.foundation
Companion guide	Collections Programming Topics for Cocoa

Class at a Glance

An NSSet object stores an immutable set of objects.

Principal Attributes

- The objects that make up the set.

`"NSSet"` (page 567)
Creates a set.

Commonly Used Methods

`allObjects` (page 568)
Returns an array containing the set's member objects.

`count` (page 568)
Returns the number of objects in the set.

`containsObject` (page 568)
Indicates whether a given object is present in the set.

Primitive Methods

`count` (page 568)
`member` (page 569)
`objectEnumerator` (page 570)

Overview

The NSSet and NSMutableSet 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 class is provided for sets whose contents may be altered.

The mutable subclass of NSSet is NSMutableSet (page 351).

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.

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 added to a set are not copied; rather, an object is added directly to a set.

NSSet provides methods for querying the elements of the set. `allObjects` (page 568) returns an array containing the objects in a set. `anyObject` (page 568) returns some object in the set. `count` (page 568) returns the number of objects currently in the set. `member` (page 569) returns the object in the set that is equal to a specified object. Additionally, `intersectsSet` (page 569) tests for set intersection, `isEqualToSet` (page 569) tests for set equality, and `isSubsetOfSet` (page 569) tests for one set being a subset of another.

The `objectEnumerator` (page 570) method provides for traversing elements of the set one by one.

Tasks

Constructors

`NSSet` (page 567)

Counting Entries

`count` (page 568)
Returns the number of members in the receiver.

Accessing the Members

`allObjects` (page 568)
Returns an array containing the receiver's members, or an empty array if the receiver has no members.

`anyObject` (page 568)
Returns one of the objects in the receiver, or `null` if the receiver contains no objects.

`containsObject` (page 568)
Returns `true` if *`anObject`* is present in the receiver, `false` otherwise.

`member` (page 569)
If *`anObject`* is present in the receiver (as determined by `equals` (page 424)), the object in the receiver is returned.

`objectEnumerator` (page 570)
Returns an enumerator object that lets you access each object in the receiver.

Comparing Sets

`isSubsetOfSet` (page 569)
Returns `true` if every object in the receiver is also present in *`otherSet`*, `false` otherwise.

`intersectsSet` (page 569)
Returns `true` if at least one object in the receiver is also present in *`otherSet`*, `false` otherwise.

`isEqualToSet` (page 569)
Compares the receiving set to *`otherSet`*.

Joining Sets

`setByIntersectingSet` (page 570)
Returns a set with all objects that are in both the receiver and *`otherSet`*.

`setBySubtractingSet` (page 570)
Returns a set with all objects that are in the receiver but not in *`otherSet`*.

`setByUnioningSet` (page 570)
Returns a set with all objects that are in either the receiver or *`otherSet`* or both.

Constructors

NSSet

```
public NSSet()
```

Discussion
Returns an empty set.

```
public NSSet(Object anObject)
```

Discussion
Returns a set containing a single member, *`anObject`*.

```
public NSSet(Object[] objects)
```

Discussion
Returns a set containing those objects in *`objects`*.

```
public NSSet(NSSet aSet)
```

Discussion
Returns a set containing those objects contained within the set *aSet*.

Instance Methods

allObjects

Returns an array containing the receiver's members, or an empty array if the receiver has no members.

```
public NSArray allObjects()
```

Discussion
The order of the objects in the array isn't defined.

See Also
anyObject (page 568)
objectEnumerator (page 570)

anyObject

Returns one of the objects in the receiver, or `null` if the receiver contains no objects.

```
public Object anyObject()
```

Discussion
The object returned is chosen at the receiver's convenience—the selection is not guaranteed to be random.

See Also
allObjects (page 568)
objectEnumerator (page 570)

containsObject

Returns `true` if *anObject* is present in the receiver, `false` otherwise.

```
public boolean containsObject(Object anObject)
```

See Also
member (page 569)

count

Returns the number of members in the receiver.

```
public int count()
```

intersectsSet

Returns `true` if at least one object in the receiver is also present in *`otherSet`*, `false` otherwise.

```
public boolean intersectsSet(NSSet otherSet)
```

See Also
`isEqualToSet` (page 569)
`isSubsetOfSet` (page 569)

isEqualToSet

Compares the receiving set to *`otherSet`*.

```
public boolean isEqualToSet(NSSet otherSet)
```

Discussion
If the contents of *`otherSet`* are equal to the contents of the receiver, this method returns `true`. If not, it returns `false`.

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.

See Also
`intersectsSet` (page 569)
`equals` (page 424) (NSObject)
`isSubsetOfSet` (page 569)

isSubsetOfSet

Returns `true` if every object in the receiver is also present in *`otherSet`*, `false` otherwise.

```
public boolean isSubsetOfSet(NSSet otherSet)
```

See Also
`intersectsSet` (page 569)
`isEqualToSet` (page 569)

member

If *`anObject`* is present in the receiver (as determined by `equals` (page 424)), the object in the receiver is returned.

```
public Object member(Object anObject)
```

Discussion
Otherwise, returns `null`. If you override `equals`, you must also override the `hash` method for the `member` method to work on a set of objects of your class.

objectEnumerator

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

```
public java.util.Enumeration objectEnumerator()
```

Discussion
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 568) method to create a "snapshot" of the set's members. Enumerate the snapshot, but make your modifications to the original set.

See Also
`nextElement` (page 167) (NSEnumerator)

setByIntersectingSet

Returns a set with all objects that are in both the receiver and *otherSet*.

```
public NSSet setByIntersectingSet(NSSet otherSet)
```

See Also
`intersectsSet` (page 569)
`isSubsetOfSet` (page 569)
`isEqualToSet` (page 569)
`setBySubtractingSet` (page 570)
`setByUnioningSet` (page 570)

setBySubtractingSet

Returns a set with all objects that are in the receiver but not in *otherSet*.

```
public NSSet setBySubtractingSet(NSSet otherSet)
```

See Also
`intersectsSet` (page 569)
`isSubsetOfSet` (page 569)
`isEqualToSet` (page 569)
`setByIntersectingSet` (page 570)
`setByUnioningSet` (page 570)

setByUnioningSet

Returns a set with all objects that are in either the receiver or *otherSet* or both.

```
public NSSet setByUnioningSet(NSSet otherSet)
```

Discussion
If an object is in both, the set contains only one copy.

See Also
`intersectsSet` (page 569)

`isSubsetOfSet` (page 569)
`isEqualToSet` (page 569)
`setByIntersectingSet` (page 570)
`setBySubtractingSet` (page 570)

NSSetCommand

Inherits from	NSScriptCommand : NSObject
Package:	com.apple.cocoa.foundation
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:`; for the former, it invokes `setValue:forKey:` (or, if the receiver overrides `takeValue:forKey:`, it invokes that method, to support backward binary compatibility.)

Tasks

Constructors

`NSSetCommand` (page 574)
Returns an NSSetCommand with no data.

Working with Specifiers

`keySpecifier` (page 574)
Returns a specifier that identifies the attribute or relationship that is to be set for the receiver of the Set command.

`setReceiversSpecifier` (page 574)
Sets the receiver's object specifier; when evaluated, the specifier indicates the receiver or receivers of the Set command.

Constructors

NSSetCommand

Returns an NSSetCommand with no data.

```
public NSSetCommand()
```

Discussion
Do not use this constructor.

Initializes an instance of NSSetCommand with the command description supplied by *commandDesc*.

```
public NSSetCommand(NSScriptCommandDescription commandDesc)
```

Discussion
Note that such an instance has no receiver specifier, arguments, or direct parameter and is not a fully functional command.

Instance Methods

keySpecifier

Returns a specifier that identifies the attribute or relationship that is to be set for the receiver of the Set command.

```
public NSScriptObjectSpecifier keySpecifier()
```

setReceiversSpecifier

Sets the receiver's object specifier; when evaluated, the specifier indicates the receiver or receivers of the Set command.

```
public void setReceiversSpecifier(NSScriptObjectSpecifier receiversRef)
```

Discussion
When the command is executed, it sets attributes or relationships in the specified receivers.

This method overrides `setReceiversSpecifier` (page 525) 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`.

NSSize

Inherits from	Object
Implements	Cloneable
Package:	com.apple.cocoa.foundation
Companion guide	Geometry and Range Utilities

Overview

An NSSize object represents a dimension, a specification of width and height. An NSSize, together with an NSPoint, can be used to represent a rectangle (NSRect). The methods of NSSize allow you to access the height and width of an object, to compare and test NSSizes, to convert NSSizes to AWT Dimension objects, and to convert NSSizes to and from string objects.

Tasks

Constructors

`NSSize` (page 576)

Accessing Dimensions

`height` (page 577)
Returns the height dimension of the receiver.

`width` (page 578)
Returns the width of the receiver.

Testing NSSizes

`equals` (page 577)
Returns whether *`otherObject`* is an NSSize and is equal in width and height to the receiver.

`hashCode` (page 577)
Provides an appropriate hash code useful for storing the receiver in a hash-based data structure.

`isEmpty` (page 578)
Returns whether either dimension (width or height) of the receiver is 0.

`isEqualToSize` (page 578)
Returns whether the NSSize *aSize* is equal in width and height to the receiver.

Transforming NSSizes

`toAWTDimension` (page 578)
Returns the receiver as an AWT Dimension object.

`toString` (page 578)
Returns the receiver as converted to a string object.

`fromString` (page 577)

Copying

`clone` (page 577)
Creates and returns a copy of the receiver.

Constructors

NSSize

```
public NSSize()
```

Discussion
Initializes the instance to a dimensionless NSSize.

```
public NSSize(float w, float h)
```

Discussion
Initializes the NSSize with the width dimension *w* and the height dimension *y*.

Normally, the values of *w* and *h* are non-negative. The constructors that create an NSSize do not prevent you from setting a negative value for these attributes. If the value of *w* or *h* is negative, however, the behavior of some methods may be undefined.

```
public NSSize(NSSize aSize)
```

Discussion
Initializes the new NSSize with the width and height values of an existing NSSize, *aSize*; this constructor is used in cloning the receiver.

```
public NSSize(java.awt.Dimension javaDimension)
```

Discussion
Initializes the NSSize with the values extracted from an AWT Dimension object, *javaDimension*.

Static Methods

fromString

```
public static NSSize fromString(String sizeAsString)
```

Discussion
Creates an NSSize from the string *sizeAsString*, which must be of the form "{*w*, *h*}" where *w* is a float representation of the width and *h* is a float representation of the height. Throws an `IllegalArgumentException` if the string is improperly formatted.

See Also
`toString` (page 578)

Instance Methods

clone

Creates and returns a copy of the receiver.

```
public Object clone()
```

equals

Returns whether *otherObject* is an NSSize and is equal in width and height to the receiver.

```
public boolean equals(Object otherObject)
```

See Also
`isEqualToSize` (page 578)

hashCode

Provides an appropriate hash code useful for storing the receiver in a hash-based data structure.

```
public int hashCode()
```

Discussion
This value is the sum of the receiver's width and height, rounded to the nearest integer.

height

Returns the height dimension of the receiver.

```
public float height()
```

See Also
width (page 578)

isEmpty

Returns whether either dimension (width or height) of the receiver is 0.

```
public boolean isEmpty()
```

isEqualToSize

Returns whether the NSSize *aSize* is equal in width and height to the receiver.

```
public boolean isEqualToSize(NSSize aSize)
```

See Also
equals (page 577)

toAWTDimension

Returns the receiver as an AWT Dimension object.

```
public java.awt.Dimension toAWTDimension()
```

Discussion
The float values of width and height are rounded up to the nearest integers with which the resulting Dimension object can enclose the original size.

See Also
toString (page 578)

toString

Returns the receiver as converted to a string object.

```
public String toString()
```

Discussion
The string has the form of "{*w*, *h*}", where *w* is the float representation of width and *h* is the float representation of height.

See Also
fromString (page 577)

width

Returns the width of the receiver.

```
public float width()
```

See Also
`height` (page 577)

Constants

NSSize provides the following constant as a convenience; you can use it to compare values returned by some NSSize methods:

Constant	Description
`ZeroSize`	An NSSize set to 0 in both dimensions

NSSortDescriptor

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.foundation
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Sort Descriptor Programming Topics

Overview

NSSortDescriptor describes how an array of objects should be sorted. Objects of this class do not actually sort the array, the sort methods are defined as a category on NSArray and NSMutableArray. Instances of NSSortDescriptor are immutable.

You construct objects of this class by specifying the property key to be compared, the order of the sort (ascending or descending), and a selector that is used to perform the comparison. If null, the selector parameter defaults to the selector `compare`. The three-argument constructor allows you to specify other comparison selectors such as `caseInsensitiveCompare` and `localizedCompare`. Sorting throws an exception if the objects to be sorted do not respond to the sort descriptor's comparison selector.

Tasks

Constructors

`NSSortDescriptor` (page 582)
Creates and returns an empty NSSortDescriptor.

Getting Information About a Sort Descriptor

`ascending` (page 582)
Returns a Boolean value that indicates whether the receiver will sort items in ascending order.

`key` (page 583)
Returns the receiver's property key.

`selector` (page 583)
Returns the selector the receiver will use when comparing objects.

Using Sort Descriptors

`compareObjects` (page 583)

`reversedSortDescriptor` (page 583)
Returns a copy of the receiver with the sort order reversed.

Constructors

NSSortDescriptor

Creates and returns an empty NSSortDescriptor.

```
public NSSortDescriptor()
```

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

Creates and returns an NSSortDescriptor with the property key specified by *key*, sort order specified by *ascending*, and the default selector `compare`.

```
public NSSortDescriptor(String key, boolean ascending)
```

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

Creates and returns an NSSortDescriptor with the property key specified by *key*, sort order specified by *ascending*, and the selector specified by *selector*.

```
public NSSortDescriptor(String key, boolean ascending, NSSelector selector)
```

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

Instance Methods

ascending

Returns a Boolean value that indicates whether the receiver will sort items in ascending order.

```
public boolean ascending()
```

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

compareObjects

```
public int compareObjects(Object object1, Object object2)
```

Discussion
Compares *object1* with *object2*, using the selector specified by the receiver. Returns `OrderedAscending` if *object1* is less than *object2*, `OrderedDescending` if *object1* is greater than *object2*, or `OrderedSame` if *object1* is equal to *object2*.

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

key

Returns the receiver's property key.

```
public String key()
```

Discussion
This key specifies the property that is compared during sorting.

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

reversedSortDescriptor

Returns a copy of the receiver with the sort order reversed.

```
public Object reversedSortDescriptor()
```

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

selector

Returns the selector the receiver will use when comparing objects.

```
public NSSelector selector()
```

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

NSSpecifierTest

Inherits from	NSScriptWhoseTest : NSObject
Package:	com.apple.cocoa.foundation
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 constructors.

When an NSSpecifierTest is properly initialized, it holds two objects:

- A "value" or "test" object (*object2*) 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 (*object1*) evaluating to the container ("words").

The instance also encapsulates a selector identifying the method performing this comparison. The interface NSComparisonMethods (page 685) defines a set of comparison methods useful for this purpose, while NSScriptingComparisonMethods (page 691) 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 544) invoked on their topmost container.

You should rarely need to subclass NSSpecifierTest.

Constants

The following constants are defined by NSSpecifierTest and are passed to the constructor to specify the comparison operator:

Constant	Description
`EqualToComparison`	Binary comparison operator that results in `true` if the two objects are equal.
`LessThanOrEqual-ToComparison`	Binary comparison operator that results in `true` if the value of the test object is equal to or less than the value of the other object.

Constant	Description
`LessThanComparison`	Binary comparison operator that results in `true` if the value of the test object is less than the value of the other object.
`GreaterThanOrEqual-ToComparison`	Binary comparison operator that results in `true` if the value of the test object is greater than or equal to the value of the other object.
`GreaterThanComparison`	Binary comparison operator that results in `true` if the value of the test object is greater than the value of the other object.
`BeginsWithComparison`	Binary containment operator that results in `true` if the test object is a list or string that matches the beginning of the other object (which is also a list or string).
`EndsWithComparison`	Binary containment operator that results in `true` if the test object is a list or string that matches the end of the other object (which is also a list or string).
`ContainsComparison`	Binary containment operator that results in `true` if the test object is a list or string that matches the other object (which is also a list or string) at any location.

Tasks

Constructors

`NSSpecifierTest` (page 586)
Returns an NSSpecifierTest with no data.

Constructors

NSSpecifierTest

Returns an NSSpecifierTest with no data.

```
public NSSpecifierTest()
```

Discussion
Do not use this constructor.

Initializes an instance of NSSpecifierTest with the object specifier supplied by *obj1*, the test object supplied by *obj2*, and the comparison operation specified by *compOp*

```
public NSSpecifierTest(NSScriptObjectSpecifier obj1, int compOp, Object obj2)
```

Discussion
. Possible comparison operations are described in "Constants" (page 585).

NSSpellServer

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Spell Checking

Overview

The NSSpellServer class gives you a way to make your particular spelling checker a service that's available to any application. A service 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

Constructors

`NSSpellServer` (page 588)
Creates an empty NSSpellServer.

Registering Your Service

`registerLanguage` (page 589)
Notifies the receiver of a language your spelling checker can check.

Assigning a Delegate

`setDelegate` (page 589)
Assigns a delegate, *`anObject`*, to the receiver.

`delegate` (page 588)
Returns the receiver's delegate.

Running the Service

`run` (page 589)
Causes the receiver to start listening for spell-checking requests.

Checking User Dictionaries

`isWordInUserDictionaries` (page 589)
Indicates whether *`word`* is in the user's list of learned words or the document's list of words to ignore.

Checking spelling

`spellServerDidForgetWord` (page 590) *delegate method*
Notifies the delegate that *`sender`* has removed *`word`* from the user's list of acceptable words in the language, *`language`*.

`spellServerDidLearnWord` (page 590) *delegate method*
Notifies the delegate that *`sender`* has added *`word`* to the user's list of acceptable words in the language, *`language`*.

`spellServerSuggestCompletionsForPartialWordRange` (page 590) *delegate method*
This delegate method returns an array of possible word completions from the spell checker, based off a partially completed string *`string`* and a given range *`range`*.

`spellServerSuggestGuessesForWord` (page 590) *delegate method*
Gives the delegate the opportunity to suggest guesses to *`sender`* for the correct spelling of the misspelled word, *`word`* in the language, *`language`*.

Constructors

NSSpellServer

Creates an empty NSSpellServer.

```
public NSSpellServer()
```

Instance Methods

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

See Also
`setDelegate` (page 589)

isWordInUserDictionaries

Indicates whether *word* is in the user's list of learned words or the document's list of words to ignore.

```
public boolean isWordInUserDictionaries(String word, boolean flag)
```

Discussion
If `true`, the word is acceptable to the user. *flag* indicates whether the comparison is to be case-sensitive.

registerLanguage

Notifies the receiver of a language your spelling checker can check.

```
public boolean registerLanguage(String language, String vendor)
```

Discussion
language is the English name of a language on Apple's list of languages. *vendor* identifies the vendor (to distinguish your spelling checker from those that others may offer for the same language). 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 list of spelling checkers.

Returns `true` if the language is registered, `false` if for some reason it can't be registered.

run

Causes the receiver to start listening for spell-checking requests.

```
public void run()
```

Discussion
This method starts a loop that never returns; you need to set the NSSpellServer's delegate before sending this message.

See Also
`setDelegate` (page 589)

setDelegate

Assigns a delegate, *anObject*, to the receiver.

```
public void setDelegate(Object anObject)
```

Discussion
Because the delegate is where the real work is done, this step is essential before telling the NSSpellServer to run.

See Also
`delegate` (page 588)
`run` (page 589)

Delegate Methods

spellServerDidForgetWord

Notifies the delegate that *sender* has removed *word* from the user's list of acceptable words in the language, *language*.

```
public abstract void spellServerDidForgetWord(NSSpellServer sender, String word,
    String language)
```

Discussion
If your delegate maintains a similar auxiliary word list, you may wish to edit the list accordingly.

spellServerDidLearnWord

Notifies the delegate that *sender* has added *word* to the user's list of acceptable words in the language, *language*.

```
public abstract void spellServerDidLearnWord(NSSpellServer sender, String word,
    String language)
```

Discussion
If your delegate maintains a similar auxiliary word list, you may wish to edit the list accordingly.

spellServerSuggestCompletionsForPartialWordRange

This delegate method returns an array of possible word completions from the spell checker, based off a partially completed string *string* and a given range *range*.

```
public abstract NSArray
    spellServerSuggestCompletionsForPartialWordRange(NSSpellServer sender, NSRange
     range, String string, String language)
```

Discussion
See `completionsForPartialWordRange` in NSSpellChecker for more information.

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

spellServerSuggestGuessesForWord

Gives the delegate the opportunity to suggest guesses to *sender* for the correct spelling of the misspelled word, *word* in the language, *language*.

```
public abstract NSArray spellServerSuggestGuessesForWord(NSSpellServer sender,
    String word, String language)
```

Discussion
Returns the guesses as an array of Strings.

NSStringReference

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	String Programming Guide for Cocoa

Overview

The NSStringReference 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. An immutable string is implemented as an array of Unicode characters (in other words, a text string).

The mutable subclass of NSStringReference is NSMutableStringReference (page 361).

The NSStringReference class has two primitive methods—`length` (page 600) and `characterAtIndex` (page 597)—that provide the basis for all other methods in its interface. The `length` (page 600) method returns the total number of Unicode characters in the string. `characterAtIndex` (page 597) gives access to each character in the string by index, with index values starting at 0.

NSStringReference 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.

String Objects

NSStringReference 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 two classes, NSStringReference and NSMutableStringReference, declare the programmatic interface for noneditable 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 NSStringReference class. To construct and manage a string that can be changed after it has been created, use NSMutableStringReference.

The objects you create using NSStringReference and NSMutableStringReference 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.

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 600) method and can retrieve a specific character with the `characterAtIndex` (page 597) 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/.

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 592)).

Make sure your reasons for subclassing NSString so 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 593)). 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 600) and `characterAtIndex` (page 597). 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.

You might want to implement an constructor for your subclass that is suited to the backing store that the subclass is managing. The NSString class adopts the NSCopying, NSMutableCopying, and NSCoding interfaces; if you want instances of your own custom subclass created from copying or coding, override the methods in these interfaces.

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 (a subclass of NSObject typically) 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.

Tasks

Constructors

`NSStringReference` (page 595)
Creates an empty NSStringReference.

Getting a String's Length

`length` (page 600)
Returns the number of Unicode characters in the receiver.

Accessing Characters

`characterAtIndex` (page 597)
Returns the character at the array position given by *`index`*.

Dividing Strings

`componentsSeparatedByString` (page 597)
Returns an NSArray containing substrings from the receiver that have been divided by *`separator`*.

`substringWithRange` (page 602)
Returns a string object containing the characters of the receiver that lie within *`aRange`*.

Finding Characters and Substrings

`rangeOfString` (page 601)
Returns an NSRange giving the location and length of the first occurrence of *`subString`* within the receiver.

Determining Line and Paragraph Ranges

`lineRangeForRange` (page 600)
Returns the smallest range of characters representing the lines containing *`aRange`*, including the characters that terminate the lines.

`paragraphRangeForRange` (page 601)
Returns the smallest range of characters representing the paragraph containing *`aRange`*, including the characters that terminate the paragraph.

Identifying and Comparing Strings

`hasPrefix` (page 600)
Returns `true` if *`aString`* matches the beginning characters of the receiver, `false` otherwise.

`hasSuffix` (page 600)
Returns `true` if *`aString`* matches the ending characters of the receiver, `false` otherwise.

`string` (page 602)
Returns a Java `String` version of the receiver.

Getting a Shared Prefix

`commonPrefixWithString` (page 597)
Returns 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.

Getting Strings with Mapping

`decomposedStringWithCanonicalMapping` (page 599)
Returns a string made by normalizing the receiver's contents.

`decomposedStringWithCompatibilityMapping` (page 599)
Returns a string made by normalizing the receiver's contents.

`precomposedStringWithCanonicalMapping` (page 601)
Returns a string made by normalizing the receiver's contents.

`precomposedStringWithCompatibilityMapping` (page 601)
Returns a string made by normalizing the receiver's contents.

Working with Encodings

`availableStringEncodings` (page 596)
Returns a zero-terminated list of the encodings string objects support in the application's environment.

`defaultCStringEncoding` (page 596)
Returns the C-string encoding assumed for any method accepting a C string as an argument

`localizedNameOfStringEncoding` (page 596)
Returns a human-readable string giving the name of *`encoding`* in the current locale's language.

`canBeConvertedToEncoding` (page 597)
Returns `true` if the receiver can be converted to *`encoding`* without loss of information. Returns `false` if characters would have to be changed or deleted in the process of changing encodings.

`dataUsingEncoding` (page 598)
Returns an NSData object containing a representation of the receiver in *`encoding`*.

`fastestEncoding` (page 599)
Returns the fastest encoding to which the receiver may be converted without loss of information.

`smallestEncoding` (page 602)
Returns the smallest encoding to which the receiver can be converted without loss of information.

Deprecated

`writeToURL` (page 603)
This method is deprecated.

Constructors

NSStringReference

Creates an empty NSStringReference.

```
public NSStringReference()
```

Creates a new NSStringReference by converting the bytes in *`aData`* into Unicode characters.

```
public NSStringReference(NSData aData, int encoding)
```

Discussion
`aData` must be 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.

Creates a new NSStringReference by reading characters from the location named by *`aURL`*.

```
public NSStringReference(java.net.URL aURL)
```

Discussion
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 characters in the default C-string encoding. Returns `null` if the location can't be opened.

Creates a new NSStringReference by converting the bytes at *aURL* into Unicode characters.

```
public NSStringReference(java.net.URL aURL, int encoding)
```

Discussion
aURL must contain bytes in *encoding* and the default plain text format (that is, pure content with no attributes or other markups) for that encoding.

Static Methods

availableStringEncodings

Returns a zero-terminated list of the encodings string objects support in the application's environment.

```
public static NSArray availableStringEncodings()
```

Discussion
Among the more commonly used are:

```
ASCIIStringEncoding
NEXTSTEPStringEncoding
UnicodeStringEncoding
ISOLatin1StringEncoding
ISOLatin2StringEncoding
SymbolStringEncoding
```

See the "Constants" (page 603) section for a larger list and descriptions of many supported encodings.

See Also
localizedNameOfStringEncoding (page 596)

defaultCStringEncoding

Returns the C-string encoding assumed for any method accepting a C string as an argument

```
public static int defaultCStringEncoding()
```

Discussion
. (These methods use ...CString... in the keywords for such arguments.) The default C-string encoding is determined from system information and can't be changed programmatically for an individual process.

localizedNameOfStringEncoding

Returns a human-readable string giving the name of *encoding* in the current locale's language.

```
public static String localizedNameOfStringEncoding(int encoding)
```

Instance Methods

canBeConvertedToEncoding

Returns `true` if the receiver can be converted to *`encoding`* without loss of information. Returns `false` if characters would have to be changed or deleted in the process of changing encodings.

```
public boolean canBeConvertedToEncoding(int encoding)
```

Discussion
If you plan to actually convert a string, `dataUsingEncoding` returns `null` on failure, so you can avoid the overhead of invoking this method yourself by simply trying to convert the string.

See Also
`dataUsingEncoding` (page 598)

characterAtIndex

Returns the character at the array position given by *`index`*.

```
public char characterAtIndex(int index)
```

Discussion
Throws a `RangeException` if *`index`* lies beyond the end of the receiver.

commonPrefixWithString

Returns 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.

```
public String commonPrefixWithString(String aString, int mask)
```

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". The following search options may be specified in *`mask`* by combining them with the C bitwise OR operator:

```
CaseInsensitiveSearch
LiteralSearch
```

See "Strings" for details on these options.

See Also
`hasPrefix` (page 600)

componentsSeparatedByString

Returns an NSArray containing substrings from the receiver that have been divided by *`separator`*.

```
public NSArray componentsSeparatedByString(String separator)
```

Discussion
The substrings in the array appear in the order they did in the receiver. If the string begins or ends with the separator, the first or last substring, respectively, is empty. For example, this code excerpt:

```
NSStringReference list = "wrenches, hammers, saws";
NSArray listItems = [list.componentsSeparatedByString (", ")];
```

produces an array with these contents:

Index	Substring
0	wrenches
1	hammers
2	saws

If *list* begins with a comma and space—for example, ", wrenches, hammers, saws"—the array has these contents:

Index	Substring
0	(empty string)
1	wrenches
2	hammers
3	saws

If *list* has no separators—for example, "wrenches"—the array contains the string itself, in this case "wrenches".

See Also
componentsJoinedByString (page 61) (NSArray)

dataUsingEncoding

Returns an NSData object containing a representation of the receiver in *encoding*.

```
public NSData dataUsingEncoding(int encoding, boolean flag)
```

Discussion
Returns null if *flag* is false and the receiver can't be converted without losing some information (such as accents or case). If *flag* is true 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 UnicodeStringEncoding to ASCIIStringEncoding, 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.

See Also
availableStringEncodings (page 596)
canBeConvertedToEncoding (page 597)

decomposedStringWithCanonicalMapping

Returns a string made by normalizing the receiver's contents.

```
public String decomposedStringWithCanonicalMapping()
```

Discussion
This method normalizes the string using the Unicode Normalization Form D.

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

See Also
precomposedStringWithCanonicalMapping (page 601)
decomposedStringWithCompatibilityMapping (page 599)

decomposedStringWithCompatibilityMapping

Returns a string made by normalizing the receiver's contents.

```
public String decomposedStringWithCompatibilityMapping()
```

Discussion
This method normalizes the string using the Unicode Normalization Form KD.

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

See Also
precomposedStringWithCompatibilityMapping (page 601)
decomposedStringWithCanonicalMapping (page 599)

fastestEncoding

Returns the fastest encoding to which the receiver may be converted without loss of information.

```
public int fastestEncoding()
```

Discussion
"Fastest" applies to retrieval of characters from the string. This encoding may not be space efficient.

See Also
smallestEncoding (page 602)

hasPrefix

Returns `true` if *aString* matches the beginning characters of the receiver, `false` otherwise.

```
public boolean hasPrefix(String aString)
```

Discussion
Returns `false` if *aString* is empty. This method is a convenience for comparing strings using the `AnchoredSearch` option. See "Strings" for more information.

See Also
hasSuffix (page 600)

hasSuffix

Returns `true` if *aString* matches the ending characters of the receiver, `false` otherwise.

```
public boolean hasSuffix(String aString)
```

Discussion
Returns `false` if *aString* is empty. This method is a convenience for comparing strings using the `AnchoredSearch` and `BackwardsSearch` options. See "Strings" for more information.

See Also
hasPrefix (page 600)

length

Returns the number of Unicode characters in the receiver.

```
public int length()
```

Discussion
This number includes the individual characters of composed character sequences, so you can't use this method to determine if a string will be visible when printed or how long it will appear.

lineRangeForRange

Returns the smallest range of characters representing the lines containing *aRange*, including the characters that terminate the lines.

```
public NSRange lineRangeForRange(NSRange aRange)
```

See Also
paragraphRangeForRange (page 601)
substringWithRange (page 602)

paragraphRangeForRange

Returns the smallest range of characters representing the paragraph containing *aRange*, including the characters that terminate the paragraph.

```
public NSRange paragraphRangeForRange(NSRange range)
```

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

See Also
lineRangeForRange (page 600)

precomposedStringWithCanonicalMapping

Returns a string made by normalizing the receiver's contents.

```
public String precomposedStringWithCanonicalMapping()
```

Discussion
This method normalizes the string using the Unicode Normalization Form C.

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

See Also
precomposedStringWithCompatibilityMapping (page 601)
decomposedStringWithCanonicalMapping (page 599)

precomposedStringWithCompatibilityMapping

Returns a string made by normalizing the receiver's contents.

```
public String precomposedStringWithCompatibilityMapping()
```

Discussion
This method normalizes the string using the Unicode Normalization Form KC.

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

See Also
precomposedStringWithCanonicalMapping (page 601)
decomposedStringWithCompatibilityMapping (page 599)

rangeOfString

Returns an NSRange giving the location and length of the first occurrence of *subString* within the receiver.

```
public NSRange rangeOfString(String subString)
```

Discussion
If *subString* isn't found, returns a range of {NSArray.NotFound, 0}. The length of the returned range and that of *subString* may differ if equivalent composed character sequences are matched.

Returns a range of {NSArray.NotFound, 0} if *subString* is the null string or empty.

Returns an NSRange giving the location and length of the first occurrence of *subString* within *aRange* in the receiver.

```
public NSRange rangeOfString(String subString, int mask, NSRange aRange)
```

Discussion
If *subString* isn't found, returns a range of {NSArray.NotFound, 0}. The length of the returned range and that of *subString* may differ if equivalent composed character sequences are matched. The following options may be specified in *mask* by combining them with the C bitwise OR operator:

```
CaseInsensitiveSearch
LiteralSearch
BackwardsSearch
AnchoredSearch
```

See "Strings" for details on these options. Throws a RangeException if any part of *aRange* lies beyond the end of the string. Returns a range of {NSArray.NotFound, 0} if *subString* is the null string or empty.

smallestEncoding

Returns the smallest encoding to which the receiver can be converted without loss of information.

```
public int smallestEncoding()
```

Discussion
This encoding may not be the fastest for accessing characters, but is very space-efficient. This method itself may take some time to execute.

See Also
fastestEncoding (page 599)

string

Returns a Java String version of the receiver.

```
public String string()
```

substringWithRange

Returns a string object containing the characters of the receiver that lie within *aRange*.

```
public String substringWithRange(NSRange aRange)
```

Discussion
Throws a `RangeException` if any part of *`aRange`* lies beyond the end of the receiver. This method treats the length of the string as a valid range value that returns an empty string.

writeToURL

```
public boolean writeToURL(java.net.URL aURL, boolean atomically)
```

Discussion
Writes the contents of the receiver to the location specified by *`aURL`*.

If *`atomically`* is `true`, the receiver is written to an auxiliary location, and then the auxiliary location is renamed to *`aURL`*. If *`atomically`* is `false`, the receiver is written directly to *`aURL`*. The `true` option guarantees that *`aURL`*, if it exists at all, won't be corrupted even if the system should crash during writing.

This method returns `true` if the location is written successfully, and `false` otherwise.

The *`atomically`* parameter is ignored if *`aURL`* is not of a type that can be accessed atomically.

See Also
`defaultCStringEncoding` (page 596)

This method is deprecated.

```
public boolean writeToURL(java.net.URL aURL, boolean atomically, int encoding)
```

Discussion

Writes the contents of the receiver, converted into encoding *`encoding`*, to the location specified by *`aURL`*, writing atomically if *`atomically`* is `true` and *`aURL`* supports it. Returns `true` if the location is written successfully, and `false` otherwise.

Availability
Deprecated.
Available in Mac OS X v10.0 and later. Deprecated in Mac OS X v10.4.

Constants

The following constants are provided by NSStringReference as possible string encodings. This is an incomplete list.

Constant	Description
`ASCIIStringEncoding`	Strict 7-bit ASCII encoding within 8-bit chars; ASCII values 0..127 only
`ISO2022JPStringEncoding`	ISO 2022 Japanese encoding for email
`ISOLatin1StringEncoding`	8-bit ISO Latin 1 encoding
`ISOLatin2StringEncoding`	8-bit ISO Latin 2 encoding

Constant	Description
`JapaneseEUCStringEncoding`	8-bit EUC encoding for Japanese text
`MacOSRomanStringEncoding`	Classic Macintosh Roman encoding
`NEXTSTEPStringEncoding`	8-bit ASCII encoding with NEXTSTEP extensions
`NonLossyASCIIStringEncoding`	7-bit verbose ASCII to represent all Unicode characters
`ShiftJISStringEncoding`	8-bit Shift-JIS encoding for Japanese text
`SymbolStringEncoding`	8-bit Adobe Symbol encoding vector
`UTF8StringEncoding`	An 8-bit representation of Unicode characters, suitable for transmission or storage by ASCII-based systems
`UnicodeStringEncoding`	The canonical Unicode encoding for string objects
`WindowsCP1250StringEncoding`	Microsoft Windows codepage 1250; equivalent to WinLatin2
`WindowsCP1251StringEncoding`	Microsoft Windows codepage 1251, encoding Cyrillic characters; equivalent to AdobeStandardCyrillic font encoding
`WindowsCP1252StringEncoding`	Microsoft Windows codepage 1252; equivalent to WinLatin1
`WindowsCP1253StringEncoding`	Microsoft Windows codepage 1253, encoding Greek characters
`WindowsCP1254StringEncoding`	Microsoft Windows codepage 1254, encoding Turkish characters

NSSystem

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Interacting with the Operating System

Overview

The NSSystem class provides methods to access process-wide information. The class object can return such information as the host name, the user's name and home directory, the process name and its arguments, and the process's environment variables. The class also provides a method, `log` (page 608), to send strings to `stderr`.

Tasks

Constructors

`NSSystem` (page 606)
Creates a new NSSystem instance.

Getting User Information

`currentFullUserName` (page 607)
Returns the full name of the current user.

`currentHomeDirectory` (page 607)
Returns a path to the current user's home directory.

`currentUserName` (page 607)
Returns the logon name of the current user.

`homeDirectoryForUser` (page 608)
Returns a path to the home directory for the user specified by *`userName`*.

Getting Framework Information

`foundationVersionNumber` (page 607)
Returns the version number for the Foundation framework.

Getting Process Information

`arguments` (page 607)
Returns the command line arguments as an array of strings.

`environment` (page 607)
Returns a dictionary of variables for the environment from which the process was launched.

`processName` (page 609)
Returns the name of the process.

Getting Host Information

`hostName` (page 608)
Returns the name of the host system.

`operatingSystem` (page 608)
Returns a constant to indicate the operating system on which the process is executing.

`operatingSystemName` (page 608)
Returns a string containing the name of the operating system on which the process is executing.

`operatingSystemVersionString` (page 608)
Returns a string containing the version of the operating system on which the process is executing.

Logging and Setting Arguments

`log` (page 608)
Logs *`aString`* to `stderr`.

`setArguments` (page 609)
Modifies the process's arguments to the elements of *`args`*.

`setProcessName` (page 609)
Sets the name of the process to *`newName`*.

Constructors

NSSystem

Creates a new NSSystem instance.

```
public NSSystem()
```

Discussion
All of the NSSystem methods are static, so there is no need create individual instances.

Static Methods

arguments

Returns the command line arguments as an array of strings.

```
public static NSArray arguments()
```

currentFullUserName

Returns the full name of the current user.

```
public static String currentFullUserName()
```

See Also
currentUserName (page 607)

currentHomeDirectory

Returns a path to the current user's home directory.

```
public static String currentHomeDirectory()
```

See Also
homeDirectoryForUser (page 608)

currentUserName

Returns the logon name of the current user.

```
public static String currentUserName()
```

See Also
currentFullUserName (page 607)

environment

Returns a dictionary of variables for the environment from which the process was launched.

```
public static NSDictionary environment()
```

Discussion
The dictionary keys are the environment variable names.

foundationVersionNumber

Returns the version number for the Foundation framework.

```
public static double foundationVersionNumber()
```

homeDirectoryForUser

Returns a path to the home directory for the user specified by *userName*.

```
public static String homeDirectoryForUser(String userName)
```

See Also
currentHomeDirectory (page 607)

hostName

Returns the name of the host system.

```
public static String hostName()
```

log

Logs *aString* to stderr.

```
public static void log(String aString)
```

operatingSystem

Returns a constant to indicate the operating system on which the process is executing.

```
public static int operatingSystem()
```

Discussion
See "Constants" (page 610) for a list of possible values.

operatingSystemName

Returns a string containing the name of the operating system on which the process is executing.

```
public static String operatingSystemName()
```

Discussion
In Mac OS X, this returns the string "NSMACHOperatingSystem".

operatingSystemVersionString

Returns a string containing the version of the operating system on which the process is executing.

```
public static String operatingSystemVersionString()
```

Discussion
This string is human readable, localized, and is appropriate for displaying to the user or using in bug emails and such. This string is NOT appropriate for parsing

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

processName

Returns the name of the process.

```
public static String processName()
```

Discussion
This name is used to register application defaults and is in error messages. It does not uniquely identify the process.

See Also
`setProcessName` (page 609)

setArguments

Modifies the process's arguments to the elements of *args*.

```
public static void setArguments(String[] args)
```

Discussion
A "`java`" argument is automatically inserted as the process's first argument.

Modifies the process's arguments to the elements of *args*.

```
public static void setArguments(NSArray args)
```

See Also
`arguments` (page 607)

setProcessName

Sets the name of the process to *newName*.

```
public static void setProcessName(String newName)
```

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.

See Also
`processName` (page 609)

Constants

The following constants are provided by NSSystem as return values from `operatingSystem` (page 608):

Constant	Description
`HPUXOperatingSystem`	Indicates the HP UX operating system.
`MACHOperatingSystem`	Indicates the Mac OS X operating system.
`SolarisOperatingSystem`	Indicates the Solaris operating system.
`Windows95OperatingSystem`	Indicates the Windows 95 operating system.
`WindowsNTOperatingSystem`	Indicates the Windows NT operating system.

The following constant can be used to determine if you are using a version of the Foundation framework newer than the version delivered in Mac OS X v10.0:

Constant	Description
`FoundationVersionNumber10_0`	The Foundation framework included in Mac OS X v10.0.
`FoundationVersionNumber10_1`	The Foundation framework included in Mac OS X v10.1.

NSTimer

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guides	Timer Programming Topics for Cocoa Run Loops

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 that sends a message to a window, telling it to update itself after a certain time interval.

If you specify in the constructor 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, send the timer the `invalidate` (page 612) message from the same thread on which the timer was installed. This message immediately disables the timer, so it no longer affects the NSRunLoop. The NSRunLoop removes and releases the timer, either just before the `invalidate` (page 612) method returns or at some later point.

Tasks

Constructors

`NSTimer` (page 612)
Returns an empty NSTimer.

Stopping a Timer

`invalidate` (page 612)
Stops the receiver from ever firing again and requests its removal from its NSRunLoop.

Information About a Timer

`isValid` (page 613)
Returns `true` if the receiver is currently valid, `false` otherwise.

`timeInterval` (page 613)
Returns the receiver's time interval.

`userInfo` (page 613)
Returns the *`userInfo`* object, containing additional data the target may use when the receiver is fired.

Constructors

NSTimer

Returns an empty NSTimer.

```
public NSTimer()
```

Creates a new NSTimer that, when added to a run loop, will fire after *`seconds`*.

```
public NSTimer(double seconds, Object target, NSSelector aSelector, Object userInfo,
     boolean repeats)
```

Discussion
Upon firing, the timer sends *`aSelector`* to *`target`*. The *`aSelector`* method must have the following syntax:

```
public void myTimerFireMethod(NSTimer* theTimer)
```

The timer passes itself as the argument to *`aSelector`*. To pass more information to the target, use *`userInfo`*. The target gets *`userInfo`* by sending `userInfo` (page 613) to the timer.

If *`seconds`* is less than or equal to 0.0, this method chooses a nonnegative interval. If *`repeats`* is `true`, the timer will repeatedly reschedule itself until invalidated. If *`repeats`* is `false`, the timer will be invalidated after it fires.

Instance Methods

invalidate

Stops the receiver from ever firing again and requests its removal from its NSRunLoop.

```
public void invalidate()
```

Discussion
This is the only way to remove a timer from an NSRunLoop. The NSRunLoop removes and releases the timer, either just before the `invalidate` (page 612) method returns or at some later point.

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.

isValid

Returns `true` if the receiver is currently valid, `false` otherwise.

```
public boolean isValid()
```

timeInterval

Returns the receiver's time interval.

```
public double timeInterval()
```

userInfo

Returns the *`userInfo`* object, containing additional data the target may use when the receiver is fired.

```
public Object userInfo()
```

Discussion
Do not invoke this method after the timer is invalidated. Use `isValid` (page 613) to test whether the timer is valid.

NSTimeZone

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.foundation
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 static methods to get time zone objects: `timeZoneWithName` (page 619), `timeZoneWithNameAndData` (page 620), and `timeZoneForSecondsFromGMT` (page 619). The class also permits you to set the default time zone within your application (`setDefaultTimeZone` (page 619)). You can access this default time zone at any time with the `defaultTimeZone` (page 618) static method, and with the `localTimeZone` (page 618) static 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.

Some NSGregorianDate 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 NSGregorianDate are bound to the default time zone for the current locale.

Tasks

Constructors

`NSTimeZone` (page 617)

Getting Time Zones

`timeZoneWithName` (page 619)

Returns the time zone object identified by the name *aTimeZoneName*.

`timeZoneWithNameAndData` (page 620)
Returns the time zone with the name *aTimeZoneName* whose data has been initialized using the contents of *data*.

`timeZoneForSecondsFromGMT` (page 619)
Returns a time zone object offset from Greenwich Mean Time by *seconds*.

Getting the Default Time Zone

`localTimeZone` (page 618)
Returns an object that forwards all messages to the default time zone for your application.

`defaultTimeZone` (page 618)
Returns the default time zone set for your application.

`setDefaultTimeZone` (page 619)
Sets the default time zone for your application to *aTimeZone*.

`resetSystemTimeZone` (page 619)
Clears the previously determined system time zone, if any.

`systemTimeZone` (page 619)
Returns the time zone currently used by the system.

Getting Time Zone Information

`abbreviationDictionary` (page 618)
Returns a dictionary holding the mappings of time zone abbreviations to time zone names.

`knownTimeZoneNames` (page 618)
Returns an array of strings listing the names of all the time zones known to the system.

Getting Information About a Specific Time Zone

`abbreviation` (page 620)
Returns the abbreviation for the receiver, such as “EDT” (Eastern Daylight Time).

`abbreviationForDate` (page 620)
Returns the abbreviation for the receiver at the specified date.

`name` (page 621)
Returns the geopolitical region name that identifies the receiver.

`secondsFromGMT` (page 622)
Returns the current difference in seconds between the receiver and Greenwich Mean Time.

`secondsFromGMTForDate` (page 622)
Returns the difference in seconds between the receiver and Greenwich Mean Time at *aDate*.

`isDaylightSavingTime` (page 621)
Returns `true` if the receiver is currently using daylight savings time.

`isDaylightSavingTimeForDate` (page 621)
Returns `true` if the receiver uses daylight savings time at *aDate*.

`data` (page 620)
Returns the data that stores the information used by the receiver.

Comparing Time Zones

`equals` (page 621)
Returns `true` if *`anObject`* is an instance of NSTimeZone and satisfies `isEqualToTimeZone` (page 621).

`isEqualToTimeZone` (page 621)
Returns `true` if *`aTimeZone`* and the receiver have the same name and data.

Describing a Time Zone

`toString` (page 622)

`hashCode` (page 621)
Returns an integer that can be used as a table address in a hash table structure.

Constructors

NSTimeZone

```
public NSTimeZone()
```

Discussion
This constructor has been deprecated. Use any of the `timeZone...` static methods instead.

```
public NSTimeZone(int seconds)
```

Discussion
This constructor has been deprecated. Use `timeZoneForSecondsFromGMT` (page 619) instead.

```
public NSTimeZone(String aTimeZoneName, NSData data)
```

Discussion
This constructor has been deprecated. Use `timeZoneWithNameAndData` (page 620) instead.

```
public NSTimeZone(String aTimeZoneName, boolean isAbbrev)
```

Discussion
This constructor has been deprecated. Use `timeZoneWithName` (page 619) instead.

Static Methods

abbreviationDictionary

Returns a dictionary holding the mappings of time zone abbreviations to time zone names.

```
public static NSDictionary abbreviationDictionary()
```

Discussion
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.

defaultTimeZone

Returns the default time zone set for your application.

```
public static NSTimeZone defaultTimeZone()
```

Discussion
If no default time zone has been set, this method invokes `systemTimeZone` (page 619) and returns the system time zone.

See Also
`localTimeZone` (page 618)
`setDefaultTimeZone` (page 619)
`systemTimeZone` (page 619)

knownTimeZoneNames

Returns an array of strings listing the names of all the time zones known to the system.

```
public static NSArray knownTimeZoneNames()
```

localTimeZone

Returns an object that forwards all messages to the default time zone for your application.

```
public static NSTimeZone localTimeZone()
```

Discussion
This behavior is particularly useful for NSGregorianDate objects that are archived or sent as distributed objects and may be interpreted in different locales.

See Also
`defaultTimeZone` (page 618)
`setDefaultTimeZone` (page 619)

resetSystemTimeZone

Clears the previously determined system time zone, if any.

```
public static void resetSystemTimeZone()
```

Discussion
This method also resets the default time zone if it is the same as the system time zone. Subsequent calls to `systemTimeZone` (page 619) will attempt to redetermine the system time zone.

setDefaultTimeZone

Sets the default time zone for your application to *aTimeZone*.

```
public static void setDefaultTimeZone(NSTimeZone aTimeZone)
```

Discussion
There can be only one default time zone, so by setting a new default time zone, you lose the previous one.

See Also
`defaultTimeZone` (page 618)
`localTimeZone` (page 618)

systemTimeZone

Returns the time zone currently used by the system.

```
public static NSTimeZone systemTimeZone()
```

Discussion
If it can't figure out the current time zone, returns the GMT time zone.

timeZoneForSecondsFromGMT

Returns a time zone object offset from Greenwich Mean Time by *seconds*.

```
public static Object timeZoneForSecondsFromGMT(int 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.

See Also
`timeZoneWithName` (page 619)

timeZoneWithName

Returns the time zone object identified by the name *aTimeZoneName*.

```
public static Object timeZoneWithName(String aTimeZoneName, boolean flag)
```

Discussion
It searches the time zone information directory for matching names. Returns `null` if there is no match for the name.

See Also
`timeZoneForSecondsFromGMT` (page 619)
`knownTimeZoneNames` (page 618)

timeZoneWithNameAndData

Returns the time zone with the name *aTimeZoneName* whose data has been initialized using the contents of *data*.

```
public static Object timeZoneWithNameAndData(String aTimeZoneName, NSData data)
```

Discussion
You should not call this method directly—use `timeZoneWithName` (page 619) to get the time zone object for a given name.

See Also
`timeZoneWithName` (page 619)

Instance Methods

abbreviation

Returns the abbreviation for the receiver, such as "EDT" (Eastern Daylight Time).

```
public String abbreviation()
```

Discussion
Invokes `abbreviationForDate` (page 620) with the current date as the argument.

abbreviationForDate

Returns the abbreviation for the receiver at the specified date.

```
public String abbreviationForDate(NSDate 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."

data

Returns the data that stores the information used by the receiver.

```
public NSData data()
```

Discussion
This data should be treated as an opaque object.

equals

Returns `true` if *anObject* is an instance of NSTimeZone and satisfies `isEqualToTimeZone` (page 621).

```
public boolean equals(Object anObject)
```

Discussion
Returns `false` otherwise.

hashCode

Returns an integer that can be used as a table address in a hash table structure.

```
public int hashCode()
```

isDaylightSavingTime

Returns `true` if the receiver is currently using daylight savings time.

```
public boolean isDaylightSavingTime()
```

Discussion
This method invokes `isDaylightSavingTimeForDate` (page 621) with the current date as the argument.

isDaylightSavingTimeForDate

Returns `true` if the receiver uses daylight savings time at *aDate*.

```
public boolean isDaylightSavingTimeForDate(NSDate aDate)
```

isEqualToTimeZone

Returns `true` if *aTimeZone* and the receiver have the same name and data.

```
public boolean isEqualToTimeZone(NSTimeZone aTimeZone)
```

name

Returns the geopolitical region name that identifies the receiver.

```
public String name()
```

secondsFromGMT

Returns the current difference in seconds between the receiver and Greenwich Mean Time.

```
public int secondsFromGMT()
```

secondsFromGMTForDate

Returns the difference in seconds between the receiver and Greenwich Mean Time at *aDate*.

```
public int secondsFromGMTForDate(NSDate aDate)
```

Discussion
This 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.

toString

```
public String toString()
```

Discussion
Returns a string description of the receiver, including the name, abbreviation, offset from GMT, and whether or not daylight savings time is currently in effect.

NSUnarchiver

Inherits from	NSCoder : NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Archives and Serializations Programming Guide for Cocoa

Overview

NSUnarchiver, a concrete subclass of NSCoder, defines methods for decoding a set of objects from an archive. Such archives are produced by objects of the NSArchiver (page 47) class.

Tasks

Constructors

`NSUnarchiver` (page 624)
Creates an empty NSUnarchiver.

Decoding Objects

`decodeByte` (page 627)
Decodes and returns a `byte` value that was previously encoded with `encodeByte` (page 51).

`decodeChar` (page 627)
Decodes and returns a `char` value that was previously encoded with `encodeChar` (page 51).

`decodeDataObject` (page 627)
Decodes and returns an `NSData` object that was previously encoded with `encodeDataObject` (page 52).

`decodeDouble` (page 627)
Decodes and returns a `double` value that was previously encoded with `encodeDouble` (page 52).

`decodeFloat` (page 628)
Decodes and returns a `float` value that was previously encoded with `encodeFloat` (page 52).

`decodeInt` (page 628)
Decodes and returns an `int` value that was previously encoded with `encodeInt` (page 53).

`decodeLong` (page 628)
Decodes and returns a `long` value that was previously encoded with `encodeLong` (page 53).

`decodeObject` (page 628)
Decodes and returns an `Object` object that was previously encoded with `encodeObject` (page 53).

`decodeShort` (page 628)
Decodes and returns a `short` value that was previously encoded with `encodeShort` (page 53).

`unarchiveObjectWithData` (page 626)
Decodes and returns the object archived in *`data`*.

`unarchiveObjectWithFile` (page 626)
Decodes and returns the object archived in the file *`path`*.

Managing an NSUnarchiver

`isAtEnd` (page 628)
Returns `true` if the receiver has reached the end of the encoded data while decoding, `false` if more data follows.

`data` (page 627)
Returns the archive data.

`versionForClassName` (page 629)
Returns the version number for the archived implementation of the class named *`className`* or `NSArray.NotFound` if no class named *`className`* exists in the archive.

Substituting Classes or Objects

`classNameGloballyDecodedForArchiveClassName` (page 625)
Returns the name of the class used when instantiating objects whose ostensible class, according to the archived data, is *`nameInArchive`*.

`globallyDecodeClassNameAsClassName` (page 625)
Instructs instances of NSUnarchiver to use the class named *`trueName`* when instantiating objects whose ostensible class, according to the archived data, is *`nameInArchive`*.

`classNameDecodedForArchiveClassName` (page 626)
Returns the name of the class that will be used when instantiating objects whose ostensible class, according to the archived data, is *`nameInArchive`*.

`decodeClassNameAsClassName` (page 627)
Instructs the receiver to use the class named *`trueName`* when instantiating objects whose ostensible class, according to the archived data, is *`nameInArchive`*.

`replaceObject` (page 628)
Causes the receiver to substitute *`newObject`* for *`object`* whenever *`object`* is extracted from the archive.

Constructors

NSUnarchiver

Creates an empty NSUnarchiver.

```
public NSUnarchiver()
```

Discussion
Use the other constructor or the static methods `unarchiveObjectWithData` (page 626) or `unarchiveObjectWithFile` (page 626), instead.

Creates an NSUnarchiver from the data object *data* and prepares the NSUnarchiver for a subsequent invocation of `decodeObject` (page 628).

```
public NSUnarchiver(NSData data)
```

Discussion
Throws an `InvalidArgumentException` if *data* is `null`.

Static Methods

classNameGloballyDecodedForArchiveClassName

Returns the name of the class used when instantiating objects whose ostensible class, according to the archived data, is *nameInArchive*.

```
public static String classNameGloballyDecodedForArchiveClassName(String
    nameInArchive)
```

Discussion
This method returns *nameInArchive* if no substitute name has been specified using the static method (not the instance method) `globallyDecodeClassNameAsClassName` (page 625).

Note that each individual instance of NSUnarchiver can be given its own class name mappings by invoking the instance method `decodeClassNameAsClassName` (page 627). The NSUnarchiver class has no information about these instance-specific mappings, however, so they don't affect the return value of `classNameGloballyDecodedForArchiveClassName`.

See Also
`classNameDecodedForArchiveClassName` (page 626)

globallyDecodeClassNameAsClassName

Instructs instances of NSUnarchiver to use the class named *trueName* when instantiating objects whose ostensible class, according to the archived data, is *nameInArchive*.

```
public static void globallyDecodeClassNameAsClassName(String nameInArchive, String
     trueName)
```

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 a similar 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.

See Also
`classNameGloballyDecodedForArchiveClassName` (page 625)
`decodeClassNameAsClassName` (page 627)

unarchiveObjectWithData

Decodes and returns the object archived in *data*.

```
public static Object unarchiveObjectWithData(NSData data)
```

Discussion
This method invokes `decodeObject` to create a temporary NSUnarchiver that decodes the object. If the archived object is the root of a graph of objects, the entire graph is unarchived.

Archives are produced by objects of the NSArchiver (page 47) class.

See Also
`encodeRootObject` (page 53) (NSArchiver)

unarchiveObjectWithFile

Decodes and returns the object archived in the file *path*.

```
public static Object unarchiveObjectWithFile(String path)
```

Discussion
This convenience method reads the file and then invokes `unarchiveObjectWithData` (page 626).

Archives are produced by objects of the NSArchiver (page 47) class.

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 *nameInArchive*.

```
public String classNameDecodedForArchiveClassName(String nameInArchive)
```

Discussion
This method returns *nameInArchive* unless a substitute name has been specified using the instance method (not the static method) `decodeClassNameAsClassName` (page 627).

See Also
`classNameGloballyDecodedForArchiveClassName` (page 625)

data

Returns the archive data.

```
public NSData data()
```

Discussion
The returned data object is the same one specified as the argument to the constructor.

decodeByte

Decodes and returns a `byte` value that was previously encoded with `encodeByte` (page 51).

```
public byte decodeByte()
```

decodeChar

Decodes and returns a `char` value that was previously encoded with `encodeChar` (page 51).

```
public char decodeChar()
```

decodeClassNameAsClassName

Instructs the receiver to use the class named *`trueName`* when instantiating objects whose ostensible class, according to the archived data, is *`nameInArchive`*.

```
public void decodeClassNameAsClassName(String nameInArchive, String trueName)
```

Discussion
This method enables easy conversion of unarchived data when the name of a class has changed since the archive was created.

See Also
`classNameDecodedForArchiveClassName` (page 626)
`globallyDecodeClassNameAsClassName` (page 625)

decodeDataObject

Decodes and returns an `NSData` object that was previously encoded with `encodeDataObject` (page 52).

```
public NSData decodeDataObject()
```

decodeDouble

Decodes and returns a `double` value that was previously encoded with `encodeDouble` (page 52).

```
public double decodeDouble()
```

decodeFloat

Decodes and returns a `float` value that was previously encoded with `encodeFloat` (page 52).

```
public float decodeFloat()
```

decodeInt

Decodes and returns an `int` value that was previously encoded with `encodeInt` (page 53).

```
public int decodeInt()
```

decodeLong

Decodes and returns a `long` value that was previously encoded with `encodeLong` (page 53).

```
public long decodeLong()
```

decodeObject

Decodes and returns an `Object` object that was previously encoded with `encodeObject` (page 53).

```
public Object decodeObject()
```

decodeShort

Decodes and returns a `short` value that was previously encoded with `encodeShort` (page 53).

```
public short decodeShort()
```

isAtEnd

Returns `true` if the receiver has reached the end of the encoded data while decoding, `false` if more data follows.

```
public boolean isAtEnd()
```

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.

replaceObject

Causes the receiver to substitute *`newObject`* for *`object`* whenever *`object`* is extracted from the archive.

```
public void replaceObject(Object object, Object newObject)
```

Discussion
newObject can be of a different class from object, and the class mappings set by `classNameGloballyDecodedForArchiveClassName` (page 625) and `decodeClassNameAsClassName` (page 627) are ignored.

versionForClassName

Returns the version number for the archived implementation of the class named *className* or `NSArray.NotFound` if no class named *className* exists in the archive.

```
public int versionForClassName(String className)
```

Discussion
The class version number of each encoded object is written to the archive so that newer versions of the class can detect and properly decode older archived versions.

NSUndoManager

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	Undo Architecture

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. NSUndoManager groups all operations within a single cycle of the run loop, so that performing an undo reverts all changes that occurred during the loop. Also, when performing undo an NSUndoManager saves the operations reverted so that you can redo the undos.

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

Constructors

`NSUndoManager` (page 634)
Creates an empty NSUndoManager.

Registering Undo Operations

`registerUndoWithTarget` (page 638)
Records a single undo operation for *`target`*, so that when an undo is performed it is sent *`aSelector`* with *`anObject`* as the sole argument.

`registerUndoWithTargetAndArguments` (page 639)
Records a single undo operation for *`target`*, so that when an undo is performed it is sent *`aSelector`* with multiple arguments specified in *`objects`*.

Checking Undo Ability

`canUndo` (page 635)
Returns `true` if the receiver has any actions to undo, `false` if it doesn't.

`canRedo` (page 634)
Returns `true` if the receiver has any actions to redo, `false` if it doesn't.

Performing Undo and Redo

`undo` (page 641)
Closes the top-level undo group if necessary and invokes undoNestedGroup.

`undoNestedGroup` (page 642)
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 637)
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 640)
Sets the maximum number of top-level undo groups the receiver holds to *`anInt`*.

`levelsOfUndo` (page 637)
Returns the maximum number of top-level undo groups the receiver holds.

Creating Undo Groups

`beginUndoGrouping` (page 634)
Marks the beginning of an undo group.

`endUndoGrouping` (page 635)
Marks the end of an undo group.

`enableUndoRegistration` (page 635)
Enables the recording of undo operations.

`groupsByEvent` (page 636)
Returns `true` if the receiver automatically creates undo groups around each pass of the run loop, `false` if it doesn't.

`setGroupsByEvent` (page 640)
Sets a Boolean value that specifies whether the receiver automatically groups undo operations during the run loop.

`groupingLevel` (page 636)
Returns the number of nested undo groups (or redo groups, if Redo was last invoked) in the current event loop.

Disabling Undo

`disableUndoRegistration` (page 635)
Disables the recording of undo operations by registerUndoWithTarget.

`isUndoRegistrationEnabled` (page 637)
Returns a Boolean value that indicates whether the recording of undo operations is enabled.

Checking Whether Undo or Redo Is Being Performed

`isUndoing` (page 637)
Returns `true` if the receiver is in the process of performing its undo or undoNestedGroup method, `false` otherwise.

`isRedoing` (page 636)
Returns `true` if the receiver is in the process of performing its redo method, `false` otherwise.

Clearing Undo Operations

`removeAllActions` (page 639)
Clears the undo and redo stacks and reenables the receiver.

`removeAllActionsWithTarget` (page 639)
Clears the undo and redo stacks of all operations involving *`target`* as the recipient of the undo message.

Setting and Getting the Action Name

`setActionName` (page 640)
Sets the name of the action associated with the Undo or Redo command to *`actionName`*.

`redoActionName` (page 638)
Returns the name identifying the redo action.

`undoActionName` (page 641)
Returns the name identifying the undo action.

Getting and Localizing Menu Item Title

`redoMenuItemTitle` (page 638)
Returns the complete title of the Redo menu command, for example, “Redo Paste.”

`undoMenuItemTitle` (page 642)
Returns the complete title of the Undo menu command, for example, “Undo Paste.”

`redoMenuTitleForUndoActionName` (page 638)
Returns the complete, localized title of the Redo menu command for the action identified by *`actionName`*.

`undoMenuTitleForUndoActionName` (page 642)
Returns the complete, localized title of the Undo menu command for the action identified by *`actionName`*.

Working with Run Loops

`runLoopModes` (page 639)
Returns the modes governing the types of input handled during a cycle of the run loop.

`setRunLoopModes` (page 641)
Sets the modes that determine the types of input handled during a cycle of the run loop.

Constructors

NSUndoManager

Creates an empty NSUndoManager.

```
public NSUndoManager()
```

Instance Methods

beginUndoGrouping

Marks the beginning of an undo group.

```
public void beginUndoGrouping()
```

Discussion
All individual undo operations before a subsequent `endUndoGrouping` (page 635) message are grouped together and reversed by a later `undo` (page 641) 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 `CheckpointNotification` (page 643) unless a top-level undo is in progress. It posts an `DidOpenUndoGroupNotification` (page 643) if a new group was successfully created.

canRedo

Returns `true` if the receiver has any actions to redo, `false` if it doesn't.

```
public boolean canRedo()
```

Discussion
Because any undo operation registered clears the redo stack, this method posts an `CheckpointNotification` (page 643) to allow clients to apply their pending operations before testing the redo stack.

See Also
`canUndo` (page 635)
`redo` (page 637)

canUndo

Returns `true` if the receiver has any actions to undo, `false` if it doesn't.

```
public boolean canUndo()
```

Discussion
This fact does not mean you can safely invoke `undo` (page 641) or `undoNestedGroup` (page 642)—you may have to close open undo groups first.

See Also
`canRedo` (page 634)
`enableUndoRegistration` (page 635)
`registerUndoWithTarget` (page 638)

disableUndoRegistration

Disables the recording of undo operations by registerUndoWithTarget.

```
public void disableUndoRegistration()
```

Discussion
This method can be invoked multiple times by multiple clients. `enableUndoRegistration` (page 635) must be invoked an equal number of times to reenable undo registration.

enableUndoRegistration

Enables the recording of undo operations.

```
public void enableUndoRegistration()
```

Discussion
Because undo registration is enabled by default, it is often used to balance a prior `disableUndoRegistration` (page 635) message. Undo registration isn't actually reenabled until an enable message balances the last disable message in effect. Throws an `InternalInconsistencyException` if invoked while no `disableUndoRegistration` (page 635) message is in effect.

endUndoGrouping

Marks the end of an undo group.

```
public void endUndoGrouping()
```

Discussion
All individual undo operations back to the matching `beginUndoGrouping` (page 634) message are grouped together and reversed by a later `undo` (page 641) or `undoNestedGroup` (page 642) message. Undo groups can be nested, thus providing functionality similar to nested transactions. Throws an `InternalInconsistencyException` if there's no `beginUndoGrouping` (page 634) message in effect.

This method posts an `CheckpointNotification` (page 643) and an `WillCloseUndoGroupNotification` (page 643) just before the group is closed.

See Also
`levelsOfUndo` (page 637)

groupingLevel

Returns the number of nested undo groups (or redo groups, if Redo was last invoked) in the current event loop.

```
public int groupingLevel()
```

Discussion
If 0 is returned, there is no open undo or redo group.

See Also
`levelsOfUndo` (page 637)
`setLevelsOfUndo` (page 640)

groupsByEvent

Returns `true` if the receiver automatically creates undo groups around each pass of the run loop, `false` if it doesn't.

```
public boolean groupsByEvent()
```

Discussion
The default is `true`.

See Also
`beginUndoGrouping` (page 634)
`setGroupsByEvent` (page 640)

isRedoing

Returns `true` if the receiver is in the process of performing its redo method, `false` otherwise.

```
public boolean isRedoing()
```

See Also
`isUndoing` (page 637)

isUndoing

Returns `true` if the receiver is in the process of performing its undo or undoNestedGroup method, `false` otherwise.

```
public boolean isUndoing()
```

See Also
`isRedoing` (page 636)

isUndoRegistrationEnabled

Returns a Boolean value that indicates whether the recording of undo operations is enabled.

```
public boolean isUndoRegistrationEnabled()
```

Discussion
Undo registration is enabled by default.

See Also
`disableUndoRegistration` (page 635)
`enableUndoRegistration` (page 635)

levelsOfUndo

Returns the maximum number of top-level undo groups the receiver holds.

```
public int levelsOfUndo()
```

Discussion
When ending an undo group results in the number of groups exceeding this limit, the oldest groups are dropped from the stack. A limit of 0 indicates no limit, so old undo groups are never dropped. The default is 0.

See Also
`enableUndoRegistration` (page 635)
`setLevelsOfUndo` (page 640)

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.

```
public void redo()
```

Discussion
Throws an `InternalInconsistencyException` if the method is invoked during an undo operation.

This method posts an `CheckpointNotification` (page 643) and `WillRedoChangeNotification` (page 643) before it performs the redo operation, and it posts the `DidRedoChangeNotification` (page 643) after it performs the redo operation.

See Also
registerUndoWithTarget (page 638)

redoActionName

Returns the name identifying the redo action.

```
public String redoActionName()
```

Discussion
For example, if the menu title is "Redo Delete," the string returned is "Delete." Returns an empty string if no action name has been assigned or `null` if there is nothing to redo.

See Also
setActionName (page 640)
undoActionName (page 641)

redoMenuItemTitle

Returns the complete title of the Redo menu command, for example, "Redo Paste."

```
public String redoMenuItemTitle()
```

Discussion
Returns "Redo" if no action name has been assigned or `null` if there is nothing to redo.

See Also
undoMenuItemTitle (page 642)

redoMenuTitleForUndoActionName

Returns the complete, localized title of the Redo menu command for the action identified by *actionName*.

```
public String redoMenuTitleForUndoActionName(String actionName)
```

Discussion
Override this method if you want to customize the localization behavior. This method is invoked by redoMenuItemTitle (page 638).

See Also
undoMenuTitleForUndoActionName (page 642)

registerUndoWithTarget

Records a single undo operation for *target*, so that when an undo is performed it is sent *aSelector* with *anObject* as the sole argument.

```
public void registerUndoWithTarget(Object target, NSSelector aSelector, Object
    anObject)
```

Discussion
Also clears the redo stack. See "Registering Undo Operations" for more information.

Throws an `InternalInconsistencyException` if invoked when no undo group has been established using `beginUndoGrouping` (page 634). Undo groups are normally set by default, so you should rarely need to begin a top-level undo group explicitly.

See Also
`undoNestedGroup` (page 642)
`groupingLevel` (page 636)

registerUndoWithTargetAndArguments

Records a single undo operation for *target*, so that when an undo is performed it is sent *aSelector* with multiple arguments specified in *objects*.

```
public void registerUndoWithTargetAndArguments(Object target, NSSelector aSelector,
    Object[] objects)
```

See Also
`registerUndoWithTarget` (page 638)

removeAllActions

Clears the undo and redo stacks and reenables the receiver.

```
public void removeAllActions()
```

See Also
`enableUndoRegistration` (page 635)
`removeAllActionsWithTarget` (page 639)

removeAllActionsWithTarget

Clears the undo and redo stacks of all operations involving *target* as the recipient of the undo message.

```
public void removeAllActionsWithTarget(Object target)
```

Discussion
Doesn't reenable the receiver if it's disabled. An object that shares an NSUndoManager with other clients should invoke this message in its implementation of `dealloc`.

See Also
`enableUndoRegistration` (page 635)
`removeAllActions` (page 639)

runLoopModes

Returns the modes governing the types of input handled during a cycle of the run loop.

```
public NSArray runLoopModes()
```

Discussion
By default, the sole run-loop mode is `NSRunLoop.DefaultRunLoopMode` (which excludes data from NSConnections).

See Also
`setRunLoopModes` (page 641)

setActionName

Sets the name of the action associated with the Undo or Redo command to *actionName*.

```
public void setActionName(String actionName)
```

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 `null`.

See Also
`redoActionName` (page 638)
`undoActionName` (page 641)

setGroupsByEvent

Sets a Boolean value that specifies whether the receiver automatically groups undo operations during the run loop.

```
public void setGroupsByEvent(boolean flag)
```

Discussion
If *flag* is `true`, the receiver creates undo groups around each pass through the run loop; if flag is `false` it doesn't. The default is `true`.

If you turn automatic grouping off, you must close groups explicitly before invoking either `undo` (page 641) or `undoNestedGroup` (page 642).

See Also
`groupingLevel` (page 636)
`groupsByEvent` (page 636)

setLevelsOfUndo

Sets the maximum number of top-level undo groups the receiver holds to *anInt*.

```
public void setLevelsOfUndo(int anInt)
```

Discussion
When ending an undo group results in the number of groups exceeding this limit, the oldest groups are dropped from the stack. A limit of 0 indicates no limit, so that old undo groups are never dropped. The default is 0.

If invoked with a limit below the prior limit, old undo groups are immediately dropped.

See Also
`enableUndoRegistration` (page 635)
`levelsOfUndo` (page 637)

setRunLoopModes

Sets the modes that determine the types of input handled during a cycle of the run loop.

```
public void setRunLoopModes(NSArray modes)
```

Discussion
By default, the sole run-loop mode is `NSRunLoop.DefaultRunLoopMode` (which excludes data from NSConnections). With this method, you could limit the input to data received during a mouse-tracking session by setting the mode to `NSApplication.EventTrackingRunLoopMode`, or you could limit it to data received from a modal panel with `NSApplication.ModalPanelRunLoopMode`.

See Also
`runLoopModes` (page 639)

undo

Closes the top-level undo group if necessary and invokes undoNestedGroup.

```
public void undo()
```

Discussion
It also invokes `endUndoGrouping` (page 635) if the nesting level is 1. Throws an `InternalInconsistencyException` if more than one undo group is open (that is, if the last group isn't at the top level).

This method posts an `CheckpointNotification` (page 643).

See Also
`enableUndoRegistration` (page 635)
`groupingLevel` (page 636)

undoActionName

Returns the name identifying the undo action.

```
public String undoActionName()
```

Discussion
For example, if the menu title is "Undo Delete," the string returned is "Delete." Returns an empty string if no action name has been assigned or `null` if there is nothing to undo.

See Also
`redoActionName` (page 638)
`setActionName` (page 640)

undoMenuItemTitle

Returns the complete title of the Undo menu command, for example, “Undo Paste.”

```
public String undoMenuItemTitle()
```

Discussion
Returns “Undo” if no action name has been assigned or `null` if there is nothing to undo.

See Also
`redoMenuItemTitle` (page 638)

undoMenuTitleForUndoActionName

Returns the complete, localized title of the Undo menu command for the action identified by *actionName*.

```
public String undoMenuTitleForUndoActionName(String actionName)
```

Discussion
Override this method if you want to customize the localization behavior. This method is invoked by `undoMenuItemTitle` (page 642).

See Also
`redoMenuTitleForUndoActionName` (page 638)

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.

```
public void undoNestedGroup()
```

Discussion
Throws an `InternalInconsistencyException` if any undo operations have been registered since the last `enableUndoRegistration` (page 635) message.

This method posts an `CheckpointNotification` (page 643) and `WillUndoChangeNotification` (page 643) before it performs the undo operation, and it posts an `DidUndoChangeNotification` (page 643) after it performs the undo operation.

See Also
`undo` (page 641)

Constants

NSUndoManager provides the following constant as a convenience; you can use it to compare to values returned by some NSUndoManager methods:

Constant	Description
`UndoCloseGroupingRunLoopOrdering`	Used with NSRunLoop's `performSelectorWithOrder` (page 501).

Notifications

CheckpointNotification

Posted whenever an NSUndoManager opens or closes an undo group (except when it opens a top-level group) and when checking the redo stack in `canRedo` (page 634). The notification object is the NSUndoManager. This notification does not contain a *`userInfo`* dictionary.

DidOpenUndoGroupNotification

Posted whenever an NSUndoManager opens an undo group, which occurs in the implementation of the `beginUndoGrouping` (page 634) method. The notification object is the NSUndoManager. This notification does not contain a *`userInfo`* dictionary.

DidRedoChangeNotification

Posted just after an NSUndoManager performs a redo operation (`redo` (page 637)). The notification object is the NSUndoManager. This notification does not contain a *`userInfo`* dictionary.

DidUndoChangeNotification

Posted just after an NSUndoManager performs an undo operation. If you invoke `undo` (page 641) or `undoNestedGroup` (page 642), this notification is posted. The notification object is the NSUndoManager. This notification does not contain a *`userInfo`* dictionary.

WillCloseUndoGroupNotification

Posted before an NSUndoManager closes an undo group, which occurs in the implementation of the `endUndoGrouping` (page 635) method. The notification object is the NSUndoManager. This notification does not contain a *`userInfo`* dictionary.

WillRedoChangeNotification

Posted just before an NSUndoManager performs a redo operation (`redo` (page 637)). The notification object is the NSUndoManager. This notification does not contain a *`userInfo`* dictionary.

WillUndoChangeNotification

Posted just before an NSUndoManager performs an undo operation. If you invoke `undo` (page 641) or `undoNestedGroup` (page 642), this notification is posted. The notification object is the NSUndoManager. This notification does not contain a *`userInfo`* dictionary.

NSUniqueIDSpecifier

Inherits from	NSScriptObjectSpecifier : NSObject
Package:	com.apple.cocoa.foundation
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 unique ID. This specifier works only for objects that have an ID property. The unique ID object passed into NSUniqueIDSpecifiers must be either a number object, such as an Integer, or a String. 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 NSUniqueIDSpecifiers 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` method, the method is invoked. This method can potentially be very fast, but it is relatively difficult to implement.

3. An NSWhoseSpecifier that specifies the first object whose relevant `'ID '` attribute matches the ID is synthesized and evaluated. The NSWhoseSpecifier must search through all of the keyed elements in the container, looking for a match. The search is potentially very slow.

Tasks

Constructors

`NSUniqueIDSpecifier` (page 646)
Creates an NSUniqueIDSpecifier with no data.

Accessing Unique ID Information

`setUniqueID` (page 647)
Sets the ID encapsulated by the receiver.

`uniqueID` (page 647)
Returns the ID encapsulated by the receiver.

Constructors

NSUniqueIDSpecifier

Creates an NSUniqueIDSpecifier with no data.

```
public NSUniqueIDSpecifier()
```

Discussion
Do not use this constructor.

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

Returns a newly created unidentified NSUniqueIDSpecifier with container specifier *container* and key *property*.

```
public NSUniqueIDSpecifier(NSScriptObjectSpecifier container, String property)
```

Discussion
The class description of container is set automatically. Use `setUniqueID` (page 647) to assign an ID to the returned object.

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

Creates an unidentified NSUniqueIDSpecifier initialized with container specifier *container*, key *property*, and the class description of the object specifier *classDescription*, derived from the value of the specifier's key.

```
public NSUniqueIDSpecifier(NSScriptClassDescription classDescription,
    NSScriptObjectSpecifier container, String property)
```

Discussion
The receiver's child specifier reference is set to `null`. Use `setUniqueID` (page 647) to assign a name to the returned object.

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

Creates an NSUniqueIDSpecifier with the ID *uniqueID* initialized with container specifier *container*, key *property*, and the class description of the object specifier *classDescription*, derived from the value of the specifier's key.

```
public NSUniqueIDSpecifier(NSScriptClassDescription classDescription,
    NSScriptObjectSpecifier container, String property, String uniqueID)
```

Discussion
The receiver's child specifier reference is set to `null`.

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

Instance Methods

setUniqueID

Sets the ID encapsulated by the receiver.

```
public void setUniqueID(Object uniqueID)
```

Discussion
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 '`. 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 647)

uniqueID

Returns the ID encapsulated by the receiver.

```
public Object uniqueID()
```

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

See Also
`setUniqueID` (page 647)

NSUserDefaults

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Companion guide	User Defaults Programming Topics for Cocoa

Class at a Glance

The NSUserDefaults class provides a programmatic interface for interacting with the defaults system.

Principal Attributes

- A dictionary of defaults

Commonly Used Methods

`objectForKey` (page 658)
: Returns the default value for the specified key.

`setObjectForKey` (page 662)
: Sets the default value for the specified key.

`removeObjectForKey` (page 660)
: Removes the default entry identified by the specified key.

`registerDefaults` (page 660)
: Adds the specified defaults to the `RegistrationDomain`—a cache of application-provided defaults that are used unless a user overrides them.

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.

A defaults database is created automatically for each user.

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 664) method, which is automatically invoked at periodic intervals, keeps the in-memory cache in sync with a user's defaults database.

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.

A default's value can be only property list objects: NSData, string, number, NSDate, NSArray, or NSDictionary.

Warning: User defaults are not thread safe.

Tasks

Constructors

`NSUserDefaults` (page 653)
Creates an NSUserDefaults for the current user account and with the argument and registration domains set up.

Getting the Shared Instance

`standardUserDefaults` (page 653)
Returns the shared defaults object.

`resetStandardUserDefaults` (page 653)
Synchronizes any changes made to the shared user defaults object and releases it from memory.

Getting a Default

`arrayForKey` (page 654)
Invokes `objectForKey` (page 658) with key *`defaultName`*.

`booleanForKey` (page 655)
Invokes `stringForKey` (page 664) with key *`defaultName`*.

`dataForKey` (page 655)
Invokes `objectForKey` (page 658) with key *`defaultName`*.

`dictionaryForKey` (page 655)
Invokes `objectForKey` (page 658) with key *`defaultName`*.

`doubleForKey` (page 656)
Invokes `stringForKey` (page 664) with key *`defaultName`*.

floatForKey (page 656)
Invokes stringForKey (page 664) with key *defaultName*.

integerForKey (page 657)
Invokes stringForKey (page 664) with key *defaultName*.

longForKey (page 657)
Invokes stringForKey (page 664) with key *defaultName*.

objectForKey (page 658)
Returns the value of the first occurrence of the default identified by *defaultName*, searching the domains included in the search list in the order they're listed.

objectForKeyInDomain (page 658)
Returns the value for *defaultName* in the domain *domainName*.

stringForKey (page 664)
Invokes objectForKey (page 658) with key *defaultName*.

Setting and Removing Defaults

removeObjectForKey (page 660)
Removes the value for the default identified by *defaultName* in the standard application domain.

removeObjectForKeyInDomain (page 660)
Deletes *defaultName* from the domain *domainName* for the current user on any host.

setBooleanForKey (page 661)
Sets the value of the default identified by *defaultName* to a string representation of `true` or `false`, depending on *value*.

setDoubleForKey (page 662)
Sets the value of the default identified by *defaultName* to a string representation of *value*.

setFloatForKey (page 662)
Sets the value of the default identified by *defaultName* to a string representation of *value*.

setIntegerForKey (page 662)
Sets the value of the default identified by *defaultName* to a string representation of *value*.

setLongForKey (page 662)
Sets the value of the default identified by *defaultName* to a string representation of *value*.

setObjectForKey (page 662)
Sets the value of the default identified by *defaultName* in the standard application domain.

setObjectForKeyInDomain (page 663)
Sets *value* as the value for *defaultName* in the domain *domainName* for the current user on any host.

Accessing Managed Environment Keys

objectIsForcedForKey (page 658)
Determines whether *key* is managed by an administrator.

objectIsForcedForKeyInDomain (page 659)
Determines whether *key* is managed by an administrator.

Setting and Getting the Search List

`setSearchList` (page 663)
This method has been deprecated.

`searchList` (page 661)
This method has been deprecated.

`dictionaryRepresentation` (page 656)
Returns a dictionary that contains a union of all key-value pairs in the domains in the search list.

Maintaining Persistent Domains

`persistentDomainForName` (page 659)
Returns a dictionary representing the persistent domain identified by *domainName*.

`persistentDomainNames` (page 659)
Returns an array containing the names of the current persistent domains.

`removePersistentDomainForName` (page 660)
Removes the persistent domain identified by *domainName* from the user's defaults.

`setPersistentDomainForName` (page 663)
Sets the dictionary to *domain* for the persistent domain named *domainName*.

`synchronize` (page 664)
Saves any modifications to the persistent domains and updates all persistent domains that were not modified to what is on disk.

Maintaining Volatile Domains

`removeVolatileDomainForName` (page 661)
Removes the volatile domain identified by *domainName* from the user's defaults.

`setVolatileDomainForName` (page 664)
Sets the dictionary for the volatile domain named *domainName* to *domain*.

`volatileDomainForName` (page 665)
Returns a dictionary representing the volatile domain identified by *domainName*.

`volatileDomainNames` (page 665)
Returns an array containing the names of the current volatile domains.

Registering Defaults

`registerDefaults` (page 660)
Adds the contents of *dictionary* to the registration domain.

Maintaining Suites

`addSuiteNamed` (page 654)
Inserts a new domain, *suiteName*, into the receiver's search list.

`removeSuiteNamed` (page 661)
Removes the *`suiteName`* domain from the receiver's search list.

Constructors

NSUserDefaults

Creates an NSUserDefaults for the current user account and with the argument and registration domains set up.

```
public NSUserDefaults()
```

Discussion
This constructor does not put anything in the search list.

See Also
`standardUserDefaults` (page 653)

Static Methods

resetStandardUserDefaults

Synchronizes any changes made to the shared user defaults object and releases it from memory.

```
public static void resetStandardUserDefaults()
```

Discussion
A subsequent invocation of `standardUserDefaults` (page 653) creates a new shared user defaults object with the standard search list.

standardUserDefaults

Returns the shared defaults object.

```
public static NSUserDefaults standardUserDefaults()
```

Discussion
If it does not exist yet, it is created with a search list containing the names of the following domains, in this order:

- `ArgumentDomain`, consisting of defaults parsed from the application's arguments
- A domain identified by the application's bundle identifier
- `GlobalDomain`, consisting of defaults meant to be seen by all applications
- Separate domains for each of the user's preferred languages

- `RegistrationDomain`, 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.

Instance Methods

addSuiteNamed

Inserts a new domain, *suiteName*, into the receiver's search list.

```
public void addSuiteNamed(String suiteName)
```

Discussion
The suite domain is inserted after the application domain.

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.

See Also
`standardUserDefaults` (page 653)
`removeSuiteNamed` (page 661)

arrayForKey

Invokes `objectForKey` (page 658) with key *defaultName*.

```
public NSArray arrayForKey(String defaultName)
```

Discussion
Returns the value associated with *defaultName* if it's an NSArray object and `null` otherwise.

See Also
`booleanForKey` (page 655)
`dataForKey` (page 655)
`dictionaryForKey` (page 655)
`doubleForKey` (page 656)
`floatForKey` (page 656)
`integerForKey` (page 657)
`longForKey` (page 657)
`objectForKey` (page 658)
`stringForKey` (page 664)

booleanForKey

Invokes `stringForKey` (page 664) with key *defaultName*.

```
public boolean booleanForKey(String defaultName)
```

Discussion
Returns `true` if the value associated with *defaultName* is an String containing the word "yes" in uppercase or lowercase or responds to the `intValue` message by returning a nonzero value. Otherwise, returns `false`.

See Also
`arrayForKey` (page 654)
`dataForKey` (page 655)
`dictionaryForKey` (page 655)
`doubleForKey` (page 656)
`floatForKey` (page 656)
`integerForKey` (page 657)
`longForKey` (page 657)
`objectForKey` (page 658)
`stringForKey` (page 664)

dataForKey

Invokes `objectForKey` (page 658) with key *defaultName*.

```
public NSData dataForKey(String defaultName)
```

Discussion
Returns the corresponding value if it's an NSData object and `null` otherwise.

See Also
`arrayForKey` (page 654)
`booleanForKey` (page 655)
`dictionaryForKey` (page 655)
`doubleForKey` (page 656)
`floatForKey` (page 656)
`integerForKey` (page 657)
`longForKey` (page 657)
`objectForKey` (page 658)
`stringForKey` (page 664)

dictionaryForKey

Invokes `objectForKey` (page 658) with key *defaultName*.

```
public NSDictionary dictionaryForKey(String defaultName)
```

Discussion
Returns the corresponding value if it's an NSDictionary object and `null` otherwise.

See Also
arrayForKey (page 654)
booleanForKey (page 655)
dataForKey (page 655)
doubleForKey (page 656)
floatForKey (page 656)
integerForKey (page 657)
longForKey (page 657)
objectForKey (page 658)
stringForKey (page 664)

dictionaryRepresentation

Returns a dictionary that contains a union of all key-value pairs in the domains in the search list.

```
public NSDictionary dictionaryRepresentation()
```

Discussion
As with objectForKey (page 658), key-value pairs in domains that are earlier in the search list take precedence. The combined result doesn't preserve information about which domain each entry came from.

See Also
searchList (page 661)

doubleForKey

Invokes stringForKey (page 664) with key *defaultName*.

```
public double doubleForKey(String defaultName)
```

Discussion
Returns 0 if no string is returned. Otherwise, the resulting string is sent a doubleValue message, which provides this method's return value.

See Also
arrayForKey (page 654)
booleanForKey (page 655)
dataForKey (page 655)
dictionaryForKey (page 655)
integerForKey (page 657)
floatForKey (page 656)
longForKey (page 657)
objectForKey (page 658)
stringForKey (page 664)

floatForKey

Invokes stringForKey (page 664) with key *defaultName*.

```
public float floatForKey(String defaultName)
```

Discussion
Returns 0 if no string is returned. Otherwise, the resulting string is sent a `floatValue` message, which provides this method's return value.

See Also
`arrayForKey` (page 654)
`booleanForKey` (page 655)
`dataForKey` (page 655)
`dictionaryForKey` (page 655)
`doubleForKey` (page 656)
`integerForKey` (page 657)
`longForKey` (page 657)
`objectForKey` (page 658)
`stringForKey` (page 664)

integerForKey

Invokes `stringForKey` (page 664) with key *defaultName*.

```
public int integerForKey(String defaultName)
```

Discussion
Returns 0 if no string is returned. Otherwise, the resulting string is sent an `intValue` message, which provides this method's return value.

See Also
`arrayForKey` (page 654)
`booleanForKey` (page 655)
`dataForKey` (page 655)
`doubleForKey` (page 656)
`dictionaryForKey` (page 655)
`floatForKey` (page 656)
`longForKey` (page 657)
`objectForKey` (page 658)
`stringForKey` (page 664)

longForKey

Invokes `stringForKey` (page 664) with key *defaultName*.

```
public long longForKey(String defaultName)
```

Discussion
Returns 0 if no string is returned. Otherwise, the resulting string is sent a `longValue` message, which provides this method's return value.

See Also
`arrayForKey` (page 654)

booleanForKey (page 655)
dataForKey (page 655)
doubleForKey (page 656)
dictionaryForKey (page 655)
floatForKey (page 656)
integerForKey (page 657)
objectForKey (page 658)
stringForKey (page 664)

objectForKey

Returns the value of the first occurrence of the default identified by *defaultName*, searching the domains included in the search list in the order they're listed.

```
public Object objectForKey(String defaultName)
```

Discussion
Returns `null` if the default isn't found.

See Also
arrayForKey (page 654)
booleanForKey (page 655)
dataForKey (page 655)
dictionaryForKey (page 655)
doubleForKey (page 656)
floatForKey (page 656)
longForKey (page 657)
stringForKey (page 664)

objectForKeyInDomain

Returns the value for *defaultName* in the domain *domainName*.

```
public Object objectForKeyInDomain(String defaultName, String domainName)
```

Discussion
Only the defaults defined in the *domainName* for the current user for any host are searched. If *defaultName* is not found, `null` is returned.

See Also
removeObjectForKeyInDomain (page 660)
setObjectForKeyInDomain (page 663)

objectIsForcedForKey

Determines whether *key* is managed by an administrator.

```
public boolean objectIsForcedForKey(String key)
```

Discussion
Returns `true` if the value of *key* is managed by an administrator; otherwise returns `false`. This method assumes that *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
`objectIsForcedForKeyInDomain` (page 659)

objectIsForcedForKeyInDomain

Determines whether *key* is managed by an administrator.

```
public boolean objectIsForcedForKeyInDomain(String key, String domain)
```

Discussion
Returns `true` if the value of *key* in the specified *domain* is managed by an administrator; otherwise returns `false`. This method assumes that *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 658)

persistentDomainForName

Returns a dictionary representing the persistent domain identified by *domainName*.

```
public NSDictionary persistentDomainForName(String domainName)
```

Discussion
The keys in the dictionary are names of defaults, and the value corresponding to each key is a property list object (NSData, String, Number, NSDate, NSArray, or NSDictionary). *domainName* should be your application's bundle identifier.

See Also
`removePersistentDomainForName` (page 660)
`setPersistentDomainForName` (page 663)

persistentDomainNames

Returns an array containing the names of the current persistent domains.

```
public NSArray persistentDomainNames()
```

Discussion
You can get each domain by using the domain names in the array as arguments to `persistentDomainForName` (page 659).

See Also
`removePersistentDomainForName` (page 660)
`setPersistentDomainForName` (page 663)

registerDefaults

Adds the contents of *dictionary* to the registration domain.

```
public void registerDefaults(NSDictionary dictionary)
```

Discussion
If there is no registration domain, it's created using *dictionary*, and `RegistrationDomain` is added to the end of the search list.

removeObjectForKey

Removes the value for the default identified by *defaultName* in the standard application domain.

```
public void removeObjectForKey(String defaultName)
```

Discussion
Removing a default has no effect on the value returned by the `objectForKey` (page 658) method if the same key exists in a domain that precedes the standard application domain in the search list.

See Also
`setObjectForKey` (page 662)

removeObjectForKeyInDomain

Deletes *defaultName* from the domain *domainName* for the current user on any host.

```
public void removeObjectForKeyInDomain(String defaultName, String domainName)
```

Discussion
If *domainName* is part of the receiver's search list, posts an `UserDefaultsDidChangeNotification` (page 667).

See Also
`objectForKeyInDomain` (page 658)
`setObjectForKeyInDomain` (page 663)

removePersistentDomainForName

Removes the persistent domain identified by *domainName* from the user's defaults.

```
public void removePersistentDomainForName(String domainName)
```

Discussion
The first time a persistent domain is changed after `synchronize` (page 664), a `UserDefaultsDidChangeNotification` (page 667) is posted. *domainName* should be your application's bundle identifier.

See Also
`setPersistentDomainForName` (page 663)

removeSuiteNamed

Removes the *suiteName* domain from the receiver's search list.

```
public void removeSuiteNamed(String suiteName)
```

See Also
`addSuiteNamed` (page 654)

removeVolatileDomainForName

Removes the volatile domain identified by *domainName* from the user's defaults.

```
public void removeVolatileDomainForName(String domainName)
```

See Also
`setVolatileDomainForName` (page 664)

searchList

This method has been deprecated.

```
public NSArray searchList()
```

See Also
`setSearchList` (page 663)
`dictionaryRepresentation` (page 656)

setBooleanForKey

Sets the value of the default identified by *defaultName* to a string representation of `true` or `false`, depending on *value*.

```
public void setBooleanForKey(boolean value, String defaultName)
```

Discussion
Invokes `setObjectForKey` (page 662) as part of its implementation.

See Also
`booleanForKey` (page 655)

setDoubleForKey

Sets the value of the default identified by *defaultName* to a string representation of *value*.

```
public void setDoubleForKey(double value, String defaultName)
```

Discussion
Invokes setObjectForKey (page 662) as part of its implementation.

See Also
doubleForKey (page 656)

setFloatForKey

Sets the value of the default identified by *defaultName* to a string representation of *value*.

```
public void setFloatForKey(float value, String defaultName)
```

Discussion
Invokes setObjectForKey (page 662) as part of its implementation.

See Also
floatForKey (page 656)

setIntegerForKey

Sets the value of the default identified by *defaultName* to a string representation of *value*.

```
public void setIntegerForKey(int value, String defaultName)
```

Discussion
Invokes setObjectForKey (page 662) as part of its implementation.

See Also
integerForKey (page 657)

setLongForKey

Sets the value of the default identified by *defaultName* to a string representation of *value*.

```
public void setLongForKey(long value, String defaultName)
```

Discussion
Invokes setObjectForKey (page 662) as part of its implementation.

See Also
longForKey (page 657)

setObjectForKey

Sets the value of the default identified by *defaultName* in the standard application domain.

```
public void setObjectForKey(Object value, String defaultName)
```

Discussion
Setting a default has no effect on the value returned by the `objectForKey` (page 658) method if the same key exists in a domain that precedes the application domain in the search list.

See Also
`removeObjectForKey` (page 660)

setObjectForKeyInDomain

Sets *value* as the value for *defaultName* in the domain *domainName* for the current user on any host.

```
public void setObjectForKeyInDomain(Object value, String defaultName, String
    domainName)
```

Discussion
If *domainName* is part of the receiver's search list, it posts an `UserDefaultsDidChangeNotification` (page 667).

See Also
`objectForKeyInDomain` (page 658)
`removeObjectForKeyInDomain` (page 660)

setPersistentDomainForName

Sets the dictionary to *domain* for the persistent domain named *domainName*.

```
public void setPersistentDomainForName(NSDictionary domain, String domainName)
```

Discussion
The first time a persistent domain is changed after `synchronize` (page 664), a `UserDefaultsDidChangeNotification` (page 667) is posted. *domainName* should be your application's bundle identifier.

See Also
`persistentDomainForName` (page 659)
`persistentDomainNames` (page 659)

setSearchList

This method has been deprecated.

```
public void setSearchList(NSArray array)
```

See Also
`searchList` (page 661)

setVolatileDomainForName

Sets the dictionary for the volatile domain named *domainName* to *domain*.

```
public void setVolatileDomainForName(NSDictionary domain, String domainName)
```

Discussion
This method throws an `InvalidArgumentException` if a volatile domain with *domainName* already exists.

See Also
`volatileDomainForName` (page 665)
`volatileDomainNames` (page 665)

stringForKey

Invokes `objectForKey` (page 658) with key *defaultName*.

```
public String stringForKey(String defaultName)
```

Discussion
Returns the corresponding value if it's a String object and `null` otherwise.

See Also
`arrayForKey` (page 654)
`booleanForKey` (page 655)
`dataForKey` (page 655)
`dictionaryForKey` (page 655)
`doubleForKey` (page 656)
`floatForKey` (page 656)
`integerForKey` (page 657)
`longForKey` (page 657)
`objectForKey` (page 658)

synchronize

Saves any modifications to the persistent domains and updates all persistent domains that were not modified to what is on disk.

```
public boolean synchronize()
```

Discussion
Returns `false` if it could not save data to disk. Because `synchronize` 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 user defaults to what is on disk even though you have not made any changes.

See Also
`persistentDomainForName` (page 659)
`persistentDomainNames` (page 659)
`removePersistentDomainForName` (page 660)
`setPersistentDomainForName` (page 663)

volatileDomainForName

Returns a dictionary representing the volatile domain identified by *domainName*.

```
public NSDictionary volatileDomainForName(String domainName)
```

Discussion
The keys in the dictionary are names of defaults, and the value corresponding to each key is a property list object (NSData, String, Number, NSDate, NSArray, or NSDictionary).

See Also
`removeVolatileDomainForName` (page 661)
`setVolatileDomainForName` (page 664)

volatileDomainNames

Returns an array containing the names of the current volatile domains.

```
public NSArray volatileDomainNames()
```

Discussion
You can get each domain by using the domain names in the array as arguments to `volatileDomainForName` (page 665).

See Also
`removeVolatileDomainForName` (page 661)
`setVolatileDomainForName` (page 664)

Constants

NSUserDefaults 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".

Constant	Description
`AMPMDesignation`	An array of strings that specify how the morning and afternoon designations are printed. The defaults are AM and PM.
`CurrencySymbol`	A string that specifies the symbol used to denote currency in this language. The default is "$".
`DateFormatString`	A format string that specifies how dates are printed using the date format specifiers. The default is to use weekday names with full month names and full years, as in "Saturday, March 24, 2001."

Constant	Description
`DateTimeOrdering`	A 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."
`DecimalDigits`	Strings that identify the decimal digits in addition to or instead of the ASCII digits.
`DecimalSeparator`	A string that specifies the decimal separator. The decimal separator separates the ones place from the tenths place. The default is ".".
`EarlierTimeDesignations`	An array of strings that denote a time in the past. These are adjectives that modify values from `YearMonthWeekDesignations`. The defaults are "prior," "last," "past," and "ago."
`HourNameDesignations`	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).
`International-CurrencyString`	A string containing a three-letter abbreviation for currency, following the ISO 4217 standard.
`LaterTimeDesignations`	An array of strings that denotes a time in the future. This array is an adjective that modifies a value from `YearMonthWeekDesignations`. The default is (next).
`MonthNameArray`	An array that specifies the full names for the months.
`NegativeCurrency-FormatString`	A format string that specifies how negative numbers are printed when representing a currency value. The default is "–$9,999.00".
`NextDayDesignations`	A string that identifies the day after today. The default is (tomorrow).
`NextNextDayDesignations`	A string that identifies the day after tomorrow. The default is (nextday).
`PositiveCurrency-FormatString`	A format string that specifies how positive numbers are printed when representing a currency value. The default is "$9,999.00".
`PriorDayDesignations`	A string that identifies the day before today. The default is (yesterday).
`ShortDateFormatString`	A format string that specifies how dates are abbreviated. The default is to separate the day month and year with slashes and to put the day first, as in 31/10/01.
`ShortMonthNameArray`	An array that specifies the abbreviations for the months.

Constant	Description
`ShortTimeDate-FormatString`	A format string that specifies how times and dates are abbreviated. 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. "
`ShortWeekDayNameArray`	An array that specifies the abbreviations for the days of the week. Sunday should be the first day of the week.
`ThisDayDesignations`	A string that identifies what this day is called. The default is (today, now).
`ThousandsSeparator`	A string that specifies the separator character for the thousands place of a decimal number. The default is a comma.
`TimeDateFormatString`	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."
`TimeFormatString`	A format string that specifies how dates with times are printed. The default is to use a 12-hour clock.
`WeekDayNameArray`	An array that gives the names for the days of the week. Sunday should be the first day of the week.
`YearMonthWeek-Designations`	An array of strings that specifies the words for year, month, and week in the current locale. The defaults are "year," "month," and "week."

Notifications

UserDefaultsDidChangeNotification

This notification is posted the first time after a synchronize when a change is made to defaults in a persistent domain.

The notification object is the NSUserDefaults instance. This notification does not contain a *`userInfo`* dictionary.

NSValueTransformer

Inherits from	NSObject
Package:	com.apple.cocoa.foundation
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Value Transformer Programming Guide

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.

Tasks

Constructors

`NSValueTransformer` (page 670)
Creates and returns an empty NSValueTransformer.

Using Name-based Registry

`setValueTransformerForName` (page 670)
Registers the value transformer specified by *`transformer`* as handling transformations with the identifier *`name`*.

`valueTransformerForName` (page 671)
Returns the value transformer identified by *`name`* in the shared registry or `null` if not found.

`valueTransformerNames` (page 671)
Returns an array of all the registered value transformers.

Getting Information About a Transformer

`allowsReverseTransformation` (page 670)
Returns a Boolean value that indicates whether the value transformer can reverse a transformation.

`transformedValueClass` (page 671)
Returns the class of the value returned by the transformer.

Using Transformers

`transformedValue` (page 672)
Returns the result of transforming *value*.

`reverseTransformedValue` (page 671)
Returns the result of the reverse transformation of *value*.

Constructors

NSValueTransformer

Creates and returns an empty NSValueTransformer.

```
public NSValueTransformer()
```

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

Static Methods

allowsReverseTransformation

Returns a Boolean value that indicates whether the value transformer can reverse a transformation.

```
public static boolean allowsReverseTransformation()
```

Discussion
Subclasses should override this method and return `true` if it supports reverse value transformations.

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

setValueTransformerForName

Registers the value transformer specified by *transformer* as handling transformations with the identifier *name*.

```
public static void setValueTransformerForName(NSValueTransformer transformer, String
    name)
```

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

See Also
valueTransformerForName (page 671)

transformedValueClass

Returns the class of the value returned by the transformer.

```
public static Class transformedValueClass()
```

Discussion
Subclasses should override this method and return the appropriate class.

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

valueTransformerForName

Returns the value transformer identified by *name* in the shared registry or `null` if not found.

```
public static NSValueTransformer valueTransformerForName(String name)
```

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

See Also
setValueTransformerForName (page 670)

valueTransformerNames

Returns an array of all the registered value transformers.

```
public static NSArray valueTransformerNames()
```

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

Instance Methods

reverseTransformedValue

Returns the result of the reverse transformation of *value*.

```
public Object reverseTransformedValue(Object value)
```

Discussion
The default implementation throws an exception if `allowsReverseTransformation` (page 670) returns `false`; otherwise it will invoke `transformedValue` (page 672) with *value*.

Subclasses should override this method if they require more complex reverse transformations than, for example, negation. If your value transformer converts a nonlocalized string to a localized string, your implementation of this method might return a nonlocalized version of the string passed to it.

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

See Also
`transformedValue` (page 672)

transformedValue

Returns the result of transforming *value*.

```
public Object transformedValue(Object value)
```

Discussion
Override this method in your implementation to transform and return an object based on *value*. The default implementation simply returns *value*.

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

See Also
`reverseTransformedValue` (page 671)

Constants

The following named value transformers are defined by NSValueTransformer:

Constant	Description
`NegateBoolean-TransformerName`	This value transformer negates a boolean value, transforming `true` to `false` and `false` to `true`. This transformer is reversible.
`IsNilTransformerName`	This value transformer returns `true` if the value is `null`. This transformer is not reversible.
`IsNotNilTransformerName`	This value transformer returns `true` if the value is non-`null`.This transformer is not reversible.

Constant	Description
`UnarchiveFromData-TransformerName`	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.

NSWhoseSpecifier

Inherits from	NSScriptObjectSpecifier : NSObject
Package:	com.apple.cocoa.foundation
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 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 558). See "Boolean Expressions and Logical Operations" (page 541) in NSScriptObjectSpecifier and the descriptions in NSComparisonMethods (page 685) and NSScriptingComparisonMethods (page 691) for more information.

The set of elements specified by an NSWhoseSpecifier can be a subset of those that pass the NSWhoseSpecifier's test. This subset is specified by the various subelement properties of the NSWhoseSpecifier. Consider as an example the specifier `paragraphs where color of third word is blue`. This would be represented by an NSWhoseSpecifier 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`).

NSScriptWhoseSpecifier is part of Cocoa's built-in script handling. You don't normally subclass it.

Tasks

Constructors

`NSWhoseSpecifier` (page 676)
Returns an NSWhoseSpecifier with no data.

Accessing Information About a Whose Specifier

`endSubelementIdentifier` (page 677)
Returns the end subelement identifier for the specifier, or `NoSubelement` if there is none.

`endSubelementIndex` (page 677)
Returns the index position of the last subelement within the range of objects being tested that passes the specifier's test.

`setEndSubelementIdentifier` (page 677)
Sets the end subelement identifier for the specifier to the value of *`subelement`*.

`setEndSubelementIndex` (page 677)
Sets the index position to *`index`* of the last subelement within the range of objects being tested that pass the specifier's test.

`setStartSubelementIdentifier` (page 677)
Sets the start subelement identifier for the specifier to the value of *`subelement`*.

`setStartSubelementIndex` (page 678)
Sets the index position to *`index`* of the first subelement within the range of objects being tested that passes the specifier's test.

`setTest` (page 678)
Sets the test object to *`test`* that is encapsulated by the receiver.

`startSubelementIdentifier` (page 678)
Returns the start subelement identifier for the specifier.

`startSubelementIndex` (page 678)
Returns the index position of the first subelement within the range of objects being tested that pass the specifier's test.

`test` (page 678)
Returns the test object encapsulated by the receiver.

Constructors

NSWhoseSpecifier

Returns an NSWhoseSpecifier with no data.

```
public NSWhoseSpecifier()
```

Discussion
Do not use this constructor.

Constructs an NSWhoseSpecifier with the class description, container specifier, and key supplied by *`classDescription`*, *`specifier`*, and *`key`*, respectively.

```
public NSWhoseSpecifier(NSScriptClassDescription classDescription,
    NSScriptObjectSpecifier specifier, String key)
```

Constructs an NSWhoseSpecifier with the class description, container specifier, key, and whose test supplied by *`classDescription`*, *`specifier`*, *`key`*, and *`test`*, respectively.

```
public NSWhoseSpecifier(NSScriptClassDescription classDescription,
    NSScriptObjectSpecifier specifier, String key, NSScriptWhoseTest test)
```

Constructs an NSWhoseSpecifier with the class description and key supplied by *specifier* and *key*, respectively.

```
public NSWhoseSpecifier(NSScriptObjectSpecifier specifier, String key)
```

Instance Methods

endSubelementIdentifier

Returns the end subelement identifier for the specifier, or `NoSubelement` if there is none.

```
public int endSubelementIdentifier()
```

Discussion
See "Constants" (page 679) for a list of possible return values.

endSubelementIndex

Returns the index position of the last subelement within the range of objects being tested that passes the specifier's test.

```
public int endSubelementIndex()
```

setEndSubelementIdentifier

Sets the end subelement identifier for the specifier to the value of *subelement*.

```
public void setEndSubelementIdentifier(int subelement)
```

Discussion
See "Constants" (page 679) for a list of possible values for *subelement*.

setEndSubelementIndex

Sets the index position to *index* of the last subelement within the range of objects being tested that pass the specifier's test.

```
public void setEndSubelementIndex(int index)
```

Discussion
Used only if the end subelement identifier is `IndexSubelement`.

setStartSubelementIdentifier

Sets the start subelement identifier for the specifier to the value of *subelement*.

```
public void setStartSubelementIdentifier(int subelement)
```

Discussion
See "Constants" (page 679) for a list of possible values for *subelement*.

setStartSubelementIndex

Sets the index position to *index* of the first subelement within the range of objects being tested that passes the specifier's test.

```
public void setStartSubelementIndex(int index)
```

Discussion
Used only if the start subelement identifier is `IndexSubelement`.

setTest

Sets the test object to *test* that is encapsulated by the receiver.

```
public void setTest(NSScriptWhoseTest test)
```

startSubelementIdentifier

Returns the start subelement identifier for the specifier.

```
public int startSubelementIdentifier()
```

Discussion
See "Constants" (page 679) for a list of possible return values.

startSubelementIndex

Returns the index position of the first subelement within the range of objects being tested that pass the specifier's test.

```
public int startSubelementIndex()
```

test

Returns the test object encapsulated by the receiver.

```
public NSScriptWhoseTest test()
```

Constants

The following constants are defined by NSWhoseSpecifier and are used by `startSubelementIdentifier` (page 678), `setStartSubelementIdentifier` (page 677), `endSubelementIdentifier` (page 677), and `setEndSubelementIdentifier` (page 677). NSWhoseSpecifier uses these constants to specify subelements within the collection of objects being tested that pass the specifier's test.

Constant	Description
`IndexSubelement`	An element at a given index that meets the specifier test.
`EverySubelement`	Every element that meets the specifier test.
`MiddleSubelement`	The middle element that meets the specifier test.
`RandomSubelement`	Any element that meets the specifier test.
`NoSubelement`	No subelement met the specifier test. Valid only for specifying the end subelement.; that is, there is no end, so consider all elements.

Interfaces

NSCoding

Implemented by	Various Cocoa classes
Package:	com.apple.cocoa.foundation
Companion guide	Archives and Serializations Programming Guide for Cocoa

Overview

The NSCoding interface is adopted by classes that can be coded with an NSCoder object. Although the interface technically contains no methods, Java classes must implement the following two methods to be codable:

```
protected ClassName(NSCoder decoder, long token);
protected void encodeWithCoder(NSCoder encoder);
```

The first is a constructor method where `ClassName` is replaced with the name of your class. It is invoked when decoding an object. The method must invoke a similar method of its super class if the super class also implements NSCoding; otherwise, the method needs to invoke another constructor of the super class. The second method is invoked when encoding an object. The method must invoke the some method of its super class if the super class also implements NSCoding.

NSCoding is independent of Java's own serialization protocol, `java.io.Serializable`. Unlike `java.io.Serializable`, NSCoding does not mark classes for automatic coding; instead, each class handles its own encoding and decoding operations with the methods described above.

NSComparisonMethods

(informal protocol)

Package: com.apple.cocoa.foundation

Companion guide Cocoa Scripting Guide

Overview

This interface defines a set of default comparison methods useful for the comparisons in NSSpecifierTest (page 585). However, 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 (page 691).

Tasks

Performing Comparisons

`doesContain` (page 686)
Returns `true` if the receiver contains *`object`*, `false` otherwise.

`isCaseInsensitiveLike` (page 686)
Returns `true` if the receiver is considered to be "like" *`aString`* when the case of characters in the receiver is ignored, `false` otherwise.

`isEqualTo` (page 686)
Returns `true` if the receiver is equal to *`object`*, `false` otherwise.

`isGreaterThan` (page 686)
Returns `true` if the receiver is greater than *`object`*, `false` otherwise.

`isGreaterThanOrEqualTo` (page 686)
Returns `true` if the receiver is greater than or equal to *`object`*, `false` otherwise.

`isLessThan` (page 687)
Returns `true` if the receiver is less than *`object`*, `false` otherwise.

`isLessThanOrEqualTo` (page 687)
Returns `true` if the receiver is less than or equal to *`object`*, `false` otherwise.

`isLike` (page 687)
Returns `true` if the receiver is considered to be "like" *`object`*, `false` otherwise.

`isNotEqualTo` (page 687)
Returns `true` if the receiver is not equal to *`object`*, `false` otherwise.

Instance Methods

doesContain

Returns `true` if the receiver contains *`object`*, `false` otherwise.

```
public abstract boolean doesContain(Object object)
```

Discussion
Currently, `doesContain` messages are never sent to any object from within Cocoa itself.

isCaseInsensitiveLike

Returns `true` if the receiver is considered to be "like" *`aString`* when the case of characters in the receiver is ignored, `false` otherwise.

```
public abstract boolean isCaseInsensitiveLike(String aString)
```

Discussion
Currently, `isCaseInsensitiveLike` messages are never sent to any object from within Cocoa itself.

isEqualTo

Returns `true` if the receiver is equal to *`object`*, `false` otherwise.

```
public abstract boolean isEqualTo(Object object)
```

Discussion
During the evaluation of an NSWhoseSpecifier 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 693) method.

isGreaterThan

Returns `true` if the receiver is greater than *`object`*, `false` otherwise.

```
public abstract boolean isGreaterThan(Object object)
```

Discussion
During the evaluation of an NSWhoseSpecifier that contains a test whose operator is `NSSpecifierTest.GreaterThanComparison`, an `isGreaterThan` message may be sent to each potentially specified object, if the potentially specified object does not implement a `scriptingIsGreaterThan` (page 693) method and the object being tested against does not implement a `scriptingIsLessThanOrEqualTo` (page 693) method.

isGreaterThanOrEqualTo

Returns `true` if the receiver is greater than or equal to *`object`*, `false` otherwise.

```
public abstract boolean isGreaterThanOrEqualTo(Object object)
```

Discussion
During the evaluation of an NSWhoseSpecifier that contains a test whose operator is `NSSpecifierTest.GreaterThanOrEqualToComparison`, an `isGreaterThanOrEqualTo` message may be sent to each potentially specified object, if the potentially specified object does not implement a `scriptingIsGreaterThanOrEqualTo` (page 693) method and the object being tested against does not implement a `scriptingIsLessThan` (page 693) method.

isLessThan

Returns `true` if the receiver is less than *object*, `false` otherwise.

```
public abstract boolean isLessThan(Object object)
```

Discussion
During the evaluation of an NSWhoseSpecifier that contains a test whose operator is `NSSpecifierTest.LessThanComparison`, an `isLessThan` message may be sent to each potentially specified object, if the potentially specified object does not implement a `scriptingIsLessThan` (page 693) method and the object being tested against does not implement a `scriptingIsGreaterThanOrEqualTo` (page 693) method.

isLessThanOrEqualTo

Returns `true` if the receiver is less than or equal to *object*, `false` otherwise.

```
public abstract boolean isLessThanOrEqualTo(Object object)
```

Discussion
During the evaluation of an NSWhoseSpecifier that contains a test whose operator is `NSSpecifierTest.LessThanOrEqualToComparison`, an `isLessThanOrEqualTo` message may be sent to each potentially specified object, if the potentially specified object does not implement a `scriptingIsLessThanOrEqualTo` (page 693) method and the object being tested against does not implement a `scriptingIsGreaterThan` (page 693) method.

isLike

Returns `true` if the receiver is considered to be "like" *object*, `false` otherwise.

```
public abstract boolean isLike(String object)
```

Discussion
Currently, `isLike` messages are never sent to any object from within Cocoa itself.

isNotEqualTo

Returns `true` if the receiver is not equal to *object*, `false` otherwise.

```
public abstract boolean isNotEqualTo(Object object)
```

Discussion
Currently, `isNotEqualTo` messages are never sent to any object from within Cocoa itself.

NSKeyValueCoding

(informal protocol)

Package:	com.apple.cocoa.foundation
Companion guide	Key-Value Coding Programming Guide

Overview

The NSKeyValueCoding interface defines a mechanism in which the properties of an object are accessed 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 method for accessing an object's value is `valueForKey` (page 690), 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 690)
Returns the value for the property identified by *key*.

Setting Values

`takeValueForKey` (page 689)
Sets the value for the property identified by *key* to *value*.

Instance Methods

takeValueForKey

Sets the value for the property identified by *key* to *value*.

```
public abstract void takeValueForKey(Object value, String key)
```

Discussion
The default implementation works as follows:

1. Searches for a public accessor method of the form `set`*`Key`*, invoking it if there is one.
2. If a public accessor method is not found, searches for a private accessor method of the form `_set`*`Key`*, invoking it if there is one.
3. If an accessor method is not found `takeValueForKey` searches for an instance variable based on *`key`* and sets the *`value`* directly. For the *`key`* "lastName", this would be `_lastName` or `lastName`.

valueForKey

Returns the value for the property identified by *`key`*.

```
public abstract Object valueForKey(String key)
```

Discussion
The default implementation works as follows:

1. Searches for a public accessor method based on *`key`*. For example, with a *`key`* of "lastName", `valueForKey` looks for a method named `getLastName` or `lastName`.
2. If a public accessor method is not found, searches for a private accessor method based on *`key`* (a method preceded by an underbar). For example, with a *`key`* of "lastName", `valueForKey` looks for a method named `_getLastName` or `_lastName`.
3. If an accessor method is not found `valueForKey` searches for an instance variable based on *`key`* and returns its value directly. For the *`key`* "lastName", this would be `_lastName` or `lastName`.

Constants

These constants define the available array operators. See "Set and Array Operators" for more information.

NSScriptingComparisonMethods

(informal protocol)

Package: com.apple.cocoa.foundation

Companion guide Cocoa Scripting Guide

Overview

This interface 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 interface 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's scripting support 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 (page 685) (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 NSStringReference 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 (page 685). If none require different comparison methods, you can implement only the methods you need from "NSComparisonMethods."

Tasks

Performing Comparisons

`scriptingBeginsWith` (page 692)
: Returns `true` if, in a scripting comparison, the compared object matches the beginning of *`object`*. A default implementation is provided for NSStringReference and NSAttributedString.

`scriptingContains` (page 692)
: Returns `true` if, in a scripting comparison, the compared object contains *`object`*. A default implementation is provided for NSStringReference and NSAttributedString.

`scriptingEndsWith` (page 692)
Returns `true` if, in a scripting comparison, the compared object matches the end of *`object`*. A default implementation is provided for NSStringReference and NSAttributedString.

`scriptingIsEqualTo` (page 693)
Returns `true` if, in a scripting comparison, the compared object is equal to *`object`*. A default implementation is provided for NSStringReference and NSAttributedString.

`scriptingIsGreaterThan` (page 693)
Returns `true` if, in a scripting comparison, the compared object is greater than *`object`*. A default implementation is provided for NSStringReference and NSAttributedString.

`scriptingIsGreaterThanOrEqualTo` (page 693)
Returns `true` if, in a scripting comparison, the compared object is greater than or equal to *`object`*. A default implementation is provided for NSStringReference and NSAttributedString.

`scriptingIsLessThan` (page 693)
Returns `true` if, in a scripting comparison, the compared object is less than *`object`*. A default implementation is provided for NSStringReference and NSAttributedString.

`scriptingIsLessThanOrEqualTo` (page 693)
Returns `true` if, in a scripting comparison, the compared object is less than or equal to *`object`*. A default implementation is provided for NSStringReference and NSAttributedString.

Instance Methods

scriptingBeginsWith

Returns `true` if, in a scripting comparison, the compared object matches the beginning of *`object`*. A default implementation is provided for NSStringReference and NSAttributedString.

```
public abstract boolean scriptingBeginsWith(Object object)
```

scriptingContains

Returns `true` if, in a scripting comparison, the compared object contains *`object`*. A default implementation is provided for NSStringReference and NSAttributedString.

```
public abstract boolean scriptingContains(Object object)
```

scriptingEndsWith

Returns `true` if, in a scripting comparison, the compared object matches the end of *`object`*. A default implementation is provided for NSStringReference and NSAttributedString.

```
public abstract boolean scriptingEndsWith(Object object)
```

scriptingIsEqualTo

Returns `true` if, in a scripting comparison, the compared object is equal to *`object`*. A default implementation is provided for NSStringReference and NSAttributedString.

```
public abstract boolean scriptingIsEqualTo(Object object)
```

scriptingIsGreaterThan

Returns `true` if, in a scripting comparison, the compared object is greater than *`object`*. A default implementation is provided for NSStringReference and NSAttributedString.

```
public abstract boolean scriptingIsGreaterThan(Object object)
```

scriptingIsGreaterThanOrEqualTo

Returns `true` if, in a scripting comparison, the compared object is greater than or equal to *`object`*. A default implementation is provided for NSStringReference and NSAttributedString.

```
public abstract boolean scriptingIsGreaterThanOrEqualTo(Object object)
```

scriptingIsLessThan

Returns `true` if, in a scripting comparison, the compared object is less than *`object`*. A default implementation is provided for NSStringReference and NSAttributedString.

```
public abstract boolean scriptingIsLessThan(Object object)
```

scriptingIsLessThanOrEqualTo

Returns `true` if, in a scripting comparison, the compared object is less than or equal to *`object`*. A default implementation is provided for NSStringReference and NSAttributedString.

```
public abstract boolean scriptingIsLessThanOrEqualTo(Object object)
```

NSScriptingKeyValueCoding

(informal protocol)

Package:	com.apple.cocoa.foundation
Companion guides	Cocoa Scripting Guide Key-Value Coding Programming Guide

Overview

Cocoa scriptability support takes advantage of key-value coding to get and set information in scriptable objects. The methods in this interface 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.

Tasks

Accessing Values

`coerceValueForKey` (page 696)
Uses type info from the class description and NSScriptCoercionHandler to attempt to convert *value* for *key* to the proper type, if necessary.

`valueAtIndexInPropertyWithKey` (page 697)
Retrieves a single value from a multivalue key.

`valueWithNameInPropertyWithKey` (page 697)
Retrieves a single value from a multi-value key.

`valueWithUniqueIDInPropertyWithKey` (page 697)
Retrieves a single value from a multi-value key.

Inserting Values

`insertValueAtIndexInPropertyWithKey` (page 696)
Inserts a single value in a multivalue key at the specified index.

`insertValueInPropertyWithKey` (page 696)
Inserts a single value in a multivalue key.

Updating Values

`removeValueAtIndexFromPropertyWithKey` (page 696)
Removes a single value at the specified index from a multivalue key.

`replaceValueAtIndexInPropertyWithKey` (page 697)
Replaces a single value in a multivalue key at the specified index.

Instance Methods

coerceValueForKey

Uses type info from the class description and NSScriptCoercionHandler to attempt to convert *value* for *key* to the proper type, if necessary.

```
public abstract Object coerceValueForKey(Object value, String key)
```

Discussion
The method `coerceValueFor<Key>` is used if it exists.

insertValueAtIndexInPropertyWithKey

Inserts a single value in a multivalue key at the specified index.

```
public abstract void insertValueAtIndexInPropertyWithKey(Object value, int index,
    String key)
```

Discussion
The method `insertIn<Key>AtIndex` is invoked if it exists.

insertValueInPropertyWithKey

Inserts a single value in a multivalue key.

```
public abstract void insertValueInPropertyWithKey(Object value, String key)
```

Discussion
The method `insertIn<Key>` is used if it exists. Otherwise, throws an exception. 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.

removeValueAtIndexFromPropertyWithKey

Removes a single value at the specified index from a multivalue key.

```
public abstract void removeValueAtIndexFromPropertyWithKey(int index, String key)
```

Discussion
The method `removeFrom<Key>AtIndex` is invoked if it exists.

replaceValueAtIndexInPropertyWithKey

Replaces a single value in a multivalue key at the specified index.

```
public abstract void replaceValueAtIndexInPropertyWithKey(int index, String key,
    Object value)
```

Discussion
The method `replaceIn<Key>AtIndex` is invoked if it exists.

valueAtIndexInPropertyWithKey

Retrieves a single value from a multivalue key.

```
public abstract Object valueAtIndexInPropertyWithKey(int index, String 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.

valueWithNameInPropertyWithKey

Retrieves a single value from a multi-value key.

```
public abstract Object valueWithNameInPropertyWithKey(String name, String key)
```

Discussion
The method `valueIn<Key>WithName` is used if it exists. Otherwise, an exception is thrown.

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

valueWithUniqueIDInPropertyWithKey

Retrieves a single value from a multi-value key.

```
public abstract Object valueWithUniqueIDInPropertyWithKey(Object uniqueID, String
     key)
```

Discussion
The method `valueIn<Key>WithUniqueID` is invoked if it exists. Otherwise, an exception is thrown. The declared type of *uniqueID* in the constructed method must be Object, String, or one of the scalar types that can be encapsulated by the numeric classes, such as Integer or Double.

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

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z