NSString Class Reference

Cocoa > Data Management

2009-02-04

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

Simultaneously published in the United States and Canada.

Contents

NSString Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSMutableCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	NSPathUtilities.h NSString.h NSURL.h
Companion guides	String Programming Guide for Cocoa Property List Programming Guide
Related sample code	Dicey GLSLShowpiece Quartz Composer WWDC 2005 TextEdit StickiesExample TextEditPlus

Overview

The `NSString` class declares the programmatic interface for an object that manages immutable strings. (An immutable string is a text string that is defined when it is created and subsequently cannot be changed. `NSString` is implemented to represent an array of Unicode characters (in other words, a text string).

The mutable subclass of `NSString` is `NSMutableString`.

The `NSString` class has two primitive methods—`length` (page 67) and `characterAtIndex:` (page 34)—that provide the basis for all other methods in its interface. The `length` (page 67) method returns the total number of Unicode characters in the string. `characterAtIndex:` (page 34) gives access to each character in the string by index, with index values starting at `0`.

`NSString` declares methods for finding and comparing strings. It also declares methods for reading numeric values from strings, for combining strings in various ways, and for converting a string to different forms (such as encoding and case changes).

The Application Kit also uses `NSParagraphStyle` and its subclass `NSMutableParagraphStyle` to encapsulate the paragraph or ruler attributes used by the `NSAttributedString` classes. Additionally, methods to support string drawing are described in NSString Application Kit Additions Reference, found in the Application Kit.

`NSString` is "toll-free bridged" with its Core Foundation counterpart, CFString (see `CFStringRef`). This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSString *` parameter, you can pass a `CFStringRef`, and in a function where you see a `CFStringRef` parameter, you can pass an `NSString` instance (you cast one type to the other to suppress compiler warnings). This also applies to your concrete subclasses of `NSString`. See Interchangeable Data Types for more information on toll-free bridging.

String Objects

`NSString` objects represent character strings in frameworks. Representing strings as objects allows you to use strings wherever you use other objects. It also provides the benefits of encapsulation, so that string objects can use whatever encoding and storage are needed for efficiency while simply appearing as arrays of characters. The cluster's two public classes, `NSString` and `NSMutableString`, declare the programmatic interface for non-editable and editable strings, respectively.

Note: An immutable string is a text string that is defined when it is created and subsequently cannot be changed. An immutable string is implemented as an array of Unicode characters (in other words, a text string). To create and manage an immutable string, use the `NSString` class. To construct and manage a string that can be changed after it has been created, use `NSMutableString`.

The objects you create using `NSString` and `NSMutableString` are referred to as string objects (or, when no confusion will result, merely as strings). The term C string refers to the standard `char *` type. Because of the nature of class clusters, string objects aren't actual instances of the `NSString` or `NSMutableString` classes but of one of their private subclasses. Although a string object's class is private, its interface is public, as declared by these abstract superclasses, `NSString` and `NSMutableString`. The string classes adopt the `NSCopying` and `NSMutableCopying` protocols, making it convenient to convert a string of one type to the other.

Understanding characters

A string object presents itself as an array of Unicode characters (Unicode is a registered trademark of Unicode, Inc.). You can determine how many characters a string object contains with the `length` (page 67) method and can retrieve a specific character with the `characterAtIndex:` (page 34) method. These two "primitive" methods provide basic access to a string object.

Most use of strings, however, is at a higher level, with the strings being treated as single entities: You compare strings against one another, search them for substrings, combine them into new strings, and so on. If you need to access string objects character by character, you must understand the Unicode character encoding, specifically issues related to composed character sequences. For details see *The Unicode Standard, Version 4.0* (The Unicode Consortium, Boston: Addison-Wesley, 2003, ISBN 0-321-18578-1) and the Unicode Consortium web site: http://www.unicode.org/. See also Characters and Grapheme Clusters in *String Programming Guide for Cocoa*.

Interpreting UTF-16-encoded data

When creating an `NSString` object from a UTF-16-encoded string (or a byte stream interpreted as UTF-16), if the byte order is not otherwise specified, `NSString` assumes that the UTF-16 characters are big-endian, unless there is a BOM (byte-order mark), in which case the BOM dictates the byte order. When creating an `NSString` object from an array of Unicode characters, the returned string is always native-endian, since the array always contains Unicode characters in native byte order.

Distributed objects

Over distributed-object connections, mutable string objects are passed by-reference and immutable string objects are passed by-copy.

Subclassing Notes

It is possible to subclass `NSString` (and `NSMutableString`), but doing so requires providing storage facilities for the string (which is not inherited by subclasses) and implementing two primitive methods. The abstract `NSString` and `NSMutableString` classes are the public interface of a class cluster consisting mostly of private, concrete classes that create and return a string object appropriate for a given situation. Making your own concrete subclass of this cluster imposes certain requirements (discussed in "Methods to Override" (page 11)).

Make sure your reasons for subclassing `NSString` are valid. Instances of your subclass should represent a string and not something else. Thus the only attributes the subclass should have are the length of the character buffer it's managing and access to individual characters in the buffer. Valid reasons for making a subclass of `NSString` include providing a different backing store (perhaps for better performance) or implementing some aspect of object behavior differently, such as memory management. If your purpose is to add non-essential attributes or metadata to your subclass of `NSString`, a better alternative would be object composition (see "Alternatives to Subclassing" (page 12)). 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 67) and `characterAtIndex:` (page 34). These methods must operate on the backing store that you provide for the characters of the string. For this backing store you can use a static array, a dynamically allocated buffer, a standard `NSString` object, or some other data type or mechanism. You may also choose to override, partially or fully, any other `NSString` method for which you want to provide an alternative implementation. For example, for better performance it is recommended that you override `getCharacters:range:` (page 48) and give it a faster implementation.

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

Note that you shouldn't override the `hash` (page 52) method.

Alternatives to Subclassing

Often a better and easier alternative to making a subclass of `NSString`—or of any other abstract, public class of a class cluster, for that matter—is object composition. This is especially the case when your intent is to add to the subclass metadata or some other attribute that is not essential to a string object. In object composition, you would have an `NSString` object as one instance variable of your custom class (typically a subclass of `NSObject`) and one or more instance variables that store the metadata that you want for the custom object. Then just design your subclass interface to include accessor methods for the embedded string object and the metadata.

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

Adopted Protocols

NSCoding
- `encodeWithCoder:`
- `initWithCoder:`

NSCopying
- `copyWithZone:`

NSMutableCopying
- `mutableCopyWithZone:`

Tasks

Creating and Initializing Strings

`+ string` (page 25)
Returns an empty string.

`- init` (page 53)
Returns an initialized `NSString` object that contains no characters.

`- initWithBytes:length:encoding:` (page 54)
Returns an initialized `NSString` object containing a given number of bytes from a given C array of bytes in a given encoding.

`- initWithBytesNoCopy:length:encoding:freeWhenDone:` (page 54)
Returns an initialized `NSString` object that contains a given number of bytes from a given C array of bytes in a given encoding, and optionally frees the array on deallocation.

`- initWithCharacters:length:` (page 55)
Returns an initialized `NSString` object that contains a given number of characters from a given C array of Unicode characters.

- `initWithCharactersNoCopy:length:freeWhenDone:` (page 55)
  Returns an initialized `NSString` object that contains a given number of characters from a given C array of Unicode characters.
- `initWithString:` (page 63)
  Returns an `NSString` object initialized by copying the characters from another given string.
- `initWithCString:encoding:` (page 58)
  Returns an `NSString` object initialized using the characters in a given C array, interpreted according to a given encoding.
- `initWithUTF8String:` (page 63)
  Returns an `NSString` object initialized by copying the characters a given C array of UTF8-encoded bytes.
- `initWithFormat:` (page 60)
  Returns an *`NSString`* object initialized by using a given format string as a template into which the remaining argument values are substituted.
- `initWithFormat:arguments:` (page 60)
  Returns an *`NSString`* object initialized by using a given format string as a template into which the remaining argument values are substituted according to the user's default locale.
- `initWithFormat:locale:` (page 61)
  Returns an *`NSString`* object initialized by using a given format string as a template into which the remaining argument values are substituted according to given locale information.
- `initWithFormat:locale:arguments:` (page 62)
  Returns an *`NSString`* object initialized by using a given format string as a template into which the remaining argument values are substituted according to given locale information.
- `initWithData:encoding:` (page 59)
  Returns an `NSString` object initialized by converting given data into Unicode characters using a given encoding.
+ `stringWithFormat:` (page 30)
  Returns a string created by using a given format string as a template into which the remaining argument values are substituted.
+ `localizedStringWithFormat:` (page 24)
  Returns a string created by using a given format string as a template into which the remaining argument values are substituted according to the user's default locale.
+ `stringWithCharacters:length:` (page 26)
  Returns a string containing a given number of characters taken from a given C array of Unicode characters.
+ `stringWithString:` (page 30)
  Returns a string created by copying the characters from another given string.
+ `stringWithCString:encoding:` (page 29)
  Returns a string containing the bytes in a given C array, interpreted according to a given encoding.
+ `stringWithUTF8String:` (page 31)
  Returns a string created by copying the data from a given C array of UTF8-encoded bytes.
+ `stringWithCString:` (page 110) Deprecated in Mac OS X v10.4
  Creates a new string using a given C-string. (Deprecated. Use `stringWithCString:encoding:` (page 29) instead.)

+ `stringWithCString:length:` (page 111) Deprecated in Mac OS X v10.4
  Returns a string containing the characters in a given C-string. (Deprecated. Use `stringWithCString:encoding:` (page 29) instead.)

- `initWithCString:` (page 115) Deprecated in Mac OS X v10.4
  Initializes the receiver, a newly allocated `NSString` object, by converting the data in a given C-string from the default C-string encoding into the Unicode character encoding. (Deprecated. Use `initWithCString:encoding:` (page 58) instead.)

- `initWithCString:length:` (page 116) Deprecated in Mac OS X v10.4
  Initializes the receiver, a newly allocated `NSString` object, by converting the data in a given C-string from the default C-string encoding into the Unicode character encoding. (Deprecated. Use initWithCString:encoding: (page 58) instead.)

- `initWithCStringNoCopy:length:freeWhenDone:` (page 116) Deprecated in Mac OS X v10.4
  Initializes the receiver, a newly allocated `NSString` object, by converting the data in a given C-string from the default C-string encoding into the Unicode character encoding. (Deprecated. Use `initWithBytesNoCopy:length:encoding:freeWhenDone:` (page 54) instead.)

Creating and Initializing a String from a File

+ `stringWithContentsOfFile:encoding:error:` (page 26)
  Returns a string created by reading data from the file at a given path interpreted using a given encoding.

- `initWithContentsOfFile:encoding:error:` (page 56)
  Returns an `NSString` object initialized by reading data from the file at a given path using a given encoding.

+ `stringWithContentsOfFile:usedEncoding:error:` (page 27)
  Returns a string created by reading data from the file at a given path and returns by reference the encoding used to interpret the file.

- `initWithContentsOfFile:usedEncoding:error:` (page 57)
  Returns an `NSString` object initialized by reading data from the file at a given path and returns by reference the encoding used to interpret the characters.

+ `stringWithContentsOfFile:` (page 109) Deprecated in Mac OS X v10.4
  Returns a string created by reading data from the file named by a given path. (Deprecated. Use `stringWithContentsOfFile:encoding:error:` (page 26) or stringWithContentsOfFile:usedEncoding:error: (page 27) instead.)

- `initWithContentsOfFile:` (page 114) Deprecated in Mac OS X v10.4
  Initializes the receiver, a newly allocated `NSString` object, by reading data from the file named by *`path`*. (Deprecated. Use `initWithContentsOfFile:encoding:error:` (page 56) or `initWithContentsOfFile:usedEncoding:error:` (page 57) instead.)

Creating and Initializing a String from an URL

+ `stringWithContentsOfURL:encoding:error:` (page 28)
  Returns a string created by reading data from a given URL interpreted using a given encoding.

- `initWithContentsOfURL:encoding:error:` (page 57)
  Returns an `NSString` object initialized by reading data from a given URL interpreted using a given encoding.

+ `stringWithContentsOfURL:usedEncoding:error:` (page 28)
  Returns a string created by reading data from a given URL and returns by reference the encoding used to interpret the data.
- `initWithContentsOfURL:usedEncoding:error:` (page 58)
  Returns an `NSString` object initialized by reading data from a given URL and returns by reference the encoding used to interpret the data.
+ `stringWithContentsOfURL:` (page 110) Deprecated in Mac OS X v10.4
  Returns a string created by reading data from the file named by a given URL. (Deprecated. Use `stringWithContentsOfURL:encoding:error:` (page 28) or `stringWithContentsOfURL:usedEncoding:error:` (page 28) instead.)
- `initWithContentsOfURL:` (page 115) Deprecated in Mac OS X v10.4
  Initializes the receiver, a newly allocated `NSString` object, by reading data from the location named by a given URL. (Deprecated. Use `initWithContentsOfURL:encoding:error:` (page 57) or `initWithContentsOfURL:usedEncoding:error:` (page 58) instead.)

Writing to a File or URL

- `writeToFile:atomically:encoding:error:` (page 99)
  Writes the contents of the receiver to a file at a given path using a given encoding.
- `writeToURL:atomically:encoding:error:` (page 100)
  Writes the contents of the receiver to the URL specified by *`url`* using the specified encoding.
- `writeToFile:atomically:` (page 117) Deprecated in Mac OS X v10.4
  Writes the contents of the receiver to the file specified by a given path. (Deprecated. Use `writeToFile:atomically:encoding:error:` (page 99) instead.)
- `writeToURL:atomically:` (page 118) Deprecated in Mac OS X v10.4
  Writes the contents of the receiver to the location specified by a given URL. (Deprecated. Use `writeToURL:atomically:encoding:error:` (page 100) instead.)

Getting a String's Length

- `length` (page 67)
  Returns the number of Unicode characters in the receiver.
- `lengthOfBytesUsingEncoding:` (page 68)
  Returns the number of bytes required to store the receiver in a given encoding.
- `maximumLengthOfBytesUsingEncoding:` (page 71)
  Returns the maximum number of bytes needed to store the receiver in a given encoding.

Getting Characters and Bytes

- `characterAtIndex:` (page 34)
  Returns the character at a given array position.
- `getCharacters:` (page 47)
  Copies all characters from the receiver into a given buffer.

- `getCharacters:range:` (page 48)
  Copies characters from a given range in the receiver into a given buffer.
- `getBytes:maxLength:usedLength:encoding:options:range:remainingRange:` (page 46)
  Gets a given range of characters as bytes in a specified encoding.

Getting C Strings

- `cStringUsingEncoding:` (page 41)
  Returns a representation of the receiver as a C string using a given encoding.
- `getCString:maxLength:encoding:` (page 48)
  Converts the receiver's content to a given encoding and stores them in a buffer.
- `UTF8String` (page 99)
  Returns a null-terminated UTF8 representation of the receiver.
- `cString` (page 111) Deprecated in Mac OS X v10.4
  Returns a representation of the receiver as a C string in the default C-string encoding. (Deprecated. Use `cStringUsingEncoding:` (page 41) or `UTF8String` (page 99) instead.)
- `cStringLength` (page 112) Deprecated in Mac OS X v10.4
  Returns the length in `char`-sized units of the receiver's C-string representation in the default C-string encoding. (Deprecated. Use lengthOfBytesUsingEncoding: (page 68) or maximumLengthOfBytesUsingEncoding: (page 71) instead.)
- `getCString:` (page 112) Deprecated in Mac OS X v10.4
  Invokes `getCString:maxLength:range:remainingRange:` (page 114) with `NSMaximumStringLength` as the maximum length, the receiver's entire extent as the range, and `NULL` for the remaining range. (Deprecated. Use cStringUsingEncoding: (page 41) or `dataUsingEncoding:allowLossyConversion:` (page 42) instead.)
- `getCString:maxLength:` (page 113) Deprecated in Mac OS X v10.4
  Invokes `getCString:maxLength:range:remainingRange:` (page 114) with *`maxLength`* as the maximum length in `char`-sized units, the receiver's entire extent as the range, and `NULL` for the remaining range. (Deprecated. Use getCString:maxLength:encoding: (page 48) instead.)
- `getCString:maxLength:range:remainingRange:` (page 114) Deprecated in Mac OS X v10.4
  Converts the receiver's content to the default C-string encoding and stores them in a given buffer. (Deprecated. Use getCString:maxLength:encoding: (page 48) instead.)
- `lossyCString` (page 117) Deprecated in Mac OS X v10.4
  Returns a representation of the receiver as a C string in the default C-string encoding, possibly losing information in converting to that encoding. (Deprecated. Use cStringUsingEncoding: (page 41) or `dataUsingEncoding:allowLossyConversion:` (page 42) instead.)

Combining Strings

- `stringByAppendingFormat:` (page 84)
  Returns a string made by appending to the receiver a string constructed from a given format string and the following arguments.
- `stringByAppendingString:` (page 86)
  Returns a new string made by appending a given string to the receiver.

- `stringByPaddingToLength:withString:startingAtIndex:` (page 90)
  Returns a new string formed from the receiver by either removing characters from the end, or by appending as many occurrences as necessary of a given pad string.

Dividing Strings

- `componentsSeparatedByString:` (page 40)
  Returns an array containing substrings from the receiver that have been divided by a given separator.
- `componentsSeparatedByCharactersInSet:` (page 40)
  Returns an array containing substrings from the receiver that have been divided by characters in a given set.
- `stringByTrimmingCharactersInSet:` (page 95)
  Returns a new string made by removing from both ends of the receiver characters contained in a given character set.
- `substringFromIndex:` (page 96)
  Returns a new string containing the characters of the receiver from the one at a given index to the end.
- `substringWithRange:` (page 98)
  Returns a string object containing the characters of the receiver that lie within a given range.
- `substringToIndex:` (page 97)
  Returns a new string containing the characters of the receiver up to, but not including, the one at a given index.

Finding Characters and Substrings

- `rangeOfCharacterFromSet:` (page 76)
  Finds and returns the range in the receiver of the first character from a given character set.
- `rangeOfCharacterFromSet:options:` (page 76)
  Finds and returns the range in the receiver of the first character, using given options, from a given character set.
- `rangeOfCharacterFromSet:options:range:` (page 77)
  Finds and returns the range in the receiver of the first character from a given character set found in a given range with given options.
- `rangeOfString:` (page 79)
  Finds and returns the range of the first occurrence of a given string within the receiver.
- `rangeOfString:options:` (page 80)
  Finds and returns the range of the first occurrence of a given string within the receiver, subject to given options.
- `rangeOfString:options:range:` (page 80)
  Finds and returns the range of the first occurrence of a given string, within the given range of the receiver, subject to given options.
- `rangeOfString:options:range:locale:` (page 81)
  Finds and returns the range of the first occurrence of a given string within a given range of the receiver, subject to given options, using the specified locale, if any.

Replacing Substrings

- `stringByReplacingOccurrencesOfString:withString:` (page 92)
  Returns a new string in which all occurrences of a target string in the receiver are replaced by another given string.
- `stringByReplacingOccurrencesOfString:withString:options:range:` (page 92)
  Returns a new string in which all occurrences of a target string in a specified range of the receiver are replaced by another given string.
- `stringByReplacingCharactersInRange:withString:` (page 91)
  Returns a new string in which the characters in a specified range of the receiver are replaced by a given string.

Determining Line and Paragraph Ranges

- `getLineStart:end:contentsEnd:forRange:` (page 50)
  Returns by reference the beginning of the first line and the end of the last line touched by the given range.
- `lineRangeForRange:` (page 68)
  Returns the range of characters representing the line or lines containing a given range.
- `getParagraphStart:end:contentsEnd:forRange:` (page 51)
  Returns by reference the beginning of the first paragraph and the end of the last paragraph touched by the given range.
- `paragraphRangeForRange:` (page 72)
  Returns the range of characters representing the paragraph or paragraphs containing a given range.

Determining Composed Character Sequences

- `rangeOfComposedCharacterSequenceAtIndex:` (page 78)
  Returns the range in the receiver of the composed character sequence located at a given index.
- `rangeOfComposedCharacterSequencesForRange:` (page 79)
  Returns the range in the receiver of the composed character sequence in a given range.

Converting String Contents Into a Property List

- `propertyList` (page 75)
  Parses the receiver as a text representation of a property list, returning an `NSString`, `NSData`, `NSArray`, or `NSDictionary` object, according to the topmost element.
- `propertyListFromStringsFileFormat` (page 75)
  Returns a dictionary object initialized with the keys and values found in the receiver.

Identifying and Comparing Strings

- `caseInsensitiveCompare:` (page 33)
  Returns the result of invoking `compare:options:` (page 36) with `NSCaseInsensitiveSearch` as the only option.
- `localizedCaseInsensitiveCompare:` (page 69)
  Returns an `NSComparisonResult` value that indicates the lexical ordering of the receiver and a given string using a case-insensitive, localized, comparison.
- `compare:` (page 35)
  Returns the result of invoking `compare:options:range:` (page 37) with no options and the receiver's full extent as the range.
- `localizedCompare:` (page 69)
  Returns an `NSComparisonResult` value that indicates the lexical ordering of the receiver and another given string using a localized comparison.
- `compare:options:` (page 36)
  Returns the result of invoking `compare:options:range:` (page 37) with a given mask as the options and the receiver's full extent as the range.
- `compare:options:range:` (page 37)
  Returns the result of invoking `compare:options:range:locale:` (page 38) with a `nil` locale.
- `compare:options:range:locale:` (page 38)
  Returns an `NSComparisonResult` value that indicates the lexical ordering of a specified range within the receiver and a given string.
- `hasPrefix:` (page 52)
  Returns a Boolean value that indicates whether a given string matches the beginning characters of the receiver.
- `hasSuffix:` (page 53)
  Returns a Boolean value that indicates whether a given string matches the ending characters of the receiver.
- `isEqualToString:` (page 65)
  Returns a Boolean value that indicates whether a given string is equal to the receiver using an literal Unicode-based comparison.
- `hash` (page 52)
  Returns an unsigned integer that can be used as a hash table address.

Folding Strings

- `stringByFoldingWithOptions:locale:` (page 90)
  Returns a string with the given character folding options applied.

Getting a Shared Prefix

- `commonPrefixWithString:options:` (page 35)
  Returns a string containing prefix the receiver and a given string have in common.

Changing Case

- `capitalizedString` (page 33)
  Returns a capitalized representation of the receiver.
- `lowercaseString` (page 70)
  Returns lowercased representation of the receiver.
- `uppercaseString` (page 98)
  Returns an uppercased representation of the receiver.

Getting Strings with Mapping

- `decomposedStringWithCanonicalMapping` (page 43)
  Returns a string made by normalizing the receiver's contents using Form D.
- `decomposedStringWithCompatibilityMapping` (page 43)
  Returns a string made by normalizing the receiver's contents using Form KD.
- `precomposedStringWithCanonicalMapping` (page 74)
  Returns a string made by normalizing the receiver's contents using Form C.
- `precomposedStringWithCompatibilityMapping` (page 74)
  Returns a string made by normalizing the receiver's contents using Form KC.

Getting Numeric Values

- `doubleValue` (page 44)
  Returns the floating-point value of the receiver's text as a `double`.
- `floatValue` (page 46)
  Returns the floating-point value of the receiver's text as a `float`.
- `intValue` (page 64)
  Returns the integer value of the receiver's text.
- `integerValue` (page 64)
  Returns the `NSInteger` value of the receiver's text.
- `longLongValue` (page 70)
  Returns the `long long` value of the receiver's text.
- `boolValue` (page 32)
  Returns the Boolean value of the receiver's text.

Working with Encodings

+ `availableStringEncodings` (page 22)
  Returns a zero-terminated list of the encodings string objects support in the application's environment.
+ `defaultCStringEncoding` (page 23)
  Returns the C-string encoding assumed for any method accepting a C string as an argument.
+ `localizedNameOfStringEncoding:` (page 23)
  Returns a human-readable string giving the name of a given encoding.

- `canBeConvertedToEncoding:` (page 32)
  Returns a Boolean value that indicates whether the receiver can be converted to a given encoding without loss of information.
- `dataUsingEncoding:` (page 42)
  Returns an `NSData` object containing a representation of the receiver encoded using a given encoding.
- `dataUsingEncoding:allowLossyConversion:` (page 42)
  Returns an `NSData` object containing a representation of the receiver encoded using a given encoding.
- `description` (page 44)
  Returns the receiver.
- `fastestEncoding` (page 45)
  Returns the fastest encoding to which the receiver may be converted without loss of information.
- `smallestEncoding` (page 82)
  Returns the smallest encoding to which the receiver can be converted without loss of information.

Working with Paths

+ `pathWithComponents:` (page 25)
  Returns a string built from the strings in a given array by concatenating them with a path separator between each pair.
- `pathComponents` (page 72)
  Returns an array of `NSString` objects containing, in order, each path component of the receiver.
- `completePathIntoString:caseSensitive:matchesIntoArray:filterTypes:` (page 39)
  Interprets the receiver as a path in the file system and attempts to perform filename completion, returning a numeric value that indicates whether a match was possible, and by reference the longest path that matches the receiver.
- `fileSystemRepresentation` (page 45)
  Returns a file system-specific representation of the receiver.
- `getFileSystemRepresentation:maxLength:` (page 49)
  Interprets the receiver as a system-independent path and fills a buffer with a C-string in a format and encoding suitable for use with file-system calls.
- `isAbsolutePath` (page 65)
  Returning a Boolean value that indicates whether the receiver represents an absolute path.
- `lastPathComponent` (page 66)
  Returns the last path component of the receiver.
- `pathExtension` (page 73)
  Interprets the receiver as a path and returns the receiver's extension, if any.
- `stringByAbbreviatingWithTildeInPath` (page 83)
  Returns a new string representing the receiver as a path with a tilde (~) substituted for the full path to the current user's home directory.
- `stringByAppendingPathComponent:` (page 84)
  Returns a new string made by appending to the receiver a given string.
- `stringByAppendingPathExtension:` (page 85)
  Returns a new string made by appending to the receiver an extension separator followed by a given extension.

- `stringByDeletingLastPathComponent` (page 87)
  Returns a new string made by deleting the last path component from the receiver, along with any final path separator.
- `stringByDeletingPathExtension` (page 88)
  Returns a new string made by deleting the extension (if any, and only the last) from the receiver.
- `stringByExpandingTildeInPath` (page 89)
  Returns a new string made by expanding the initial component of the receiver to its full path value.
- `stringByResolvingSymlinksInPath` (page 93)
  Returns a new string made from the receiver by resolving all symbolic links and standardizing path.
- `stringByStandardizingPath` (page 94)
  Returns a new string made by removing extraneous path components from the receiver.
- `stringsByAppendingPaths:` (page 96)
  Returns an array of strings made by separately appending to the receiver each string in in a given array.

Working with URLs

- `stringByAddingPercentEscapesUsingEncoding:` (page 83)
  Returns a representation of the receiver using a given encoding to determine the percent escapes necessary to convert the receiver into a legal URL string.
- `stringByReplacingPercentEscapesUsingEncoding:` (page 93)
  Returns a new string made by replacing in the receiver all percent escapes with the matching characters as determined by a given encoding.

Class Methods

availableStringEncodings

Returns a zero-terminated list of the encodings string objects support in the application's environment.

```
+ (const NSStringEncoding *)availableStringEncodings
```

Return Value
A zero-terminated list of the encodings string objects support in the application's environment.

Discussion
Among the more commonly used encodings are:

```
NSASCIIStringEncoding
NSUnicodeStringEncoding
NSISOLatin1StringEncoding
NSISOLatin2StringEncoding
NSSymbolStringEncoding
```

See the "Constants" (page 101) section for a larger list and descriptions of many supported encodings. In addition to those encodings listed here, you can also use the encodings defined for CFString in Core Foundation; you just need to call the `CFStringConvertEncodingToNSStringEncoding` function to convert them to a usable format.

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

See Also
`+ localizedNameOfStringEncoding:` (page 23)

Declared In
`NSString.h`

defaultCStringEncoding

Returns the C-string encoding assumed for any method accepting a C string as an argument.

```
+ (NSStringEncoding)defaultCStringEncoding
```

Return Value
The C-string encoding assumed for any method accepting a C string as an argument.

Discussion
This method returns a user-dependent encoding who value is derived from user's default language and potentially other factors. You might sometimes need to use this encoding when interpreting user documents with unknown encodings, in the absence of other hints, but in general this encoding should be used rarely, if at all. Note that some potential values might result in unexpected encoding conversions of even fairly straightforward `NSString` content—for example, punctuation characters with a bidirectional encoding.

Methods that accept a C string as an argument use `...CString...` in the keywords for such arguments: for example, `stringWithCString:` (page 110)—note, though, that these are deprecated. The default C-string encoding is determined from system information and can't be changed programmatically for an individual process. See "String Encodings" (page 105) for a full list of supported encodings.

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

Related Sample Code
QTKitCreateMovie
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSString.h`

localizedNameOfStringEncoding:

Returns a human-readable string giving the name of a given encoding.

```
+ (NSString *)localizedNameOfStringEncoding:(NSStringEncoding)encoding
```

Parameters

encoding

A string encoding.

Return Value

A human-readable string giving the name of *encoding* in the current locale's language.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

NSFontAttributeExplorer

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In

`NSString.h`

localizedStringWithFormat:

Returns a string created by using a given format string as a template into which the remaining argument values are substituted according to the user's default locale.

```
+ (id)localizedStringWithFormat:(NSString *)format ...
```

Parameters

format

A format string. See Formatting String Objects for examples of how to use this method, and String Format Specifiers for a list of format specifiers. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *format* is `nil`.

...

A comma-separated list of arguments to substitute into *format*.

Return Value

A string created by using *format* as a template into which the following argument values are substituted according to the formatting information to the user's default locale.

Discussion

This method is equivalent to using `initWithFormat:locale:` (page 61) and passing `[[NSUserDefaults standardUserDefaults] dictionaryRepresentation]` as the locale argument.

As an example of formatting, this method replaces the decimal according to the locale in `%f` and `%d` substitutions, and calls `descriptionWithLocale:` instead of `description` where necessary.

This code excerpt creates a string from another string and a `float`:

```
NSString *myString = [NSString localizedStringWithFormat:@"%@:  %f\n", @"Cost",
 1234.56];
```

The resulting string has the value "`Cost: 1234.560000\n`" if the locale is `en_US`, and "`Cost: 1234,560000\n`" if the locale is `fr_FR`.

See Formatting String Objects for more information.

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

See Also
`+ stringWithFormat:` (page 30)
`- initWithFormat:locale:` (page 61)

Related Sample Code
FilterDemo
GridCalendar

Declared In
`NSString.h`

pathWithComponents:

Returns a string built from the strings in a given array by concatenating them with a path separator between each pair.

```
+ (NSString *)pathWithComponents:(NSArray *)components
```

Parameters

components

An array of `NSString` objects representing a file path. To create an absolute path, use a slash mark ("/") as the first component. To include a trailing path divider, use an empty string as the last component.

Return Value
A string built from the strings in *components* by concatenating them (in the order they appear in the array) with a path separator between each pair.

Discussion
This method doesn't clean up the path created; use `stringByStandardizingPath` (page 94) to resolve empty components, references to the parent directory, and so on.

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

See Also
`- pathComponents` (page 72)

Declared In
`NSPathUtilities.h`

string

Returns an empty string.

```
+ (id)string
```

Return Value
An empty string.

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

See Also
- `init` (page 53)

Declared In
`NSString.h`

stringWithCharacters:length:

Returns a string containing a given number of characters taken from a given C array of Unicode characters.

```
+ (id)stringWithCharacters:(const unichar *)chars length:(NSUInteger)length
```

Parameters

chars

A C array of Unicode characters; the value must not be `NULL`.

Important: Raises an exception if *chars* is `NULL`, even if *length* is `0`.

length

The number of characters to use from *chars*.

Return Value
A string containing *length* Unicode characters taken (starting with the first) from *chars*.

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

See Also
- `initWithCharacters:length:` (page 55)

Related Sample Code
CrossEvents
PDFKitLinker2
QCCocoaComponent
SharedMemory

Declared In
`NSString.h`

stringWithContentsOfFile:encoding:error:

Returns a string created by reading data from the file at a given path interpreted using a given encoding.

```
+ (id)stringWithContentsOfFile:(NSString *)path encoding:(NSStringEncoding)enc
    error:(NSError **)error
```

Parameters

path

A path to a file.

enc

The encoding of the file at *path*.

error

If an error occurs, upon returns contains an `NSError` object that describes the problem. If you are not interested in possible errors, pass in `NULL`.

Return Value

A string created by reading data from the file named by *path* using the encoding, *enc*. If the file can't be opened or there is an encoding error, returns `nil`.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- initWithContentsOfFile:encoding:error:` (page 56)

Related Sample Code

JSPong

LSMSmartCategorizer

Declared In

`NSString.h`

stringWithContentsOfFile:usedEncoding:error:

Returns a string created by reading data from the file at a given path and returns by reference the encoding used to interpret the file.

```
+ (id)stringWithContentsOfFile:(NSString *)path usedEncoding:(NSStringEncoding
    *)enc error:(NSError **)error
```

Parameters

path

A path to a file.

enc

Upon return, if the file is read successfully, contains the encoding used to interpret the file at *path*.

error

If an error occurs, upon returns contains an `NSError` object that describes the problem. If you are not interested in possible errors, you may pass in `NULL`.

Return Value

A string created by reading data from the file named by *path*. If the file can't be opened or there is an encoding error, returns `nil`.

Discussion

This method attempts to determine the encoding of the file at *path*.

Availability

Available in Mac OS X v10.4 and later.

See Also
- `initWithContentsOfFile:encoding:error:` (page 56)

Declared In
`NSString.h`

stringWithContentsOfURL:encoding:error:

Returns a string created by reading data from a given URL interpreted using a given encoding.

```
+ (id)stringWithContentsOfURL:(NSURL *)url encoding:(NSStringEncoding)enc
    error:(NSError **)error
```

Parameters

url

The URL to read.

enc

The encoding of the data at *url*.

error

If an error occurs, upon returns contains an `NSError` object that describes the problem. If you are not interested in possible errors, you may pass in `NULL`.

Return Value
A string created by reading data from *URL* using the encoding, *enc*. If the URL can't be opened or there is an encoding error, returns `nil`.

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

See Also
- `+ stringWithContentsOfURL:usedEncoding:error:` (page 28)
- `- initWithContentsOfURL:encoding:error:` (page 57)

Declared In
`NSString.h`

stringWithContentsOfURL:usedEncoding:error:

Returns a string created by reading data from a given URL and returns by reference the encoding used to interpret the data.

```
+ (id)stringWithContentsOfURL:(NSURL *)url usedEncoding:(NSStringEncoding *)enc
    error:(NSError **)error
```

Parameters

url

The URL from which to read data.

enc

Upon return, if *url* is read successfully, contains the encoding used to interpret the data.

error

If an error occurs, upon returns contains an `NSError` object that describes the problem. If you are not interested in possible errors, you may pass in `NULL`.

Return Value
A string created by reading data from *url*. If the URL can't be opened or there is an encoding error, returns `nil`.

Discussion
This method attempts to determine the encoding at *url*.

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

See Also
`+ stringWithContentsOfURL:encoding:error:` (page 28)
`- initWithContentsOfURL:usedEncoding:error:` (page 58)

Declared In
`NSString.h`

stringWithCString:encoding:

Returns a string containing the bytes in a given C array, interpreted according to a given encoding.

```
+ (id)stringWithCString:(const char *)cString encoding:(NSStringEncoding)enc
```

Parameters

cString

A C array of bytes. The array must end with a `NULL` character; intermediate `NULL` characters are not allowed.

enc

The encoding of *cString*.

Return Value
A string containing the characters described in *cString*.

Discussion
If *cString* is not a `NULL`-terminated C string, or *encoding* does not match the actual encoding, the results are undefined.

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

See Also
`- initWithCString:encoding:` (page 58)

Related Sample Code
CAPlayThrough
QTKitCreateMovie
QTMetadataEditor
SMARTQuery
VideoHardwareInfo

Declared In
NSString.h

stringWithFormat:

Returns a string created by using a given format string as a template into which the remaining argument values are substituted.

```
+ (id)stringWithFormat:(NSString *)format, ...
```

Parameters

format

A format string. See Formatting String Objects for examples of how to use this method, and String Format Specifiers for a list of format specifiers. This value must not be nil.

Important: Raises an NSInvalidArgumentException if *format* is nil.

...

A comma-separated list of arguments to substitute into *format*.

Return Value
A string created by using *format* as a template into which the remaining argument values are substituted according to the canonical locale.

Discussion
This method is similar to localizedStringWithFormat: (page 24), but using the canonical locale to format numbers. This is useful, for example, if you want to produce "non-localized" formatting which needs to be written out to files and parsed back later.

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

See Also
- initWithFormat: (page 60)
+ localizedStringWithFormat: (page 24)

Related Sample Code
CoreRecipes
Fiendishthngs
LSMSmartCategorizer
MyPhoto
Quartz Composer WWDC 2005 TextEdit

Declared In
NSString.h

stringWithString:

Returns a string created by copying the characters from another given string.

```
+ (id)stringWithString:(NSString *)aString
```

Parameters

aString

The string from which to copy characters. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aString* is `nil`.

Return Value

A string created by copying the characters from *aString*.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- initWithString:` (page 63)

Related Sample Code

OpenGL Screensaver

QTAudioExtractionPanel

QTMetadataEditor

SurfaceVertexProgram

TimelineToTC

Declared In

`NSString.h`

stringWithUTF8String:

Returns a string created by copying the data from a given C array of UTF8-encoded bytes.

```
+ (id)stringWithUTF8String:(const char *)bytes
```

Parameters

bytes

A `NULL`-terminated C array of bytes in UTF8 encoding.

Important: Raises an exception if *bytes* is `NULL`.

Return Value

A string created by copying the data from *bytes*.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- initWithString:` (page 63)

Related Sample Code

DockTile

DynamicProperties

MyPhoto

QTMetadataEditor
StickiesExample

Declared In
`NSString.h`

Instance Methods

boolValue

Returns the Boolean value of the receiver's text.

```
- (BOOL)boolValue
```

Return Value
The Boolean value of the receiver's text. Returns `YES` on encountering one of "Y", "y", "T", "t", or a digit 1-9—the method ignores any trailing characters. Returns `NO` if the receiver doesn't begin with a valid decimal text representation of a number.

Discussion
The method assumes a decimal representation and skips whitespace at the beginning of the string. It also skips initial whitespace characters, or optional -/+ sign followed by zeroes.

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

See Also
`- integerValue` (page 64)
`- scanInt:` (`NSScanner`)

Declared In
`NSString.h`

canBeConvertedToEncoding:

Returns a Boolean value that indicates whether the receiver can be converted to a given encoding without loss of information.

```
- (BOOL)canBeConvertedToEncoding:(NSStringEncoding)encoding
```

Parameters
encoding
A string encoding.

Return Value
`YES` if the receiver can be converted to *encoding* without loss of information. Returns `NO` if characters would have to be changed or deleted in the process of changing encodings.

Discussion
If you plan to actually convert a string, the `dataUsingEncoding:...` methods return `nil` on failure, so you can avoid the overhead of invoking this method yourself by simply trying to convert the string.

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

See Also
- `dataUsingEncoding:allowLossyConversion:` (page 42)

Declared In
`NSString.h`

capitalizedString

Returns a capitalized representation of the receiver.

```
- (NSString *)capitalizedString
```

Return Value
A string with the first character from each word in the receiver changed to its corresponding uppercase value, and all remaining characters set to their corresponding lowercase values.

Discussion
A "word" here is any sequence of characters delimited by spaces, tabs, or line terminators (listed under `getLineStart:end:contentsEnd:forRange:` (page 50)). Other common word delimiters such as hyphens and other punctuation aren't considered, so this method may not generally produce the desired results for multiword strings.

Case transformations aren't guaranteed to be symmetrical or to produce strings of the same lengths as the originals. See `lowercaseString` (page 70) for an example.

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

See Also
- `lowercaseString` (page 70)
- `uppercaseString` (page 98)

Related Sample Code
Mountains
StickiesExample

Declared In
`NSString.h`

caseInsensitiveCompare:

Returns the result of invoking `compare:options:` (page 36) with `NSCaseInsensitiveSearch` as the only option.

```
- (NSComparisonResult)caseInsensitiveCompare:(NSString *)aString
```

Parameters

aString

The string with which to compare the receiver.

This value must not be `nil`. If this value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

Return Value

The result of invoking `compare:options:` (page 36) with `NSCaseInsensitiveSearch` as the only option.

Discussion

If you are comparing strings to present to the end-user, you should typically use `localizedCaseInsensitiveCompare:` (page 69) instead.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `localizedCaseInsensitiveCompare:` (page 69)
- `compare:options:` (page 36)

Related Sample Code

IdentitySample

People

Declared In

`NSString.h`

characterAtIndex:

Returns the character at a given array position.

```
- (unichar)characterAtIndex:(NSUInteger)index
```

Parameters

index

The index of the character to retrieve. The index value must not lie outside the bounds of the receiver.

Return Value

The character at the array position given by *index*.

Discussion

Raises an `NSRangeException` if *index* lies beyond the end of the receiver.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `getCharacters:` (page 47)
- `getCharacters:range:` (page 48)

Related Sample Code

CocoaDVDPlayer

CubePuzzle

EnhancedAudioBurn

NSOpenGL Fullscreen
PDFKitLinker2

Declared In
`NSString.h`

commonPrefixWithString:options:

Returns a string containing prefix the receiver and a given string have in common.

```
- (NSString *)commonPrefixWithString:(NSString *)aString
    options:(NSStringCompareOptions)mask
```

Parameters

aString

The string with which to compare the receiver.

mask

Options for the comparison. The following search options may be specified by combining them with the C bitwise `OR` operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`. See *String Programming Guide for Cocoa* for details on these options.

Return Value
A string containing characters the receiver and *aString* have in common, starting from the beginning of each up to the first characters that aren't equivalent.

Discussion
The returned string is based on the characters of the receiver. For example, if the receiver is "Ma¨dchen" and *aString* is "Mädchenschule", the string returned is "Ma¨dchen", not "Mädchen".

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

See Also
`- hasPrefix:` (page 52)

Declared In
`NSString.h`

compare:

Returns the result of invoking `compare:options:range:` (page 37) with no options and the receiver's full extent as the range.

```
- (NSComparisonResult)compare:(NSString *)aString
```

Parameters

aString

The string with which to compare the receiver.

This value must not be `nil`. If this value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

Return Value
The result of invoking `compare:options:range:` (page 37) with no options and the receiver's full extent as the range.

Discussion
If you are comparing strings to present to the end-user, you should typically use `localizedCompare:` (page 69) or `localizedCaseInsensitiveCompare:` (page 69) instead.

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

See Also
- `localizedCompare:` (page 69)
- `localizedCaseInsensitiveCompare:` (page 69)
- `compare:options:` (page 36)
- `caseInsensitiveCompare:` (page 33)
- `isEqualToString:` (page 65)

Related Sample Code
QTCoreVideo102
QTCoreVideo103
QTCoreVideo201
QTCoreVideo202
QTCoreVideo301

Declared In
`NSString.h`

compare:options:

Returns the result of invoking `compare:options:range:` (page 37) with a given mask as the options and the receiver's full extent as the range.

```
- (NSComparisonResult)compare:(NSString *)aString
    options:(NSStringCompareOptions)mask
```

Parameters

aString

The string with which to compare the receiver.

This value must not be `nil`. If this value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

mask

Options for the search—you can combine any of the following using a C bitwise OR operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSNumericSearch`. See *String Programming Guide for Cocoa* for details on these options.

Return Value
The result of invoking `compare:options:range:` (page 37) with a given mask as the options and the receiver's full extent as the range.

Discussion
If you are comparing strings to present to the end-user, you should typically use `localizedCompare:` (page 69) or `localizedCaseInsensitiveCompare:` (page 69) instead, or use `compare:options:range:locale:` (page 38) and pass the user's locale.

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

See Also
- `localizedCompare:` (page 69)
- `localizedCaseInsensitiveCompare:` (page 69)
- `compare:options:range:locale:` (page 38)
- `caseInsensitiveCompare:` (page 33)
- `isEqualToString:` (page 65)

Declared In
`NSString.h`

compare:options:range:

Returns the result of invoking `compare:options:range:locale:` (page 38) with a `nil` locale.

```
- (NSComparisonResult)compare:(NSString *)aString
    options:(NSStringCompareOptions)mask range:(NSRange)range
```

Parameters

aString

The string with which to compare the range of the receiver specified by *range*.

This value must not be `nil`. If this value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

mask

Options for the search—you can combine any of the following using a C bitwise OR operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSNumericSearch`.

See *String Programming Guide for Cocoa* for details on these options.

range

The range of the receiver over which to perform the comparison. The range must not exceed the bounds of the receiver.

Important: Raises an `NSRangeException` if *range* exceeds the bounds of the receiver.

Return Value
The result of invoking `compare:options:range:locale:` (page 38) with a `nil` locale.

Discussion
If you are comparing strings to present to the end-user, you should typically use `compare:options:range:locale:` (page 38) instead and pass the user's locale (`currentLocale` [`NSLocale`]).

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

See Also
- `- localizedCompare:` (page 69)
- `- localizedCaseInsensitiveCompare:` (page 69)
- `- compare:options:` (page 36)
- `- caseInsensitiveCompare:` (page 33)
- `- isEqualToString:` (page 65)

Declared In
`NSString.h`

compare:options:range:locale:

Returns an `NSComparisonResult` value that indicates the lexical ordering of a specified range within the receiver and a given string.

```
- (NSComparisonResult)compare:(NSString *)aString
    options:(NSStringCompareOptions)mask range:(NSRange)range locale:(id)locale
```

Parameters

aString

The string with which to compare the range of the receiver specified by *range*.

This value must not be `nil`. If this value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

mask

Options for the search—you can combine any of the following using a C bitwise OR operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSNumericSearch`.

See *String Programming Guide for Cocoa* for details on these options.

range

The range of the receiver over which to perform the comparison. The range must not exceed the bounds of the receiver.

Important: Raises an `NSRangeException` if *range* exceeds the bounds of the receiver.

locale

An instance of `NSLocale`. If this value not `nil` and is not an instance of `NSLocale`, uses the current locale instead. If you are comparing strings to present to the end-user, you should typically pass the user's locale (`currentLocale` [`NSLocale`]).

The locale argument affects both equality and ordering algorithms. For example, in some locales, accented characters are ordered immediately after the base; other locales order them after "z".

Return Value
`NSOrderedAscending` if the substring of the receiver given by *range* precedes *aString* in lexical ordering for the locale given in *dict*, `NSOrderedSame` if the substring of the receiver and *aString* are equivalent in lexical value, and `NSOrderedDescending` if the substring of the receiver follows *aString*.

Special Considerations

Prior to Mac OS X v10.5, the *locale* argument was an instance of `NSDictionary`. On Mac OS X v10.5 and later, if you pass an instance of `NSDictionary` the current locale is used instead.

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

See Also
- `localizedCompare:` (page 69)
- `localizedCaseInsensitiveCompare:` (page 69)
- `caseInsensitiveCompare:` (page 33)
- `compare:` (page 35)
- `compare:options:` (page 36)
- `compare:options:range:` (page 37)
- `isEqualToString:` (page 65)

Declared In
`NSString.h`

completePathIntoString:caseSensitive:matchesIntoArray:filterTypes:

Interprets the receiver as a path in the file system and attempts to perform filename completion, returning a numeric value that indicates whether a match was possible, and by reference the longest path that matches the receiver.

```
- (NSUInteger)completePathIntoString:(NSString **)outputName caseSensitive:(BOOL)flag
    matchesIntoArray:(NSArray **)outputArray filterTypes:(NSArray *)filterTypes
```

Parameters

outputName
Upon return, contains the longest path that matches the receiver.

flag
If `YES`, the methods considers case for possible completions.

outputArray
Upon return, contains all matching filenames.

filterTypes
An array of `NSString` objects specifying path extensions to consider for completion. only paths whose extensions (not including the extension separator) match one of those strings.

Return Value
`0` if no matches are found and `1` if exactly one match is found. In the case of multiple matches, returns the actual number of matching paths if *outputArray* is provided, or simply a positive value if *outputArray* is `NULL`.

Discussion
You can check for the existence of matches without retrieving by passing `NULL` as *outputArray*.

Note that this method only works with file paths (not, for example, string representations of URLs).

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

Declared In
`NSPathUtilities.h`

componentsSeparatedByCharactersInSet:

Returns an array containing substrings from the receiver that have been divided by characters in a given set.

```
- (NSArray *)componentsSeparatedByCharactersInSet:(NSCharacterSet *)separator
```

Parameters

separator

A character set containing the characters to to use to split the receiver. Must not be `nil`.

Return Value

An `NSArray` object containing substrings from the receiver that have been divided by characters in *separator*.

Discussion

The substrings in the array appear in the order they did in the receiver. Adjacent occurrences of the separator characters produce empty strings in the result. Similarly, if the string begins or ends with separator characters, the first or last substring, respectively, is empty.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `componentsSeparatedByString:` (page 40)
- `stringByTrimmingCharactersInSet:` (page 95)

Declared In

`NSString.h`

componentsSeparatedByString:

Returns an array containing substrings from the receiver that have been divided by a given separator.

```
- (NSArray *)componentsSeparatedByString:(NSString *)separator
```

Parameters

separator

The separator string.

Return Value

An `NSArray` object containing substrings from the receiver that have been divided by *separator*.

Discussion

The substrings in the array appear in the order they did in the receiver. Adjacent occurrences of the separator string produce empty strings in the result. Similarly, if the string begins or ends with the separator, the first or last substring, respectively, is empty. For example, this code fragment:

```
NSString *list = @"Norman, Stanley, Fletcher";
NSArray *listItems = [list componentsSeparatedByString:@", "];
```

produces an array `{ @"Norman", @"Stanley", @"Fletcher" }`.

If *list* begins with a comma and space—for example, `", Norman, Stanley, Fletcher"`—the array has these contents: `{ @"", @"Norman", @"Stanley", @"Fletcher" }`

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

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

See Also
componentsJoinedByString: (NSArray)
- pathComponents (page 72)

Related Sample Code
Birthdays
CoreRecipes
iSpend
QTKitMovieShuffler
Reminders

Declared In
NSString.h

cStringUsingEncoding:

Returns a representation of the receiver as a C string using a given encoding.

```
- (const char *)cStringUsingEncoding:(NSStringEncoding)encoding
```

Parameters
encoding
The encoding for the returned C string.

Return Value
A C string representation of the receiver using the encoding specified by *encoding*. Returns NULL if the receiver cannot be losslessly converted to *encoding*.

Discussion
The returned C string is guaranteed to be valid only until either the receiver is freed, or until the current autorelease pool is emptied, whichever occurs first. You should copy the C string or use getCString:maxLength:encoding: (page 48) if it needs to store the C string beyond this time.

You can use canBeConvertedToEncoding: (page 32) to check whether a string can be losslessly converted to *encoding*. If it can't, you can use dataUsingEncoding:allowLossyConversion: (page 42) to get a C-string representation using *encoding*, allowing some loss of information (note that the data returned by dataUsingEncoding:allowLossyConversion: is not a strict C-string since it does not have a NULL terminator).

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

See Also
- getCString: (page 112)
- canBeConvertedToEncoding: (page 32)
+ defaultCStringEncoding (page 23)
- cStringLength (page 112)

- `getCharacters:` (page 47)
- `UTF8String` (page 99)

Related Sample Code
CocoaDVDPlayer
Core Data HTML Store

Declared In
`NSString.h`

dataUsingEncoding:

Returns an `NSData` object containing a representation of the receiver encoded using a given encoding.

```
- (NSData *)dataUsingEncoding:(NSStringEncoding)encoding
```

Parameters

encoding
A string encoding.

Return Value
The result of invoking `dataUsingEncoding:allowLossyConversion:` (page 42) with `NO` as the second argument (that is, requiring lossless conversion).

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

Related Sample Code
EnhancedAudioBurn
QTMetadataEditor
QTSSConnectionMonitor
QTSSInspector
Sketch-112

Declared In
`NSString.h`

dataUsingEncoding:allowLossyConversion:

Returns an `NSData` object containing a representation of the receiver encoded using a given encoding.

```
- (NSData *)dataUsingEncoding:(NSStringEncoding)encoding
    allowLossyConversion:(BOOL)flag
```

Parameters

encoding
A string encoding.

flag
If `YES`, then allows characters to be removed or altered in conversion.

Return Value
An `NSData` object containing a representation of the receiver encoded using *`encoding`*. Returns `nil` if *`flag`* is `NO` and the receiver can't be converted without losing some information (such as accents or case).

Discussion
If *`flag`* is `YES` and the receiver can't be converted without losing some information, some characters may be removed or altered in conversion. For example, in converting a character from `NSUnicodeStringEncoding` to `NSASCIIStringEncoding`, the character 'Á' becomes 'A', losing the accent.

This method creates an external representation (with a byte order marker, if necessary, to indicate endianness) to ensure that the resulting `NSData` object can be written out to a file safely. The result of this method, when lossless conversion is made, is the default "plain text" format for encoding and is the recommended way to save or transmit a string object.

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

See Also
`+ availableStringEncodings` (page 22)
`- canBeConvertedToEncoding:` (page 32)

Related Sample Code
JavaSplashScreen
Spotlight

Declared In
`NSString.h`

decomposedStringWithCanonicalMapping

Returns a string made by normalizing the receiver's contents using Form D.

```
- (NSString *)decomposedStringWithCanonicalMapping
```

Return Value
A string made by normalizing the receiver's contents using the Unicode Normalization Form D.

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

See Also
`- precomposedStringWithCanonicalMapping` (page 74)
`- decomposedStringWithCompatibilityMapping` (page 43)

Declared In
`NSString.h`

decomposedStringWithCompatibilityMapping

Returns a string made by normalizing the receiver's contents using Form KD.

```
- (NSString *)decomposedStringWithCompatibilityMapping
```

Return Value
A string made by normalizing the receiver's contents using the Unicode Normalization Form KD.

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

See Also
- `precomposedStringWithCompatibilityMapping` (page 74)
- `decomposedStringWithCanonicalMapping` (page 43)

Declared In
`NSString.h`

description

Returns the receiver.

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

Return Value
The receiver.

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

Declared In
`NSString.h`

doubleValue

Returns the floating-point value of the receiver's text as a `double`.

```
- (double)doubleValue
```

Return Value
The floating-point value of the receiver's text as a `double`. Returns `HUGE_VAL` or `-HUGE_VAL` on overflow, 0.0 on underflow. Returns `0.0` if the receiver doesn't begin with a valid text representation of a floating-point number.

Discussion
This method skips any whitespace at the beginning of the string. This method uses formatting information stored in the non-localized value; use an `NSScanner` object for localized scanning of numeric values from a string.

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

See Also
- `floatValue` (page 46)
- `longLongValue` (page 70)
- `integerValue` (page 64)
- `scanDouble:(NSScanner)`

Related Sample Code
JavaFrameEmbedding example
QTMetadataEditor
TimelineToTC
TrackBall

Declared In
`NSString.h`

fastestEncoding

Returns the fastest encoding to which the receiver may be converted without loss of information.

```
- (NSStringEncoding)fastestEncoding
```

Return Value
The fastest encoding to which the receiver may be converted without loss of information.

Discussion
"Fastest" applies to retrieval of characters from the string. This encoding may not be space efficient.

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

See Also
- `smallestEncoding` (page 82)
- `getCharacters:range:` (page 48)

Declared In
`NSString.h`

fileSystemRepresentation

Returns a file system-specific representation of the receiver.

```
- (const char *)fileSystemRepresentation
```

Return Value
A file system-specific representation of the receiver, as described for `getFileSystemRepresentation:maxLength:` (page 49).

Discussion
The returned C string will be automatically freed just as a returned object would be released; your code should copy the representation or use `getFileSystemRepresentation:maxLength:` (page 49) if it needs to store the representation outside of the autorelease context in which the representation is created.

Raises an `NSCharacterConversionException` if the receiver can't be represented in the file system's encoding.

Note that this method only works with file paths (not, for example, string representations of URLs).

To convert a `char *` path (such as you might get from a C library routine) to an `NSString` object, use `NSFileManager`'s `stringWithFileSystemRepresentation:length:` method.

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

Related Sample Code
JavaSplashScreen

Declared In
`NSPathUtilities.h`

floatValue

Returns the floating-point value of the receiver's text as a `float`.

```
- (float)floatValue
```

Return Value
The floating-point value of the receiver's text as a `float`, skipping whitespace at the beginning of the string. Returns `HUGE_VAL` or `-HUGE_VAL` on overflow, `0.0` on underflow. Also returns `0.0` if the receiver doesn't begin with a valid text representation of a floating-point number.

Discussion
This method uses formatting information stored in the non-localized value; use an `NSScanner` object for localized scanning of numeric values from a string.

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

See Also
- `doubleValue` (page 44)
- `longLongValue` (page 70)
- `integerValue` (page 64)
- `scanFloat:` (`NSScanner`)

Related Sample Code
WhackedTV

Declared In
`NSString.h`

getBytes:maxLength:usedLength:encoding:options:range:remainingRange:

Gets a given range of characters as bytes in a specified encoding.

```
- (BOOL)getBytes:(void *)buffer maxLength:(NSUInteger)maxBufferCount
    usedLength:(NSUInteger *)usedBufferCount encoding:(NSStringEncoding)encoding
    options:(NSStringEncodingConversionOptions)options range:(NSRange)range
    remainingRange:(NSRangePointer)leftover
```

Parameters
buffer
A buffer into which to store the bytes from the receiver. The returned bytes are *not* `NULL`-terminated.

maxBufferCount
The maximum number of bytes to write to *buffer*.

usedBufferCount
The number of bytes used from *buffer*. Pass NULL if you do not need this value.

encoding
The encoding to use for the returned bytes.

options
A mask to specify options to use for converting the receiver's contents to *encoding* (if conversion is necessary).

range
The range of characters in the receiver to get.

leftover
The remaining range. Pass NULL If you do not need this value.

Return Value
YES if some characters were converted, otherwise NO.

Discussion
Conversion might stop when the buffer fills, but it might also stop when the conversion isn't possible due to the chosen encoding.

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

Declared In
NSString.h

getCharacters:

Copies all characters from the receiver into a given buffer.

```
- (void)getCharacters:(unichar *)buffer
```

Parameters

buffer
Upon return, contains the characters from the receiver. *buffer* must be large enough to contain all characters in the string ([string length]*sizeof(unichar)).

Discussion
Invokes getCharacters:range: (page 48) with *buffer* and the entire extent of the receiver as the range.

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

See Also
- length (page 67)

Related Sample Code
JSheets

Declared In
NSString.h

getCharacters:range:

Copies characters from a given range in the receiver into a given buffer.

```
- (void)getCharacters:(unichar *)buffer range:(NSRange)aRange
```

Parameters

buffer

Upon return, contains the characters from the receiver. *buffer* must be large enough to contain the characters in the range *aRange* (aRange.length*sizeof(unichar)).

aRange

The range of characters to retrieve. The range must not exceed the bounds of the receiver.

Important: Raises an NSRangeException if any part of *aRange* lies beyond the bounds of the receiver.

Discussion
This method does not add a NULL character.

The abstract implementation of this method uses characterAtIndex: (page 34) repeatedly, correctly extracting the characters, though very inefficiently. Subclasses should override it to provide a fast implementation.

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

Declared In
NSString.h

getCString:maxLength:encoding:

Converts the receiver's content to a given encoding and stores them in a buffer.

```
- (BOOL)getCString:(char *)buffer maxLength:(NSUInteger)maxBufferCount
    encoding:(NSStringEncoding)encoding
```

Parameters

buffer

Upon return, contains the converted C-string plus the NULL termination byte. The buffer must include room for *maxBufferCount* bytes.

maxBufferCount

The maximum number of bytes in the string to return in buffer (*including* the NULL termination byte).

encoding

The encoding for the returned C string.

Return Value
YES if the operation was successful, otherwise NO. Returns NO if conversion is not possible due to encoding errors or if *buffer* is too small.

Discussion
Note that in the treatment of the *maxBufferCount* argument, this method differs from the deprecated getCString:maxLength: (page 113) method which it replaces. (The buffer should include room for *maxBufferCount* bytes; this number should accommodate the expected size of the return value plus the NULL termination byte, which this method adds.)

You can use canBeConvertedToEncoding: (page 32) to check whether a string can be losslessly converted to *encoding*. If it can't, you can use dataUsingEncoding:allowLossyConversion: (page 42) to get a C-string representation using *encoding*, allowing some loss of information (note that the data returned by dataUsingEncoding:allowLossyConversion: is not a strict C-string since it does not have a NULL terminator).

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

See Also
- cStringUsingEncoding: (page 41)
- canBeConvertedToEncoding: (page 32)
- getCharacters: (page 47)
- UTF8String (page 99)

Related Sample Code
QTMetadataEditor

Declared In
NSString.h

getFileSystemRepresentation:maxLength:

Interprets the receiver as a system-independent path and fills a buffer with a C-string in a format and encoding suitable for use with file-system calls.

```
- (BOOL)getFileSystemRepresentation:(char *)buffer maxLength:(NSUInteger)maxLength
```

Parameters

buffer

Upon return, contains a C-string that represent the receiver as as a system-independent path, plus the NULL termination byte. The size of *buffer* must be large enough to contain *maxLength* bytes.

maxLength

The maximum number of bytes in the string to return in *buffer* (including a terminating NULL character, which this method adds).

Return Value
YES if *buffer* is successfully filled with a file-system representation, otherwise NO (for example, if *maxLength* would be exceeded or if the receiver can't be represented in the file system's encoding).

Discussion
This method operates by replacing the abstract path and extension separator characters ('/' and '.' respectively) with their equivalents for the operating system. If the system-specific path or extension separator appears in the abstract representation, the characters it is converted to depend on the system (unless they're identical to the abstract separators).

Note that this method only works with file paths (not, for example, string representations of URLs).

The following example illustrates the use of the *maxLength* argument. The first method invocation returns failure as the file representation of the string (@"/mach_kernel") is 12 bytes long and the value passed as the *maxLength* argument (12) does not allow for the addition of a NULL termination byte.

```
char filenameBuffer[13];
BOOL success;
success = [@"/mach_kernel" getFileSystemRepresentation:filenameBuffer
maxLength:12];
// success == NO
// Changing the length to include the NULL character does work
success = [@"/mach_kernel" getFileSystemRepresentation:filenameBuffer
maxLength:13];
// success == YES
```

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

See Also
- fileSystemRepresentation (page 45)

Declared In
NSPathUtilities.h

getLineStart:end:contentsEnd:forRange:

Returns by reference the beginning of the first line and the end of the last line touched by the given range.

```
- (void)getLineStart:(NSUInteger *)startIndex end:(NSUInteger *)lineEndIndex
    contentsEnd:(NSUInteger *)contentsEndIndex forRange:(NSRange)aRange
```

Parameters

startIndex
Upon return, contains the index of the first character of the line containing the beginning of *aRange*. Pass NULL if you do not need this value (in which case the work to compute the value isn't performed).

lineEndIndex
Upon return, contains the index of the first character past the terminator of the line containing the end of *aRange*. Pass NULL if you do not need this value (in which case the work to compute the value isn't performed).

contentsEndIndex
Upon return, contains the index of the first character of the terminator of the line containing the end of *aRange*. Pass NULL if you do not need this value (in which case the work to compute the value isn't performed).

aRange
A range within the receiver. The value must not exceed the bounds of the receiver.

Important: Raises an NSRangeException if any part of *aRange* lies beyond the end of the string.

Discussion
A line is delimited by any of these characters, the longest possible sequence being preferred to any shorter:

- U+000D (\r or CR)

- U+2028 (Unicode line separator)
- U+000A (\n or LF)
- U+2029 (Unicode paragraph separator)
- \r\n, in that order (also known as CRLF)

If *aRange* is contained with a single line, of course, the returned indexes all belong to that line. You can use the results of this method to construct ranges for lines by using the start index as the range's location and the difference between the end index and the start index as the range's length.

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

See Also
- lineRangeForRange: (page 68)
- substringWithRange: (page 98)

Declared In
NSString.h

getParagraphStart:end:contentsEnd:forRange:

Returns by reference the beginning of the first paragraph and the end of the last paragraph touched by the given range.

```
- (void)getParagraphStart:(NSUInteger *)startIndex end:(NSUInteger *)endIndex
    contentsEnd:(NSUInteger *)contentsEndIndex forRange:(NSRange)aRange
```

Parameters

startIndex

Upon return, contains the index of the first character of the paragraph containing the beginning of *aRange*. Pass NULL if you do not need this value (in which case the work to compute the value isn't performed).

endIndex

Upon return, contains the index of the first character past the terminator of the paragraph containing the end of *aRange*. Pass NULL if you do not need this value (in which case the work to compute the value isn't performed).

contentsEndIndex

Upon return, contains the index of the first character of the terminator of the paragraph containing the end of *aRange*. Pass NULL if you do not need this value (in which case the work to compute the value isn't performed).

aRange

A range within the receiver. The value must not exceed the bounds of the receiver.

Discussion
If *aRange* is contained with a single paragraph, of course, the returned indexes all belong to that paragraph. Similar to getLineStart:end:contentsEnd:forRange: (page 50), you can use the results of this method to construct the ranges for paragraphs.

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

See Also
- `paragraphRangeForRange:` (page 72)

Declared In
`NSString.h`

hash

Returns an unsigned integer that can be used as a hash table address.

```
- (NSUInteger)hash
```

Return Value
An unsigned integer that can be used as a hash table address.

Discussion
If two string objects are equal (as determined by the `isEqualToString:` (page 65) method), they must have the same hash value. The abstract implementation of this method fulfills this requirement, so subclasses of `NSString` shouldn't override it.

You should not rely on this method returning the same hash value across releases of Mac OS X.

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

Declared In
`NSString.h`

hasPrefix:

Returns a Boolean value that indicates whether a given string matches the beginning characters of the receiver.

```
- (BOOL)hasPrefix:(NSString *)aString
```

Parameters
aString
A string.

Return Value
`YES` if *aString* matches the beginning characters of the receiver, otherwise `NO`. Returns `NO` if *aString* is empty.

Discussion
This method is a convenience for comparing strings using the `NSAnchoredSearch` option. See *String Programming Guide for Cocoa* for more information.

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

See Also
- `hasSuffix:` (page 53)
- `compare:options:range:` (page 37)

Related Sample Code
Reminders

Declared In
`NSString.h`

hasSuffix:

Returns a Boolean value that indicates whether a given string matches the ending characters of the receiver.

```
- (BOOL)hasSuffix:(NSString *)aString
```

Parameters

aString
A string.

Return Value
`YES` if *aString* matches the ending characters of the receiver, otherwise `NO`. Returns `NO` if *aString* is empty.

Discussion
This method is a convenience for comparing strings using the `NSAnchoredSearch` and `NSBackwardsSearch` options. See *String Programming Guide for Cocoa* for more information.

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

See Also
- `hasPrefix:` (page 52)
- `compare:options:range:` (page 37)

Declared In
`NSString.h`

init

Returns an initialized `NSString` object that contains no characters.

```
- (id)init
```

Return Value
An initialized `NSString` object that contains no characters. The returned object may be different from the original receiver.

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

See Also
+ `string` (page 25)

Declared In
`NSString.h`

initWithBytes:length:encoding:

Returns an initialized `NSString` object containing a given number of bytes from a given C array of bytes in a given encoding.

```
- (id)initWithBytes:(const void *)bytes length:(NSUInteger)length
    encoding:(NSStringEncoding)encoding
```

Parameters

bytes

A C array of bytes in the encoding specified by *encoding*. The array must not contain `NULL`.

length

The number of bytes to use from *bytes*.

encoding

The character encoding of *bytes*.

Return Value

An initialized `NSString` object containing *length* bytes from *bytes* interpreted using the encoding *encoding*. The returned object may be different from the original receiver.

Availability

Available in Mac OS X v10.3 and later.

See Also

- initWithBytesNoCopy:length:encoding:freeWhenDone: (page 54)

Related Sample Code

VideoHardwareInfo

Declared In

`NSString.h`

initWithBytesNoCopy:length:encoding:freeWhenDone:

Returns an initialized `NSString` object that contains a given number of bytes from a given C array of bytes in a given encoding, and optionally frees the array on deallocation.

```
- (id)initWithBytesNoCopy:(void *)bytes length:(NSUInteger)length
    encoding:(NSStringEncoding)encoding freeWhenDone:(BOOL)flag
```

Parameters

bytes

A C array of bytes in the encoding specified by *encoding*. The array must not contain `NULL`.

length

The number of bytes to use from *bytes*.

encoding

The character encoding of *bytes*.

flag

If `YES`, the receiver will free the memory when it no longer needs the data; if `NO` it won't.

Return Value

An initialized `NSString` object containing *length* bytes from *bytes* interpreted using the encoding *encoding*. The returned object may be different from the original receiver.

Special Considerations

If an error occurs during the creation of the string, then *bytes* is not freed even if *flag* is YES. In this case, the caller is responsible for freeing the buffer. This allows the caller to continue trying to create a string with the buffer, without having the buffer deallocated.

Availability

Available in Mac OS X v10.3 and later.

See Also

- initWithBytes:length:encoding: (page 54)

Related Sample Code

QTRecorder

Declared In

NSString.h

initWithCharacters:length:

Returns an initialized NSString object that contains a given number of characters from a given C array of Unicode characters.

```
- (id)initWithCharacters:(const unichar *)characters length:(NSUInteger)length
```

Parameters

characters

A C array of Unicode characters; the value must not be NULL.

Important: Raises an exception if *characters* is NULL, even if *length* is 0.

length

The number of characters to use from *characters*.

Return Value

An initialized NSString object containing *length* characters taken from *characters*. The returned object may be different from the original receiver.

Availability

Available in Mac OS X v10.0 and later.

See Also

+ stringWithCharacters:length: (page 26)

Declared In

NSString.h

initWithCharactersNoCopy:length:freeWhenDone:

Returns an initialized NSString object that contains a given number of characters from a given C array of Unicode characters.

```
- (id)initWithCharactersNoCopy:(unichar *)characters length:(NSUInteger)length
    freeWhenDone:(BOOL)flag
```

Parameters

characters

A C array of Unicode characters.

length

The number of characters to use from *characters*.

flag

If YES, the receiver will free the memory when it no longer needs the characters; if NO it won't.

Return Value

An initialized NSString object that contains *length* characters from *characters*. The returned object may be different from the original receiver.

Special Considerations

If an error occurs during the creation of the string, then *bytes* is not freed even if *flag* is YES. In this case, the caller is responsible for freeing the buffer. This allows the caller to continue trying to create a string with the buffer, without having the buffer deallocated.

Availability

Available in Mac OS X v10.0 and later.

See Also

+ stringWithCharacters:length: (page 26)

Declared In

NSString.h

initWithContentsOfFile:encoding:error:

Returns an NSString object initialized by reading data from the file at a given path using a given encoding.

```
- (id)initWithContentsOfFile:(NSString *)path encoding:(NSStringEncoding)enc
    error:(NSError **)error
```

Parameters

path

A path to a file.

enc

The encoding of the file at *path*.

error

If an error occurs, upon return contains an NSError object that describes the problem. If you are not interested in possible errors, pass in NULL.

Return Value

An NSString object initialized by reading data from the file named by *path* using the encoding, *enc*. The returned object may be different from the original receiver. If the file can't be opened or there is an encoding error, returns nil.

Availability

Available in Mac OS X v10.4 and later.

See Also
+ stringWithContentsOfFile:encoding:error: (page 26)
- initWithContentsOfFile:usedEncoding:error: (page 57)

Declared In
NSString.h

initWithContentsOfFile:usedEncoding:error:

Returns an NSString object initialized by reading data from the file at a given path and returns by reference the encoding used to interpret the characters.

```
- (id)initWithContentsOfFile:(NSString *)path usedEncoding:(NSStringEncoding *)enc
    error:(NSError **)error
```

Parameters

path

A path to a file.

enc

Upon return, if the file is read successfully, contains the encoding used to interpret the file at *path*.

error

If an error occurs, upon returns contains an NSError object that describes the problem. If you are not interested in possible errors, pass in NULL.

Return Value
An NSString object initialized by reading data from the file named by *path*. The returned object may be different from the original receiver. If the file can't be opened or there is an encoding error, returns nil.

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

See Also
+ stringWithContentsOfFile:encoding:error: (page 26)
- initWithContentsOfFile:encoding:error: (page 56)

Declared In
NSString.h

initWithContentsOfURL:encoding:error:

Returns an NSString object initialized by reading data from a given URL interpreted using a given encoding.

```
- (id)initWithContentsOfURL:(NSURL *)url encoding:(NSStringEncoding)enc
    error:(NSError **)error
```

Parameters

url

The URL to read.

enc

The encoding of the file at *path*.

error

If an error occurs, upon returns contains an `NSError` object that describes the problem. If you are not interested in possible errors, pass in `NULL`.

Return Value
An `NSString` object initialized by reading data from *url*. The returned object may be different from the original receiver. If the URL can't be opened or there is an encoding error, returns `nil`.

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

See Also
`+ stringWithContentsOfURL:encoding:error:` (page 28)

Declared In
`NSString.h`

initWithContentsOfURL:usedEncoding:error:

Returns an `NSString` object initialized by reading data from a given URL and returns by reference the encoding used to interpret the data.

```
- (id)initWithContentsOfURL:(NSURL *)url usedEncoding:(NSStringEncoding *)enc
    error:(NSError **)error
```

Parameters

url

The URL from which to read data.

enc

Upon return, if *url* is read successfully, contains the encoding used to interpret the data.

error

If an error occurs, upon returns contains an `NSError` object that describes the problem. If you are not interested in possible errors, pass in `NULL`.

Return Value
An `NSString` object initialized by reading data from *url*. If *url* can't be opened or the encoding cannot be determined, returns `nil`. The returned initialized object might be different from the original receiver

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

See Also
`+ stringWithContentsOfURL:usedEncoding:error:` (page 28)

Declared In
`NSString.h`

initWithCString:encoding:

Returns an `NSString` object initialized using the characters in a given C array, interpreted according to a given encoding.

```
- (id)initWithCString:(const char *)nullTerminatedCString
    encoding:(NSStringEncoding)encoding
```

Parameters

nullTerminatedCString

A C array of characters. The array must end with a `NULL` character; intermediate `NULL` characters are not allowed.

encoding

The encoding of *nullTerminatedCString*.

Return Value

An `NSString` object initialized using the characters from *nullTerminatedCString*. The returned object may be different from the original receiver

Discussion

If *nullTerminatedCString* is not a `NULL`-terminated C string, or *encoding* does not match the actual encoding, the results are undefined.

Availability

Available in Mac OS X v10.4 and later.

See Also

`+ stringWithCString:` (page 110)

`- initWithCStringNoCopy:length:freeWhenDone:` (page 116)

`+ defaultCStringEncoding` (page 23)

Declared In

`NSString.h`

initWithData:encoding:

Returns an `NSString` object initialized by converting given data into Unicode characters using a given encoding.

```
- (id)initWithData:(NSData *)data encoding:(NSStringEncoding)encoding
```

Parameters

data

An `NSData` object containing bytes in *encoding* and the default plain text format (that is, pure content with no attributes or other markups) for that encoding.

encoding

The encoding used by *data*.

Return Value

An `NSString` object initialized by converting the bytes in *data* into Unicode characters using *encoding*. The returned object may be different from the original receiver. Returns `nil` if the initialization fails for some reason (for example if *data* does not represent valid data for *encoding*).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AutoUpdater

EnhancedAudioBurn
GridCalendar
Moriarity
NameAndPassword

Declared In
NSString.h

initWithFormat:

Returns an *NSString* object initialized by using a given format string as a template into which the remaining argument values are substituted.

```
- (id)initWithFormat:(NSString *)format ...
```

Parameters

format

A format string. See Formatting String Objects for examples of how to use this method, and String Format Specifiers for a list of format specifiers. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *format* is `nil`.

...

A comma-separated list of arguments to substitute into *format*.

Return Value
An *NSString* object initialized by using *format* as a template into which the remaining argument values are substituted according to the canonical locale. The returned object may be different from the original receiver.

Discussion
Invokes `initWithFormat:locale:arguments:` (page 62) with `nil` as the locale, hence using the canonical locale to format numbers. This is useful, for example, if you want to produce "non-localized" formatting which needs to be written out to files and parsed back later.

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

See Also
`+ stringWithFormat:` (page 30)
`- initWithFormat:locale:arguments:` (page 62)

Declared In
NSString.h

initWithFormat:arguments:

Returns an *NSString* object initialized by using a given format string as a template into which the remaining argument values are substituted according to the user's default locale.

```
- (id)initWithFormat:(NSString *)format arguments:(va_list)argList
```

Parameters

format

A format string. See Formatting String Objects for examples of how to use this method, and String Format Specifiers for a list of format specifiers. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *format* is `nil`.

argList

A list of arguments to substitute into *format*.

Return Value

An *NSString* object initialized by using *format* as a template into which the values in *argList* are substituted according to the user's default locale. The returned object may be different from the original receiver.

Discussion

Invokes `initWithFormat:locale:arguments:` (page 62) with `nil` as the locale.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ stringWithFormat:` (page 30)

Declared In

`NSString.h`

initWithFormat:locale:

Returns an *NSString* object initialized by using a given format string as a template into which the remaining argument values are substituted according to given locale information.

```
- (id)initWithFormat:(NSString *)format locale:(id)locale ...
```

Parameters

format

A format string. See Formatting String Objects for examples of how to use this method, and String Format Specifiers for a list of format specifiers. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *format* is `nil`.

locale

This may be an instance of `NSDictionary` containing locale information or an instance of `NSLocale`. If this value is `nil`, uses the canonical locale.

To use a dictionary containing the current user's locale, you can use `[[NSUserDefaults standardUserDefaults] dictionaryRepresentation]`.

...

A comma-separated list of arguments to substitute into *format*.

Discussion

Invokes `initWithFormat:locale:arguments:` (page 62) with *locale* as the locale.

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

See Also
+ localizedStringWithFormat: (page 24)

Declared In
NSString.h

initWithFormat:locale:arguments:

Returns an *NSString* object initialized by using a given format string as a template into which the remaining argument values are substituted according to given locale information.

```
- (id)initWithFormat:(NSString *)format locale:(id)locale arguments:(va_list)argList
```

Parameters

format

A format string. See Formatting String Objects for examples of how to use this method, and String Format Specifiers for a list of format specifiers. This value must not be nil.

Important: Raises an NSInvalidArgumentException if *format* is nil.

locale

This may be an instance of NSDictionary containing locale information or an instance of NSLocale. If this value is nil, uses the canonical locale.

To use a dictionary containing the current user's locale, you can use [[NSUserDefaults standardUserDefaults] dictionaryRepresentation].

argList

A list of arguments to substitute into *format*.

Return Value
An *NSString* object initialized by using *format* as a template into which values in *argList* are substituted according the locale information in *locale*. The returned object may be different from the original receiver.

Discussion
The following code fragment illustrates how to create a string from *myArgs*, which is derived from a string object with the value "Cost:" and an int with the value 32:

```
va_list myArgs;

NSString *myString = [[NSString alloc] initWithFormat:@"%@:  %d\n"
        locale:[[NSUserDefaults standardUserDefaults] dictionaryRepresentation]
        arguments:myArgs];
```

The resulting string has the value "Cost: 32\n".

See *String Programming Guide for Cocoa* for more information.

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

See Also
- initWithFormat:arguments: (page 60)

Declared In
NSString.h

initWithString:

Returns an NSString object initialized by copying the characters from another given string.

```
- (id)initWithString:(NSString *)aString
```

Parameters

aString
The string from which to copy characters. This value must not be nil.

Important: Raises an NSInvalidArgumentException if *aString* is nil.

Return Value
An NSString object initialized by copying the characters from *aString*. The returned object may be different from the original receiver.

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

See Also
+ stringWithString: (page 30)

Declared In
NSString.h

initWithUTF8String:

Returns an NSString object initialized by copying the characters a given C array of UTF8-encoded bytes.

```
- (id)initWithUTF8String:(const char *)bytes
```

Parameters

bytes
A NULL-terminated C array of bytes in UTF-8 encoding. This value must not be NULL.

Important: Raises an exception if *bytes* is NULL.

Return Value
An NSString object initialized by copying the bytes from *bytes*. The returned object may be different from the original receiver.

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

See Also
+ stringWithUTF8String: (page 31)

Related Sample Code
Reminders

Declared In
NSString.h

integerValue

Returns the NSInteger value of the receiver's text.

```
- (NSInteger)integerValue
```

Return Value
The NSInteger value of the receiver's text, assuming a decimal representation and skipping whitespace at the beginning of the string. Returns 0 if the receiver doesn't begin with a valid decimal text representation of a number.

Discussion
This method uses formatting information stored in the non-localized value; use an NSScanner object for localized scanning of numeric values from a string.

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

See Also
- doubleValue (page 44)
- floatValue (page 46)
- scanInt: (NSScanner)

Related Sample Code
Core Data HTML Store

Declared In
NSString.h

intValue

Returns the integer value of the receiver's text.

```
- (int)intValue
```

Return Value
The integer value of the receiver's text, assuming a decimal representation and skipping whitespace at the beginning of the string. Returns INT_MAX or INT_MIN on overflow. Returns 0 if the receiver doesn't begin with a valid decimal text representation of a number.

Discussion
This method uses formatting information stored in the non-localized value; use an NSScanner object for localized scanning of numeric values from a string.

Special Considerations

On Mac OS X v10.5 and later, use `integerValue` (page 64) instead.

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

See Also
- `integerValue` (page 64)
- `doubleValue` (page 44)
- `floatValue` (page 46)
- `scanInt:` (`NSScanner`)

Related Sample Code
AlbumToSlideshow
DatePicker
QTAudioExtractionPanel
QTMetadataEditor
WebKitDOMElementPlugIn

Declared In
`NSString.h`

isAbsolutePath

Returning a Boolean value that indicates whether the receiver represents an absolute path.

```
- (BOOL)isAbsolutePath
```

Return Value
`YES` if the receiver (if interpreted as a path) represents an absolute path, otherwise `NO` (if the receiver represents a relative path).

Discussion
See *String Programming Guide for Cocoa* for more information on paths.

Note that this method only works with file paths (not, for example, string representations of URLs). The method does not check the filesystem for the existence of the path (use `fileExistsAtPath:` or similar methods in `NSFileManager` for that task).

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

Declared In
`NSPathUtilities.h`

isEqualToString:

Returns a Boolean value that indicates whether a given string is equal to the receiver using an literal Unicode-based comparison.

```
- (BOOL)isEqualToString:(NSString *)aString
```

Parameters

aString

The string with which to compare the receiver.

Return Value

YES if *aString* is equivalent to the receiver (if they have the same id or if they are NSOrderedSame in a literal comparison), otherwise NO.

Discussion

The comparison uses the canonical representation of strings, which for a particular string is the length of the string plus the Unicode characters that make up the string. When this method compares two strings, if the individual Unicodes are the same, then the strings are equal, regardless of the backing store. "Literal" when applied to string comparison means that various Unicode decomposition rules are not applied and Unicode characters are individually compared. So, for instance, "Ö" represented as the composed character sequence "O" and umlaut would not compare equal to "Ö" represented as one Unicode character.

Special Considerations

When you know both objects are strings, this method is a faster way to check equality than isEqual:.

Availability

Available in Mac OS X v10.0 and later.

See Also

- compare:options:range: (page 37)

Related Sample Code

Core Data HTML Store

NameAndAddress

People

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In

NSString.h

lastPathComponent

Returns the last path component of the receiver.

```
- (NSString *)lastPathComponent
```

Return Value

The last path component of the receiver.

Discussion

The following table illustrates the effect of lastPathComponent on a variety of different paths:

Receiver's String Value	String Returned
"/tmp/scratch.tiff"	"scratch.tiff"
"/tmp/scratch"	"scratch"

Receiver's String Value	String Returned
"/tmp/"	"tmp"
"scratch"	"scratch"
"/"	"/"

Note that this method only works with file paths (not, for example, string representations of URLs).

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

Related Sample Code
EnhancedAudioBurn
OpenGLCompositorLab
Quartz Composer WWDC 2005 TextEdit
StickiesExample
TextEditPlus

Declared In
NSPathUtilities.h

length

Returns the number of Unicode characters in the receiver.

```
- (NSUInteger)length
```

Return Value
The number of Unicode characters in the receiver.

Discussion
The number returned includes the individual characters of composed character sequences, so you cannot use this method to determine if a string will be visible when printed or how long it will appear.

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

See Also
- lengthOfBytesUsingEncoding: (page 68)
sizeWithAttributes: (NSString Additions)

Related Sample Code
iSpend
People
Quartz Composer WWDC 2005 TextEdit
StickiesExample
VertexPerformanceTest

Declared In
NSString.h

lengthOfBytesUsingEncoding:

Returns the number of bytes required to store the receiver in a given encoding.

```
- (NSUInteger)lengthOfBytesUsingEncoding:(NSStringEncoding)enc
```

Parameters

enc

The encoding for which to determine the receiver's length.

Return Value

The number of bytes required to store the receiver in the encoding *enc* in a non-external representation. The length does not include space for a terminating `NULL` character.

Discussion

The result is exact and is returned in `O(n)` time.

Availability

Available in Mac OS X v10.4 and later.

See Also

- `maximumLengthOfBytesUsingEncoding:` (page 71)
- `length` (page 67)

Related Sample Code

Core Data HTML Store

Declared In

`NSString.h`

lineRangeForRange:

Returns the range of characters representing the line or lines containing a given range.

```
- (NSRange)lineRangeForRange:(NSRange)aRange
```

Parameters

aRange

A range within the receiver.

Return Value

The range of characters representing the line or lines containing *aRange*, including the line termination characters.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `paragraphRangeForRange:` (page 72)
- `getLineStart:end:contentsEnd:forRange:` (page 50)
- `substringWithRange:` (page 98)

Related Sample Code

iSpend

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In
NSString.h

localizedCaseInsensitiveCompare:

Returns an NSComparisonResult value that indicates the lexical ordering of the receiver and a given string using a case-insensitive, localized, comparison.

```
- (NSComparisonResult)localizedCaseInsensitiveCompare:(NSString *)aString
```

Parameters

aString

The string with which to compare the receiver.

This value must not be nil. If this value is nil, the behavior is undefined and may change in future versions of Mac OS X.

Return Value
NSOrderedAscending the receiver precedes *aString* in lexical ordering, NSOrderedSame the receiver and *aString* are equivalent in lexical value, and NSOrderedDescending if the receiver follows *aString*.

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

See Also
- compare:options:range:locale: (page 38)

Related Sample Code
NewsReader

Declared In
NSString.h

localizedCompare:

Returns an NSComparisonResult value that indicates the lexical ordering of the receiver and another given string using a localized comparison.

```
- (NSComparisonResult)localizedCompare:(NSString *)aString
```

Parameters

aString

The string with which to compare the receiver.

This value must not be nil. If this value is nil, the behavior is undefined and may change in future versions of Mac OS X.

Return Value
NSOrderedAscending the receiver precedes *string* in lexical ordering, NSOrderedSame the receiver and *string* are equivalent in lexical value, and NSOrderedDescending if the receiver follows *string*.

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

See Also
- `compare:options:range:locale:` (page 38)

Declared In
`NSString.h`

longLongValue

Returns the `long long` value of the receiver's text.

```
- (long long)longLongValue
```

Return Value
The `long long` value of the receiver's text, assuming a decimal representation and skipping whitespace at the beginning of the string. Returns `LLONG_MAX` or `LLONG_MIN` on overflow. Returns `0` if the receiver doesn't begin with a valid decimal text representation of a number.

Discussion
This method uses formatting information stored in the non-localized value; use an `NSScanner` object for localized scanning of numeric values from a string.

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

See Also
- `doubleValue` (page 44)
- `floatValue` (page 46)
- `scanInt:` (`NSScanner`)

Declared In
`NSString.h`

lowercaseString

Returns lowercased representation of the receiver.

```
- (NSString *)lowercaseString
```

Return Value
A string with each character from the receiver changed to its corresponding lowercase value.

Discussion
Case transformations aren't guaranteed to be symmetrical or to produce strings of the same lengths as the originals. The result of this statement:

```
lcString = [myString lowercaseString];
```

might not be equal to this statement:

```
lcString = [[myString uppercaseString] lowercaseString];
```

For example, the uppercase form of "ß" in German is "SS", so converting "Straße" to uppercase, then lowercase, produces this sequence of strings:

"Straße"
"STRASSE"
"strasse"

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

See Also
- `capitalizedString` (page 33)
- `uppercaseString` (page 98)

Related Sample Code
NewsReader
People
Quartz Composer WWDC 2005 TextEdit
StickiesExample
TextEditPlus

Declared In
`NSString.h`

maximumLengthOfBytesUsingEncoding:

Returns the maximum number of bytes needed to store the receiver in a given encoding.

```
- (NSUInteger)maximumLengthOfBytesUsingEncoding:(NSStringEncoding)enc
```

Parameters

enc
The encoding for which to determine the receiver's length.

Return Value
The maximum number of bytes needed to store the receiver in *encoding* in a non-external representation. The length does not include space for a terminating `NULL` character.

Discussion
The result is an estimate and is returned in `O(1)` time; the estimate may be considerably greater than the actual length needed.

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

See Also
- `lengthOfBytesUsingEncoding:` (page 68)
- `length` (page 67)

Declared In
`NSString.h`

paragraphRangeForRange:

Returns the range of characters representing the paragraph or paragraphs containing a given range.

```
- (NSRange)paragraphRangeForRange:(NSRange)aRange
```

Parameters

aRange

A range within the receiver. The range must not exceed the bounds of the receiver.

Return Value

The range of characters representing the paragraph or paragraphs containing *aRange*, including the paragraph termination characters.

Availability

Available in Mac OS X v10.3 and later.

See Also

- `getParagraphStart:end:contentsEnd:forRange:` (page 51)
- `lineRangeForRange:` (page 68)

Declared In

`NSString.h`

pathComponents

Returns an array of `NSString` objects containing, in order, each path component of the receiver.

```
- (NSArray *)pathComponents
```

Return Value

An array of `NSString` objects containing, in order, each path component of the receiver.

Discussion

The strings in the array appear in the order they did in the receiver. If the string begins or ends with the path separator, then the first or last component, respectively, will contain the separator. Empty components (caused by consecutive path separators) are deleted. For example, this code excerpt:

```
NSString *path = @"tmp/scratch";
NSArray *pathComponents = [path pathComponents];
```

produces an array with these contents:

Index	Path Component
0	“tmp”
1	“scratch”

If the receiver begins with a slash—for example, “`/tmp/scratch`”—the array has these contents:

Index	Path Component
0	“/”

Index	Path Component
1	“tmp”
2	“scratch”

If the receiver has no separators—for example, “scratch”—the array contains the string itself, in this case “scratch”.

Note that this method only works with file paths (not, for example, string representations of URLs).

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

See Also
+ pathWithComponents: (page 25)
- stringByStandardizingPath (page 94)
- componentsSeparatedByString: (page 40)

Related Sample Code
CoreRecipes
CustomSave
ObjectPath

Declared In
NSPathUtilities.h

pathExtension

Interprets the receiver as a path and returns the receiver’s extension, if any.

```
- (NSString *)pathExtension
```

Return Value
The receiver’s extension, if any (not including the extension divider).

Discussion
The following table illustrates the effect of pathExtension on a variety of different paths:

Receiver’s String Value	String Returned
“/tmp/scratch.tiff”	“tiff”
“/tmp/scratch”	“” (an empty string)
“/tmp/”	“” (an empty string)
“/tmp/scratch..tiff”	“tiff”

Note that this method only works with file paths (not, for example, string representations of URLs).

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

Related Sample Code
GLChildWindowDemo
Quartz Composer WWDC 2005 TextEdit
Sketch-112
StickiesExample
TextEditPlus

Declared In
NSPathUtilities.h

precomposedStringWithCanonicalMapping

Returns a string made by normalizing the receiver's contents using Form C.

```
- (NSString *)precomposedStringWithCanonicalMapping
```

Return Value
A string made by normalizing the receiver's contents using the Unicode Normalization Form C.

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

See Also
- precomposedStringWithCompatibilityMapping (page 74)
- decomposedStringWithCanonicalMapping (page 43)

Declared In
NSString.h

precomposedStringWithCompatibilityMapping

Returns a string made by normalizing the receiver's contents using Form KC.

```
- (NSString *)precomposedStringWithCompatibilityMapping
```

Return Value
A string made by normalizing the receiver's contents using the Unicode Normalization Form KC.

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

See Also
- precomposedStringWithCanonicalMapping (page 74)
- decomposedStringWithCompatibilityMapping (page 43)

Declared In
NSString.h

propertyList

Parses the receiver as a text representation of a property list, returning an `NSString`, `NSData`, `NSArray`, or `NSDictionary` object, according to the topmost element.

```
- (id)propertyList
```

Return Value
A property list representation of returning an `NSString`, `NSData`, `NSArray`, or `NSDictionary` object, according to the topmost element.

Discussion
The receiver must contain a string in a property list format. For a discussion of property list formats, see *Property List Programming Guide*.

Important: Raises an `NSParseErrorException` if the receiver cannot be parsed as a property list.

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

See Also
`- propertyListFromStringsFileFormat` (page 75)
`+ stringWithContentsOfFile:` (page 109)

Declared In
`NSString.h`

propertyListFromStringsFileFormat

Returns a dictionary object initialized with the keys and values found in the receiver.

```
- (NSDictionary *)propertyListFromStringsFileFormat
```

Return Value
A dictionary object initialized with the keys and values found in the receiver

Discussion
The receiver must contain text in the format used for `.strings` files. In this format, keys and values are separated by an equal sign, and each key-value pair is terminated with a semicolon. The value is optional—if not present, the equal sign is also omitted. The keys and values themselves are always strings enclosed in straight quotation marks. Comments may be included, delimited by `/*` and `*/` as for ANSI C comments. Here's a short example of a strings file:

```
/* Question in confirmation panel for quitting. */
"Confirm Quit" = "Are you sure you want to quit?";

/* Message when user tries to close unsaved document */
"Close or Save" = "Save changes before closing?";

/* Word for Cancel */
"Cancel";
```

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

See Also
- `propertyList` (page 75)
+ `stringWithContentsOfFile:` (page 109)

Declared In
`NSString.h`

rangeOfCharacterFromSet:

Finds and returns the range in the receiver of the first character from a given character set.

```
- (NSRange)rangeOfCharacterFromSet:(NSCharacterSet *)aSet
```

Parameters

aSet

A character set. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aSet* is `nil`.

Return Value
The range in the receiver of the first character found from *aSet*. Returns a range of `{NSNotFound, 0}` if none of the characters in *aSet* are found.

Discussion
Invokes `rangeOfCharacterFromSet:options:` (page 76) with no options.

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

Declared In
`NSString.h`

rangeOfCharacterFromSet:options:

Finds and returns the range in the receiver of the first character, using given options, from a given character set.

```
- (NSRange)rangeOfCharacterFromSet:(NSCharacterSet *)aSet
    options:(NSStringCompareOptions)mask
```

Parameters

aSet

A character set. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aSet* is `nil`.

mask

A mask specifying search options. The following options may be specified by combining them with the C bitwise `OR` operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSBackwardsSearch`. See *String Programming Guide for Cocoa* for details on these options.

Return Value
The range in the receiver of the first character found from *aSet*. Returns a range of `{NSNotFound, 0}` if none of the characters in *aSet* are found.

Discussion
Invokes `rangeOfCharacterFromSet:options:range:` (page 77) with *mask* for the options and the entire extent of the receiver for the range.

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

Declared In
`NSString.h`

rangeOfCharacterFromSet:options:range:

Finds and returns the range in the receiver of the first character from a given character set found in a given range with given options.

```
- (NSRange)rangeOfCharacterFromSet:(NSCharacterSet *)aSet
    options:(NSStringCompareOptions)mask range:(NSRange)aRange
```

Parameters

aSet

A character set. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aSet* is `nil`.

mask

A mask specifying search options. The following options may be specified by combining them with the C bitwise `OR` operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSBackwardsSearch`. See *String Programming Guide for Cocoa* for details on these options.

aRange

The range in which to search. *aRange* must not exceed the bounds of the receiver.

Important: Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the string.

Return Value
The range in the receiver of the first character found from *aSet* within *aRange*. Returns a range of `{NSNotFound, 0}` if none of the characters in *aSet* are found.

Discussion
Because pre-composed characters in *aSet* can match composed character sequences in the receiver, the length of the returned range can be greater than `1`. For example, if you search for "ü" in the string "stru¨del", the returned range is `{3,2}`.

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

Related Sample Code
VertexPerformanceTest

Declared In
NSString.h

rangeOfComposedCharacterSequenceAtIndex:

Returns the range in the receiver of the composed character sequence located at a given index.

```
- (NSRange)rangeOfComposedCharacterSequenceAtIndex:(NSUInteger)anIndex
```

Parameters

anIndex

The index of a character in the receiver. The value must not exceed the bounds of the receiver.

Return Value
The range in the receiver of the composed character sequence located at *anIndex*.

Discussion
The composed character sequence includes the first base character found at or before *anIndex*, and its length includes the base character and all non-base characters following the base character.

If you want to write a method to adjust an arbitrary range so it includes the composed character sequences on its boundaries, you can create a method such as the following:

```
- (NSRange)adjustRange:(NSRange)aRange
{
    NSUInteger index, endIndex;
    NSRange newRange, endRange;

    // Check for validity of range
    if ( aRange.location >= [self length] ||
         aRange.location + aRange.length > [self length] )
    {
        [NSException raise:NSRangeException format:@"Invalid  range %@.",
        NSStringFromRange(aRange)];
    }

    index = aRange.location;
    newRange = [self rangeOfComposedCharacterSequenceAtIndex:index];

    index = aRange.location + aRange.length - 1;
    endRange = [self rangeOfComposedCharacterSequenceAtIndex:index];
    endIndex = endRange.location + endRange.length;

    newRange.length = endIndex - newRange.location;

    return newRange;
}
```

First, `adjustRange:` corrects the location for the beginning of *aRange*, storing it in *newRange*. It then works at the end of *aRange*, correcting the location and storing it in *endIndex*. Finally, it sets the length of *newRange* to the difference between *endIndex* and the new range's location.

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

See Also
- `rangeOfComposedCharacterSequencesForRange:` (page 79)

Declared In
`NSString.h`

rangeOfComposedCharacterSequencesForRange:

Returns the range in the receiver of the composed character sequence in a given range.

```
- (NSRange)rangeOfComposedCharacterSequencesForRange:(NSRange)range
```

Parameters

range

A range in the receiver. The range must not exceed the bounds of the receiver.

Return Value
The range in the receiver of the composed character sequence in *range*.

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

See Also
- `rangeOfComposedCharacterSequenceAtIndex:` (page 78)

Declared In
`NSString.h`

rangeOfString:

Finds and returns the range of the first occurrence of a given string within the receiver.

```
- (NSRange)rangeOfString:(NSString *)aString
```

Parameters

aString

The string to search for. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aString* is `nil`.

Return Value
An `NSRange` structure giving the location and length in the receiver of the first occurrence of *aString*. Returns `{NSNotFound, 0}` if *aString* is not found or is empty (@"").

Discussion
Invokes `rangeOfString:options:` (page 80) with no options.

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

Related Sample Code
People

QTSSConnectionMonitor
QTSSInspector

Declared In
NSString.h

rangeOfString:options:

Finds and returns the range of the first occurrence of a given string within the receiver, subject to given options.

```
- (NSRange)rangeOfString:(NSString *)aString options:(NSStringCompareOptions)mask
```

Parameters

aString

The string to search for. This value must not be nil.

Important: Raises an NSInvalidArgumentException if *aString* is nil.

mask

A mask specifying search options. The following options may be specified by combining them with the C bitwise OR operator: NSCaseInsensitiveSearch, NSLiteralSearch, NSBackwardsSearch, NSAnchoredSearch. See *String Programming Guide for Cocoa* for details on these options.

Return Value
An NSRange structure giving the location and length in the receiver of the first occurrence of *aString*, modulo the options in *mask*. Returns {NSNotFound, 0} if *aString* is not found or is empty (@"").

Discussion
Invokes rangeOfString:options:range: (page 80) with the options specified by *mask* and the entire extent of the receiver as the range.

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

Related Sample Code
Sketch-112

Declared In
NSString.h

rangeOfString:options:range:

Finds and returns the range of the first occurrence of a given string, within the given range of the receiver, subject to given options.

```
- (NSRange)rangeOfString:(NSString *)aString options:(NSStringCompareOptions)mask
    range:(NSRange)aRange
```

Parameters

aString

The string for which to search. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aString* is `nil`.

mask

A mask specifying search options. The following options may be specified by combining them with the C bitwise `OR` operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSBackwardsSearch`, and `NSAnchoredSearch`. See *String Programming Guide for Cocoa* for details on these options.

aRange

The range within the receiver for which to search for *aString*.

Important: Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the string.

Return Value

An `NSRange` structure giving the location and length in the receiver of *aString* within *aRange* in the receiver, modulo the options in *mask*. The range returned is relative to the start of the string, not to the passed-in range. Returns `{NSNotFound, 0}` if *aString* is not found or is empty (`@""`).

Discussion

The length of the returned range and that of *aString* may differ if equivalent composed character sequences are matched.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

VertexPerformanceTest

Declared In

`NSString.h`

rangeOfString:options:range:locale:

Finds and returns the range of the first occurrence of a given string within a given range of the receiver, subject to given options, using the specified locale, if any.

```
- (NSRange)rangeOfString:(NSString *)aString options:(NSStringCompareOptions)mask
    range:(NSRange)searchRange locale:(NSLocale *)locale
```

Parameters

aString

The string for which to search. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aString* is `nil`.

mask

A mask specifying search options. The following options may be specified by combining them with the C bitwise `OR` operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSBackwardsSearch`, and `NSAnchoredSearch`. See *String Programming Guide for Cocoa* for details on these options.

aRange

The range within the receiver for which to search for *aString*.

Important: Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the string.

locale

The locale to use when comparing the receiver with *aString*. If this value is `nil`, uses the current locale.

The locale argument affects the equality checking algorithm. For example, for the Turkish locale, case-insensitive compare matches "I" to "ı" (Unicode code point U+0131, Latin Small Dotless I), not the normal "i" character.

Return Value

An `NSRange` structure giving the location and length in the receiver of *aString* within *aRange* in the receiver, modulo the options in *mask*. The range returned is relative to the start of the string, not to the passed-in range. Returns `{NSNotFound, 0}` if *aString* is not found or is empty (@"").

Discussion

The length of the returned range and that of *aString* may differ if equivalent composed character sequences are matched.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSString.h`

smallestEncoding

Returns the smallest encoding to which the receiver can be converted without loss of information.

```
- (NSStringEncoding)smallestEncoding
```

Return Value

The smallest encoding to which the receiver can be converted without loss of information.

Discussion

The returned encoding may not be the fastest for accessing characters, but is space-efficient. This method may take some time to execute.

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

See Also
- `fastestEncoding` (page 45)
- `getCharacters:range:` (page 48)

Declared In
`NSString.h`

stringByAbbreviatingWithTildeInPath

Returns a new string representing the receiver as a path with a tilde (~) substituted for the full path to the current user's home directory.

```
- (NSString *)stringByAbbreviatingWithTildeInPath
```

Return Value
A new string representing the receiver as a path with a tilde (~) substituted for the full path to the current user's home directory. Returns a new string matching the receiver if the receiver doesn't begin with a user's home directory.

Discussion
Note that this method only works with file paths (not, for example, string representations of URLs).

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

See Also
- `stringByExpandingTildeInPath` (page 89)

Related Sample Code
SimpleDownload

Declared In
`NSPathUtilities.h`

stringByAddingPercentEscapesUsingEncoding:

Returns a representation of the receiver using a given encoding to determine the percent escapes necessary to convert the receiver into a legal URL string.

```
- (NSString *)stringByAddingPercentEscapesUsingEncoding:(NSStringEncoding)encoding
```

Parameters
encoding
The encoding to use for the returned string.

Return Value
A representation of the receiver using *encoding* to determine the percent escapes necessary to convert the receiver into a legal URL string. Returns `nil` if *encoding* cannot encode a particular character

Discussion
See `CFURLCreateStringByAddingPercentEscapes` for more complex transformations.

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

See Also
`- stringByReplacingPercentEscapesUsingEncoding:` (page 93)

Declared In
`NSURL.h`

stringByAppendingFormat:

Returns a string made by appending to the receiver a string constructed from a given format string and the following arguments.

```
- (NSString *)stringByAppendingFormat:(NSString *)format ...
```

Parameters

format

A format string. See Formatting String Objects for more information. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *format* is `nil`.

...

A comma-separated list of arguments to substitute into *format*.

Return Value
A string made by appending to the receiver a string constructed from *format* and the following arguments, in the manner of `stringWithFormat:` (page 30).

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

See Also
`- stringByAppendingString:` (page 86)

Related Sample Code
Departments and Employees
QTMetadataEditor

Declared In
`NSString.h`

stringByAppendingPathComponent:

Returns a new string made by appending to the receiver a given string.

```
- (NSString *)stringByAppendingPathComponent:(NSString *)aString
```

Parameters

aString

The path component to append to the receiver.

Return Value

A new string made by appending *aString* to the receiver, preceded if necessary by a path separator.

Discussion

The following table illustrates the effect of this method on a variety of different paths, assuming that *aString* is supplied as "scratch.tiff":

Receiver's String Value	Resulting String
"/tmp"	"/tmp/scratch.tiff"
"/tmp/"	"/tmp/scratch.tiff"
"/"	"/scratch.tiff"
"" (an empty string)	"scratch.tiff"

Note that this method only works with file paths (not, for example, string representations of URLs).

Availability

Available in Mac OS X v10.0 and later.

See Also

- stringsByAppendingPaths: (page 96)
- stringByAppendingPathExtension: (page 85)
- stringByDeletingLastPathComponent (page 87)

Related Sample Code

Core Data HTML Store

CoreRecipes

Quartz Composer WWDC 2005 TextEdit

StickiesExample

TextEditPlus

Declared In

NSPathUtilities.h

stringByAppendingPathExtension:

Returns a new string made by appending to the receiver an extension separator followed by a given extension.

```
- (NSString *)stringByAppendingPathExtension:(NSString *)ext
```

Parameters

ext

The extension to append to the receiver.

Return Value

A new string made by appending to the receiver an extension separator followed by *ext*.

Discussion
The following table illustrates the effect of this method on a variety of different paths, assuming that *ext* is supplied as `@"tiff"`:

Receiver's String Value	Resulting String
"/tmp/scratch.old"	"/tmp/scratch.old.tiff"
"/tmp/scratch."	"/tmp/scratch..tiff"
"/tmp/"	"/tmp.tiff"
"scratch"	"scratch.tiff"

Note that adding an extension to `@"/tmp/"` causes the result to be `@"/tmp.tiff"` instead of `@"/tmp/.tiff"`. This difference is because a file named `@".tiff"` is not considered to have an extension, so the string is appended to the last nonempty path component.

This method does not allow you to append file extensions to filenames starting with the tilde character (~).

Note that this method only works with file paths (not, for example, string representations of URLs).

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

See Also
- `- stringByAppendingPathComponent:` (page 84)
- `- stringByDeletingPathExtension` (page 88)

Related Sample Code
QTRecorder
Quartz Composer WWDC 2005 TextEdit
SpotlightFortunes
TextEditPlus
WhackedTV

Declared In
`NSPathUtilities.h`

stringByAppendingString:

Returns a new string made by appending a given string to the receiver.

```
- (NSString *)stringByAppendingString:(NSString *)aString
```

Parameters

aString

The string to append to the receiver. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aString* is `nil`.

Return Value

A new string made by appending *aString* to the receiver.

Discussion

This code excerpt, for example:

```
NSString *errorTag = @"Error: ";
NSString *errorString = @"premature end of file.";
NSString *errorMessage = [errorTag stringByAppendingString:errorString];
```

produces the string "`Error: premature end of file.`".

Availability

Available in Mac OS X v10.0 and later.

See Also

`- stringByAppendingFormat:` (page 84)

Related Sample Code

CocoaDVDPlayer

NumberInput_IMKit_Sample

QTSSConnectionMonitor

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In

`NSString.h`

stringByDeletingLastPathComponent

Returns a new string made by deleting the last path component from the receiver, along with any final path separator.

```
- (NSString *)stringByDeletingLastPathComponent
```

Return Value

A new string made by deleting the last path component from the receiver, along with any final path separator. If the receiver represents the root path it is returned unaltered.

Discussion

The following table illustrates the effect of this method on a variety of different paths:

Receiver's String Value	Resulting String
"/tmp/scratch.tiff"	"/tmp"

Receiver's String Value	Resulting String
"/tmp/lock/"	"/tmp"
"/tmp/"	"/"
"/tmp"	"/"
"/"	"/"
"scratch.tiff"	"" (an empty string)

Note that this method only works with file paths (not, for example, string representations of URLs).

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

See Also
- `stringByDeletingPathExtension` (page 88)
- `stringByAppendingPathComponent:` (page 84)

Related Sample Code
ExtractMovieAudioToAIFF
LSMSmartCategorizer
Quartz Composer WWDC 2005 TextEdit
TextEditPlus
WhackedTV

Declared In
`NSPathUtilities.h`

stringByDeletingPathExtension

Returns a new string made by deleting the extension (if any, and only the last) from the receiver.

```
- (NSString *)stringByDeletingPathExtension
```

Return Value
a new string made by deleting the extension (if any, and only the last) from the receiver. Strips any trailing path separator before checking for an extension. If the receiver represents the root path, it is returned unaltered.

Discussion
The following table illustrates the effect of this method on a variety of different paths:

Receiver's String Value	Resulting String
"/tmp/scratch.tiff"	"/tmp/scratch"
"/tmp/"	"/tmp"
"scratch.bundle/"	"scratch"

Receiver's String Value	Resulting String
"scratch..tiff"	"scratch."
".tiff"	".tiff"
"/"	"/"

Note that attempting to delete an extension from @".tiff" causes the result to be @".tiff" instead of an empty string. This difference is because a file named @".tiff" is not considered to have an extension, so nothing is deleted. Note also that this method only works with file paths (not, for example, string representations of URLs).

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

See Also
- pathExtension (page 73)
- stringByDeletingLastPathComponent (page 87)

Related Sample Code
AutoUpdater
EnhancedAudioBurn
QTAudioExtractionPanel
Quartz Composer Offline Rendering
StickiesExample

Declared In
NSPathUtilities.h

stringByExpandingTildeInPath

Returns a new string made by expanding the initial component of the receiver to its full path value.

```
- (NSString *)stringByExpandingTildeInPath
```

Return Value
A new string made by expanding the initial component of the receiver, if it begins with "~" or "~user", to its full path value. Returns a new string matching the receiver if the receiver's initial component can't be expanded.

Discussion
Note that this method only works with file paths (not, for example, string representations of URLs).

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

See Also
- stringByAbbreviatingWithTildeInPath (page 83)

Related Sample Code
MyPhoto

Quartz Composer Offline Rendering
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus

Declared In
`NSPathUtilities.h`

stringByFoldingWithOptions:locale:

Returns a string with the given character folding options applied.

```
- (NSString *)stringByFoldingWithOptions:(NSStringCompareOptions)options
    locale:(NSLocale *)locale
```

Parameters

options

A mask of compare flags with a suffix `InsensitiveSearch`.

locale

The locale to use for the folding. The locale affects the folding logic. For example, for the Turkish locale, case-insensitive compare matches “I” to “ı” (Unicode code point U+0131, Latin Small Dotless I), not the normal “i” character.

Return Value
A string with the character folding options applied.

Discussion
Character folding operations remove distinctions between characters. For example, case folding may replace uppercase letters with their lowercase equivalents.

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

Declared In
`NSString.h`

stringByPaddingToLength:withString:startingAtIndex:

Returns a new string formed from the receiver by either removing characters from the end, or by appending as many occurrences as necessary of a given pad string.

```
- (NSString *)stringByPaddingToLength:(NSUInteger)newLength withString:(NSString
    *)padString startingAtIndex:(NSUInteger)padIndex
```

Parameters

newLength

The new length for the receiver.

padString

The string with which to extend the receiver.

padIndex

The index in *padString* from which to start padding.

Return Value
A new string formed from the receiver by either removing characters from the end, or by appending as many occurrences of *padString* as necessary.

Discussion
Here are some examples of usage:

```
[@"abc" stringByPaddingToLength: 9 withString: @"." startingAtIndex:0];
    // Results in "abc......"

[@"abc" stringByPaddingToLength: 2 withString: @"." startingAtIndex:0];
    // Results in "ab"

[@"abc" stringByPaddingToLength: 9 withString: @". " startingAtIndex:1];
    // Results in "abc . . ."
    // Notice that the first character in the padding is " "
```

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

Declared In
NSString.h

stringByReplacingCharactersInRange:withString:

Returns a new string in which the characters in a specified range of the receiver are replaced by a given string.

```
- (NSString *)stringByReplacingCharactersInRange:(NSRange)range withString:(NSString
    *)replacement
```

Parameters

range
A range of characters in the receiver.

replacement
The string with which to replace the characters in *range*.

Return Value
A new string in which the characters in *range* of the receiver are replaced by *replacement*.

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

See Also
- stringByReplacingOccurrencesOfString:withString: (page 92)
- stringByReplacingOccurrencesOfString:withString:options:range: (page 92)
- stringByReplacingPercentEscapesUsingEncoding: (page 93)

Declared In
NSString.h

stringByReplacingOccurrencesOfString:withString:

Returns a new string in which all occurrences of a target string in the receiver are replaced by another given string.

```
- (NSString *)stringByReplacingOccurrencesOfString:(NSString *)target
    withString:(NSString *)replacement
```

Parameters

target

The string to replace.

replacement

The string with which to replace *target*.

Return Value

A new string in which all occurrences of *target* in the receiver are replaced by *replacement*.

Discussion

Invokes `stringByReplacingOccurrencesOfString:withString:options:range:` (page 92)with `0` options and range of the whole string.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `stringByReplacingOccurrencesOfString:withString:options:range:` (page 92)
- `stringByReplacingCharactersInRange:withString:` (page 91)
- `stringByReplacingPercentEscapesUsingEncoding:` (page 93)

Declared In

`NSString.h`

stringByReplacingOccurrencesOfString:withString:options:range:

Returns a new string in which all occurrences of a target string in a specified range of the receiver are replaced by another given string.

```
- (NSString *)stringByReplacingOccurrencesOfString:(NSString *)target
    withString:(NSString *)replacement options:(NSStringCompareOptions)options
    range:(NSRange)searchRange
```

Parameters

target

The string to replace.

replacement

The string with which to replace *target*.

options

A mask of options to use when comparing *target* with the receiver. Pass `0` to specify no options.

searchRange

The range in the receiver in which to search for *target*.

Return Value
A new string in which all occurrences of *target*, matched using *options*, in *searchRange* of the receiver are replaced by *replacement*.

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

See Also
- stringByReplacingOccurrencesOfString:withString: (page 92)
- stringByReplacingCharactersInRange:withString: (page 91)
- stringByReplacingPercentEscapesUsingEncoding: (page 93)

Declared In
NSString.h

stringByReplacingPercentEscapesUsingEncoding:

Returns a new string made by replacing in the receiver all percent escapes with the matching characters as determined by a given encoding.

```
- (NSString *)stringByReplacingPercentEscapesUsingEncoding:(NSStringEncoding)encoding
```

Parameters
encoding
The encoding to use for the returned string.

Return Value
A new string made by replacing in the receiver all percent escapes with the matching characters as determined by the given encoding *encoding*. Returns `nil` if the transformation is not possible, for example, the percent escapes give a byte sequence not legal in *encoding*.

Discussion
See `CFURLCreateStringByReplacingPercentEscapes` for more complex transformations.

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

See Also
- stringByAddingPercentEscapesUsingEncoding: (page 83)

Declared In
NSURL.h

stringByResolvingSymlinksInPath

Returns a new string made from the receiver by resolving all symbolic links and standardizing path.

```
- (NSString *)stringByResolvingSymlinksInPath
```

Return Value
A new string made by expanding an initial tilde expression in the receiver, then resolving all symbolic links and references to current or parent directories if possible, to generate a standardized path. If the original path is absolute, all symbolic links are guaranteed to be removed; if it's a relative path, symbolic links that can't be resolved are left unresolved in the returned string. Returns `self` if an error occurs.

Discussion
If the name of the receiving path begins with `/private`, the `stringByResolvingSymlinksInPath` method strips off the `/private` designator, provided the result is the name of an existing file.

Note that this method only works with file paths (not, for example, string representations of URLs).

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

See Also
- `stringByStandardizingPath` (page 94)
- `stringByExpandingTildeInPath` (page 89)

Related Sample Code
CoreRecipes
DeskPictAppDockMenu
PredicateEditorSample
Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSPathUtilities.h`

stringByStandardizingPath

Returns a new string made by removing extraneous path components from the receiver.

```
- (NSString *)stringByStandardizingPath
```

Return Value
A new string made by removing extraneous path components from the receiver.

Discussion
If `stringByStandardizingPath` detects symbolic links in a pathname, the `stringByResolvingSymlinksInPath` (page 93) method is called to resolve them. If an invalid pathname is provided, `stringByStandardizingPath` may attempt to resolve it by calling `stringByResolvingSymlinksInPath`, and the results are undefined. If any other kind of error is encountered (such as a path component not existing), `self` is returned.

This method can make the following changes in the provided string:

- Expand an initial tilde expression using `stringByExpandingTildeInPath` (page 89).
- 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 93), which consults the file system to resolve each potential symbolic link.

  In relative paths, because symbolic links can't be resolved, references to the parent directory are left in place.
- Remove an initial component of "`/private`" from the path if the result still indicates an existing file or directory (checked by consulting the file system).

Note that the path returned by this method may still have symbolic link components in it. Note also that this method only works with file paths (not, for example, string representations of URLs).

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

See Also
- `stringByExpandingTildeInPath` (page 89)
- `stringByResolvingSymlinksInPath` (page 93)

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

Declared In
`NSPathUtilities.h`

stringByTrimmingCharactersInSet:

Returns a new string made by removing from both ends of the receiver characters contained in a given character set.

```
- (NSString *)stringByTrimmingCharactersInSet:(NSCharacterSet *)set
```

Parameters

set

A character set containing the characters to remove from the receiver. *set* must not be `nil`.

Return Value
A new string made by removing from both ends of the receiver characters contained in *set*. If the receiver is composed entirely of characters from *set*, the empty string is returned.

Discussion
Use `whitespaceCharacterSet` or `whitespaceAndNewlineCharacterSet` to remove whitespace around strings.

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

See Also
- `componentsSeparatedByCharactersInSet:` (page 40)

Related Sample Code
CoreRecipes
iSpend
TextLinks

Declared In
`NSString.h`

stringsByAppendingPaths:

Returns an array of strings made by separately appending to the receiver each string in in a given array.

```
- (NSArray *)stringsByAppendingPaths:(NSArray *)paths
```

Parameters

paths
An array of `NSString` objects specifying paths to add to the receiver.

Return Value
An array of `NSString` objects made by separately appending each string in *paths* to the receiver, preceded if necessary by a path separator.

Discussion
Note that this method only works with file paths (not, for example, string representations of URLs). See `stringByAppendingPathComponent:` (page 84) for an individual example.

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

Declared In
`NSPathUtilities.h`

substringFromIndex:

Returns a new string containing the characters of the receiver from the one at a given index to the end.

```
- (NSString *)substringFromIndex:(NSUInteger)anIndex
```

Parameters

anIndex
An index. The value must lie within the bounds of the receiver, or be equal to the length of the receiver.

Important: Raises an `NSRangeException` if *anIndex* lies beyond the end of the receiver.

Return Value
A new string containing the characters of the receiver from the one at *anIndex* to the end. If *anIndex* is equal to the length of the string, returns an empty string.

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

See Also
- substringWithRange: (page 98)
- substringToIndex: (page 97)

Related Sample Code
Birthdays
Core Data HTML Store
People
Reminders
Sketch-112

Declared In
NSString.h

substringToIndex:

Returns a new string containing the characters of the receiver up to, but not including, the one at a given index.

```
- (NSString *)substringToIndex:(NSUInteger)anIndex
```

Parameters
anIndex
An index. The value must lie within the bounds of the receiver, or be equal to the length of the receiver.

Important: Raises an NSRangeException if (*anIndex* - 1) lies beyond the end of the receiver.

Return Value
A new string containing the characters of the receiver up to, but not including, the one at *anIndex*. If *anIndex* is equal to the length of the string, returns a copy of the receiver.

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

See Also
- substringFromIndex: (page 96)
- substringWithRange: (page 98)

Related Sample Code
DerivedProperty
People
Quartz Composer WWDC 2005 TextEdit
StickiesExample
TextEditPlus

Declared In
NSString.h

substringWithRange:

Returns a string object containing the characters of the receiver that lie within a given range.

```
- (NSString *)substringWithRange:(NSRange)aRange
```

Parameters

aRange

A range. The range must not exceed the bounds of the receiver.

Important: Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the receiver.

Return Value
A string object containing the characters of the receiver that lie within *aRange*.

Discussion
This method treats the length of the string as a valid range value that returns an empty string.

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

See Also
- `substringFromIndex:` (page 96)
- `substringToIndex:` (page 97)

Related Sample Code
EnhancedDataBurn
iSpend
Quartz Composer WWDC 2005 TextEdit
TextEditPlus
VertexPerformanceTest

Declared In
`NSString.h`

uppercaseString

Returns an uppercased representation of the receiver.

```
- (NSString *)uppercaseString
```

Return Value
A string with each character from the receiver changed to its corresponding uppercase value.

Discussion
Case transformations aren't guaranteed to be symmetrical or to produce strings of the same lengths as the originals. See `lowercaseString` (page 70) for an example.

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

See Also
- `capitalizedString` (page 33)

- lowercaseString (page 70)

Related Sample Code
QTKitMovieShuffler
Worm

Declared In
NSString.h

UTF8String

Returns a null-terminated UTF8 representation of the receiver.

```
- (const char *)UTF8String
```

Return Value
A null-terminated UTF8 representation of the receiver.

Discussion
The returned C string is automatically freed just as a returned object would be released; you should copy the C string if it needs to store it outside of the autorelease context in which the C string is created.

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

Related Sample Code
DynamicProperties
NameAndPassword

Declared In
NSString.h

writeToFile:atomically:encoding:error:

Writes the contents of the receiver to a file at a given path using a given encoding.

```
- (BOOL)writeToFile:(NSString *)path atomically:(BOOL)useAuxiliaryFile
    encoding:(NSStringEncoding)enc error:(NSError **)error
```

Parameters

path

The file to which to write the receiver. If *path* contains a tilde (~) character, you must expand it with stringByExpandingTildeInPath (page 89) before invoking this method.

useAuxiliaryFile

If YES, the receiver is written to an auxiliary file, and then the auxiliary file is renamed to *path*. If NO, the receiver is written directly to *path*. The YES option guarantees that *path*, if it exists at all, won't be corrupted even if the system should crash during writing.

enc

The encoding to use for the output.

error

If there is an error, upon return contains an `NSError` object that describes the problem. If you are not interested in details of errors, you may pass in `NULL`.

Return Value
`YES` if the file is written successfully, otherwise `NO` (if there was a problem writing to the file or with the encoding).

Discussion
This method overwrites any existing file at *path*.

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

Declared In
`NSString.h`

writeToURL:atomically:encoding:error:

Writes the contents of the receiver to the URL specified by *url* using the specified encoding.

```
- (BOOL)writeToURL:(NSURL *)url atomically:(BOOL)useAuxiliaryFile
    encoding:(NSStringEncoding)enc error:(NSError **)error
```

Parameters

url

The URL to which to write the receiver.

useAuxiliaryFile

If `YES`, the receiver is written to an auxiliary file, and then the auxiliary file is renamed to *url*. If `NO`, the receiver is written directly to *url*. The `YES` option guarantees that *url*, if it exists at all, won't be corrupted even if the system should crash during writing.

The *useAuxiliaryFile* parameter is ignored if *url* is not of a type that can be accessed atomically.

enc

The encoding to use for the output.

error

If there is an error, upon return contains an `NSError` object that describes the problem. If you are not interested in details of errors, you may pass in `NULL`.

Return Value
`YES` if the URL is written successfully, otherwise `NO` (if there was a problem writing to the URL or with the encoding).

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

Declared In
`NSString.h`

Constants

unichar

Type for Unicode characters.

```
typedef unsigned short unichar;
```

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

Declared In
`NSString.h`

NSMaximumStringLength

A constant to define the maximum number of characters in an `NSString` object. (**Deprecated.** This constant is not available in Mac OS X v10.5 and later.)

```
#define NSMaximumStringLength (INT_MAX-1)
```

Constants
`NSMaximumStringLength`
Maximum number of characters in an `NSString` object.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `NSString.h`.

Availability
Available in Mac OS X v10.0.
Removed in Mac OS X v10.5.

Declared In
`NSString.h`

NSStringCompareOptions

Type for string comparison options.

```
typedef NSUInteger NSStringCompareOptions;
```

Discussion
See "Search and Comparison Options" (page 102) for possible values.

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

Declared In
`NSString.h`

Search and Comparison Options

These values represent the options available to many of the string classes' search and comparison methods.

```
enum {
    NSCaseInsensitiveSearch = 1,
    NSLiteralSearch = 2,
    NSBackwardsSearch = 4,
    NSAnchoredSearch = 8,
    NSNumericSearch = 64,
    NSDiacriticInsensitiveSearch = 128,
    NSWidthInsensitiveSearch = 256,
    NSForcedOrderingSearch = 512
};
```

Constants

`NSCaseInsensitiveSearch`

A case-insensitive search.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSLiteralSearch`

Exact character-by-character equivalence.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSBackwardsSearch`

Search from end of source string.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSAnchoredSearch`

Search is limited to start (or end, if `NSBackwardsSearch`) of source string.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSNumericSearch`

Numbers within strings are compared using numeric value, that is, `Foo2.txt` < `Foo7.txt` < `Foo25.txt`.

This option only applies to compare methods, not find.

Available in Mac OS X v10.3 and later.

Declared in `NSString.h`.

`NSDiacriticInsensitiveSearch`

Search ignores diacritic marks.

For example, 'ö' is equal to 'o'.

Available in Mac OS X v10.5 and later.

Declared in `NSString.h`.

`NSWidthInsensitiveSearch`
Search ignores width differences in characters that have full-width and half-width forms, as occurs in East Asian character sets.

For example, with this option, the full-width Latin small letter 'a' (Unicode code point U+FF41) is equal to the basic Latin small letter 'a' (Unicode code point U+0061).

Available in Mac OS X v10.5 and later.

Declared in `NSString.h`.

`NSForcedOrderingSearch`
Comparisons are forced to return either `NSOrderedAscending` or `NSOrderedDescending` if the strings are equivalent but not strictly equal.

This option gives stability when sorting. For example, “aaa” is greater than "AAA” if `NSCaseInsensitiveSearch` is specified.

Available in Mac OS X v10.5 and later.

Declared in `NSString.h`.

Discussion
See Searching, Comparing, and Sorting Strings for details on the effects of these options.

Declared In
`NSString.h`

NSStringEncodingConversionOptions

Type for encoding conversion options.

```
typedef NSUInteger NSStringEncodingConversionOptions;
```

Discussion
See `NSStringEncodingConversionOptions` (page 103) for possible values.

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

Declared In
`NSString.h`

Encoding Conversion Options

Options for converting string encodings.

```
enum {
    NSStringEncodingConversionAllowLossy = 1,
    NSStringEncodingConversionExternalRepresentation = 2
};
```

Constants
`NSStringEncodingConversionAllowLossy`
Allows lossy conversion.

Available in Mac OS X v10.5 and later.

Declared in `NSString.h`.

`NSStringEncodingConversionExternalRepresentation`

Available in Mac OS X v10.5 and later.

Declared in `NSString.h`.

Special Considerations

These constants are available in Mac OS X v10.4; they are, however, differently named:

```
typedef enum {
    NSAllowLossyEncodingConversion = 1,
    NSExternalRepresentationEncodingConversion = 2
} NSStringEncodingConversionOptions;
```

You can use them on Mac OS X v10.4 if you define the symbols as `extern` constants.

Declared In
`NSString.h`

NSString Handling Exception Names

These constants define the names of exceptions raised if `NSString` cannot represent a string in a given encoding, or parse a string as a property list.

```
extern NSString *NSParseErrorException;
extern NSString *NSCharacterConversionException;
```

Constants

`NSCharacterConversionException`

`NSString` raises an `NSCharacterConversionException` if a string cannot be represented in a file-system or string encoding.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSParseErrorException`

`NSString` raises an `NSParseErrorException` if a string cannot be parsed as a property list.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

Declared In
`NSString.h`

NSStringEncoding

Type for string encoding.

```
typedef NSUInteger NSStringEncoding;
```

Discussion
See "String Encodings" (page 105) for possible values.

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

Declared In
NSString.h

String Encodings

The following constants are provided by NSString as possible string encodings.

```
enum {
    NSASCIIStringEncoding = 1,
    NSNEXTSTEPStringEncoding = 2,
    NSJapaneseEUCStringEncoding = 3,
    NSUTF8StringEncoding = 4,
    NSISOLatin1StringEncoding = 5,
    NSSymbolStringEncoding = 6,
    NSNonLossyASCIIStringEncoding = 7,
    NSShiftJISStringEncoding = 8,
    NSISOLatin2StringEncoding = 9,
    NSUnicodeStringEncoding = 10,
    NSWindowsCP1251StringEncoding = 11,
    NSWindowsCP1252StringEncoding = 12,
    NSWindowsCP1253StringEncoding = 13,
    NSWindowsCP1254StringEncoding = 14,
    NSWindowsCP1250StringEncoding = 15,
    NSISO2022JPStringEncoding = 21,
    NSMacOSRomanStringEncoding = 30,
    NSUTF16BigEndianStringEncoding = 0x90000100,
    NSUTF16LittleEndianStringEncoding = 0x94000100,
    NSUTF32StringEncoding = 0x8c000100,
    NSUTF32BigEndianStringEncoding = 0x98000100,
    NSUTF32LittleEndianStringEncoding = 0x9c000100,
    NSProprietaryStringEncoding = 65536
};
```

Constants

NSASCIIStringEncoding
Strict 7-bit ASCII encoding within 8-bit chars; ASCII values 0…127 only.
Available in Mac OS X v10.0 and later.
Declared in NSString.h.

NSISO2022JPStringEncoding
ISO 2022 Japanese encoding for email.
Available in Mac OS X v10.0 and later.
Declared in NSString.h.

NSISOLatin1StringEncoding
8-bit ISO Latin 1 encoding.
Available in Mac OS X v10.0 and later.
Declared in NSString.h.

NSISOLatin2StringEncoding
8-bit ISO Latin 2 encoding.
Available in Mac OS X v10.0 and later.
Declared in NSString.h.

`NSJapaneseEUCStringEncoding`
8-bit EUC encoding for Japanese text.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSMacOSRomanStringEncoding`
Classic Macintosh Roman encoding.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSNEXTSTEPStringEncoding`
8-bit ASCII encoding with NEXTSTEP extensions.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSNonLossyASCIIStringEncoding`
7-bit verbose ASCII to represent all Unicode characters.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSShiftJISStringEncoding`
8-bit Shift-JIS encoding for Japanese text.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSSymbolStringEncoding`
8-bit Adobe Symbol encoding vector.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSUTF8StringEncoding`
An 8-bit representation of Unicode characters, suitable for transmission or storage by ASCII-based systems.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSUnicodeStringEncoding`
The canonical Unicode encoding for string objects.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSWindowsCP1250StringEncoding`
Microsoft Windows codepage 1250; equivalent to WinLatin2.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSWindowsCP1251StringEncoding`
Microsoft Windows codepage 1251, encoding Cyrillic characters; equivalent to AdobeStandardCyrillic font encoding.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSWindowsCP1252StringEncoding`
Microsoft Windows codepage 1252; equivalent to WinLatin1.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSWindowsCP1253StringEncoding`
Microsoft Windows codepage 1253, encoding Greek characters.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSWindowsCP1254StringEncoding`
Microsoft Windows codepage 1254, encoding Turkish characters.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

`NSUTF16BigEndianStringEncoding`
`NSUTF16StringEncoding` encoding with explicit endianness specified.

Available in Mac OS X v10.5 and later.

Declared in `NSString.h`.

`NSUTF16LittleEndianStringEncoding`
`NSUTF16StringEncoding` encoding with explicit endianness specified.

Available in Mac OS X v10.5 and later.

Declared in `NSString.h`.

`NSUTF32StringEncoding`
32-bit UTF encoding.

Available in Mac OS X v10.5 and later.

Declared in `NSString.h`.

`NSUTF32BigEndianStringEncoding`
`NSUTF32StringEncoding` encoding with explicit endianness specified.

Available in Mac OS X v10.5 and later.

Declared in `NSString.h`.

`NSUTF32LittleEndianStringEncoding`
`NSUTF32StringEncoding` encoding with explicit endianness specified.

Available in Mac OS X v10.5 and later.

Declared in `NSString.h`.

`NSProprietaryStringEncoding`
Installation-specific encoding.

Available in Mac OS X v10.0 and later.

Declared in `NSString.h`.

Discussion
These values represent the various character encodings supported by the `NSString` classes. This is an incomplete list. Additional encodings are defined in *Strings Programming Guide for Core Foundation* (see `CFStringEncodingExt.h`); these encodings can be used with `NSString` by first passing the Core Foundation encoding to the `CFStringConvertEncodingToNSStringEncoding` function.

Declared In
`NSString.h`

Deprecated NSString Methods

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

Deprecated in Mac OS X v10.4

stringWithContentsOfFile:

Returns a string created by reading data from the file named by a given path. (Deprecated in Mac OS X v10.4. Use `stringWithContentsOfFile:encoding:error:` (page 26) or stringWithContentsOfFile:usedEncoding:error: (page 27) instead.)

```
+ (id)stringWithContentsOfFile:(NSString *)path
```

Discussion
If the contents begin with a Unicode byte-order mark (`U+FEFF` or `U+FFFE`), interprets the contents as Unicode characters. If the contents begin with a UTF-8 byte-order mark (`EFBBBF`), interprets the contents as UTF-8. Otherwise, interprets the contents as data in the default C string encoding. Since the default C string encoding will vary with the user's configuration, do not depend on this method unless you are using Unicode or UTF-8 or you can verify the default C string encoding. Returns `nil` if the file can't be opened.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
`+ stringWithContentsOfFile:encoding:error:` (page 26)
`+ stringWithContentsOfFile:usedEncoding:error:` (page 27)

Related Sample Code
CIAnnotation
GLSLShowpiece
NURBSSurfaceVertexProg
SurfaceVertexProgram
Vertex Optimization

Declared In
`NSString.h`

stringWithContentsOfURL:

Returns a string created by reading data from the file named by a given URL. (Deprecated in Mac OS X v10.4. Use `stringWithContentsOfURL:encoding:error:` (page 28) or `stringWithContentsOfURL:usedEncoding:error:` (page 28) instead.)

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

Discussion
If the contents begin with a byte-order mark (U+FEFF or U+FFFE), interprets the contents as Unicode characters. If the contents begin with a UTF-8 byte-order mark (EFBBBF), interprets the contents as UTF-8. Otherwise interprets the contents as data in the default C string encoding. Since the default C string encoding will vary with the user's configuration, do not depend on this method unless you are using Unicode or UTF-8 or you can verify the default C string encoding. Returns `nil` if the location can't be opened.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
`+ stringWithContentsOfURL:encoding:error:` (page 28)
`+ stringWithContentsOfURL:usedEncoding:error:` (page 28)

Declared In
`NSString.h`

stringWithCString:

Creates a new string using a given C-string. (Deprecated in Mac OS X v10.4. Use `stringWithCString:encoding:` (page 29) instead.)

```
+ (id)stringWithCString:(const char *)cString
```

Discussion
`cString` should contain data in the default C string encoding. If the argument passed to `stringWithCString:` is not a zero-terminated C-string, the results are undefined.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
`+ stringWithCString:encoding:` (page 29)

Related Sample Code
Quartz EB
Simon
SurfaceVertexProgram
Vertex Optimization
Video Hardware Info

Declared In
`NSString.h`

stringWithCString:length:

Returns a string containing the characters in a given C-string. (Deprecated in Mac OS X v10.4. Use `stringWithCString:encoding:` (page 29) instead.)

```
+ (id)stringWithCString:(const char *)cString length:(NSUInteger)length
```

Discussion
cString must not be NULL. *cString* should contain characters in the default C-string encoding. This method converts *length* * `sizeof(char)` bytes from *cString* and doesn't stop short at a `NULL` character.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
`+ stringWithCString:encoding:` (page 29)

Related Sample Code
CapabilitiesSample
CocoaSpeechSynthesisExample
EnhancedDataBurn
Fiendishthngs
SGDevices

Declared In
`NSString.h`

cString

Returns a representation of the receiver as a C string in the default C-string encoding. (Deprecated in Mac OS X v10.4. Use `cStringUsingEncoding:` (page 41) or `UTF8String` (page 99) instead.)

```
- (const char *)cString
```

Discussion
The returned C string will be automatically freed just as a returned object would be released; your code should copy the C string or use `getCString:` (page 112) if it needs to store the C string outside of the autorelease context in which the C string is created.

Raises an `NSCharacterConversionException` if the receiver can't be represented in the default C-string encoding without loss of information. Use `canBeConvertedToEncoding:` (page 32) if necessary to check whether a string can be losslessly converted to the default C-string encoding. If it can't, use `lossyCString` (page 117) or `dataUsingEncoding:allowLossyConversion:` (page 42) to get a C-string representation with some loss of information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
`- cStringUsingEncoding:` (page 41)
`- getCString:maxLength:encoding:` (page 48)

- UTF8String (page 99)

Related Sample Code
WhackedTV

Declared In
NSString.h

cStringLength

Returns the length in char-sized units of the receiver's C-string representation in the default C-string encoding. (Deprecated in Mac OS X v10.4. Use lengthOfBytesUsingEncoding: (page 68) or maximumLengthOfBytesUsingEncoding: (page 71) instead.)

```
- (NSUInteger)cStringLength
```

Discussion
Raises if the receiver can't be represented in the default C-string encoding without loss of information. You can also use canBeConvertedToEncoding: (page 32) to check whether a string can be losslessly converted to the default C-string encoding. If it can't, use lossyCString (page 117) to get a C-string representation with some loss of information, then check its length explicitly using the ANSI function strlen().

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
- lengthOfBytesUsingEncoding: (page 68)
- maximumLengthOfBytesUsingEncoding: (page 71)
- UTF8String (page 99)

Declared In
NSString.h

getCString:

Invokes getCString:maxLength:range:remainingRange: (page 114) with NSMaximumStringLength as the maximum length, the receiver's entire extent as the range, and NULL for the remaining range. (Deprecated in Mac OS X v10.4. Use cStringUsingEncoding: (page 41) or dataUsingEncoding:allowLossyConversion: (page 42) instead.)

```
- (void)getCString:(char *)buffer
```

Discussion
buffer must be large enough to contain the resulting C-string plus a terminating NULL character (which this method adds—[string cStringLength]).

Raises an NSCharacterConversionException if the receiver can't be represented in the default C-string encoding without loss of information. Use canBeConvertedToEncoding: (page 32) if necessary to check whether a string can be losslessly converted to the default C-string encoding. If it can't, use lossyCString (page 117) or dataUsingEncoding:allowLossyConversion: (page 42) to get a C-string representation with some loss of information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
- `cStringUsingEncoding:` (page 41)
- `getCString:maxLength:encoding:` (page 48)
- `UTF8String` (page 99)

Related Sample Code
QTMetadataEditor
ThreadsExporter
ThreadsImporter
ThreadsImportMovie

Declared In
`NSString.h`

getCString:maxLength:

Invokes `getCString:maxLength:range:remainingRange:` (page 114) with *`maxLength`* as the maximum length in `char`-sized units, the receiver's entire extent as the range, and `NULL` for the remaining range. (Deprecated in Mac OS X v10.4. Use getCString:maxLength:encoding: (page 48) instead.)

```
- (void)getCString:(char *)buffer maxLength:(NSUInteger)maxLength
```

Discussion
`buffer` must be large enough to contain *`maxLength`* `char`s plus a terminating zero `char` (which this method adds).

Raises an `NSCharacterConversionException` if the receiver can't be represented in the default C-string encoding without loss of information. Use `canBeConvertedToEncoding:` (page 32) if necessary to check whether a string can be losslessly converted to the default C-string encoding. If it can't, use `lossyCString` (page 117) or `dataUsingEncoding:allowLossyConversion:` (page 42) to get a C-string representation with some loss of information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
- `cStringUsingEncoding:` (page 41)
- `getCString:maxLength:encoding:` (page 48)
- `UTF8String` (page 99)

Declared In
`NSString.h`

getCString:maxLength:range:remainingRange:

Converts the receiver's content to the default C-string encoding and stores them in a given buffer. (Deprecated in Mac OS X v10.4. Use getCString:maxLength:encoding: (page 48) instead.)

```
- (void)getCString:(char *)buffer maxLength:(NSUInteger)maxLength
    range:(NSRange)aRange remainingRange:(NSRangePointer)leftoverRange
```

Discussion
buffer must be large enough to contain *maxLength* bytes plus a terminating zero character (which this method adds). Copies and converts as many characters as possible from *aRange* and stores the range of those not converted in the range given by *leftoverRange* (if it's non-nil). Raises an NSRangeException if any part of *aRange* lies beyond the end of the string.

Raises an NSCharacterConversionException if the receiver can't be represented in the default C-string encoding without loss of information. Use canBeConvertedToEncoding: (page 32) if necessary to check whether a string can be losslessly converted to the default C-string encoding. If it can't, use lossyCString (page 117) or dataUsingEncoding:allowLossyConversion: (page 42) to get a C-string representation with some loss of information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
- cStringUsingEncoding: (page 41)
- getCString:maxLength:encoding: (page 48)
- UTF8String (page 99)

Declared In
NSString.h

initWithContentsOfFile:

Initializes the receiver, a newly allocated NSString object, by reading data from the file named by *path*. (Deprecated in Mac OS X v10.4. Use initWithContentsOfFile:encoding:error: (page 56) or initWithContentsOfFile:usedEncoding:error: (page 57) instead.)

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

Discussion
Initializes the receiver, a newly allocated NSString object, by reading data from the file named by *path*. If the contents begin with a byte-order mark (U+FEFF or U+FFFE), interprets the contents as Unicode characters; otherwise interprets the contents as data in the default C string encoding. Returns an initialized object, which might be different from the original receiver, or nil if the file can't be opened.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
- initWithContentsOfFile:encoding:error: (page 56)
- initWithContentsOfFile:usedEncoding:error: (page 57)

Declared In
NSString.h

initWithContentsOfURL:

Initializes the receiver, a newly allocated NSString object, by reading data from the location named by a given URL. (Deprecated in Mac OS X v10.4. Use initWithContentsOfURL:encoding:error: (page 57) or initWithContentsOfURL:usedEncoding:error: (page 58) instead.)

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

Discussion
Initializes the receiver, a newly allocated NSString object, by reading data from the location named by *aURL*. If the contents begin with a byte-order mark (U+FEFF or U+FFFE), interprets the contents as Unicode characters; otherwise interprets the contents as data in the default C string encoding. Returns an initialized object, which might be different from the original receiver, or nil if the location can't be opened.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
- initWithContentsOfURL:encoding:error: (page 57)
- initWithContentsOfURL:usedEncoding:error: (page 58)

Declared In
NSString.h

initWithCString:

Initializes the receiver, a newly allocated NSString object, by converting the data in a given C-string from the default C-string encoding into the Unicode character encoding. (Deprecated in Mac OS X v10.4. Use initWithCString:encoding: (page 58) instead.)

```
- (id)initWithCString:(const char *)cString
```

Discussion
cString must be a zero-terminated C string in the default C string encoding, and may not be NULL. Returns an initialized object, which might be different from the original receiver.

To create an immutable string from an immutable C string buffer, do not attempt to use this method. Instead, use initWithCStringNoCopy:length:freeWhenDone: (page 116).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
- initWithCString:encoding: (page 58)

Related Sample Code
SimplePlayThru

Declared In
NSString.h

initWithCString:length:

Initializes the receiver, a newly allocated NSString object, by converting the data in a given C-string from the default C-string encoding into the Unicode character encoding. (Deprecated in Mac OS X v10.4. Use initWithCString:encoding: (page 58) instead.)

```
- (id)initWithCString:(const char *)cString length:(NSUInteger)length
```

Discussion
This method converts *length* * sizeof(char) bytes from *cString* and doesn't stop short at a zero character. *cString* must contain bytes in the default C-string encoding and may not be NULL. Returns an initialized object, which might be different from the original receiver.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
- initWithCString:encoding: (page 58)

Related Sample Code
CocoaSpeechSynthesisExample

Declared In
NSString.h

initWithCStringNoCopy:length:freeWhenDone:

Initializes the receiver, a newly allocated NSString object, by converting the data in a given C-string from the default C-string encoding into the Unicode character encoding. (Deprecated in Mac OS X v10.4. Use initWithBytesNoCopy:length:encoding:freeWhenDone: (page 54) instead.)

```
- (id)initWithCStringNoCopy:(char *)cString length:(NSUInteger)length
    freeWhenDone:(BOOL)flag
```

Discussion
This method converts *length* * sizeof(char) bytes from *cString* and doesn't stop short at a zero character. *cString* must contain data in the default C-string encoding and may not be NULL. The receiver becomes the owner of *cString*; if *flag* is YES it will free the memory when it no longer needs it, but if *flag* is NO it won't. Returns an initialized object, which might be different from the original receiver.

You can use this method to create an immutable string from an immutable (const char *) C-string buffer. If you receive a warning message, you can disregard it; its purpose is simply to warn you that the C string passed as the method's first argument may be modified. If you make certain the freeWhenDone argument to initWithStringNoCopy is NO, the C string passed as the method's first argument cannot be modified, so you can safely use initWithStringNoCopy to create an immutable string from an immutable (const char *) C-string buffer.

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

Deprecated in Mac OS X v10.4.

See Also
- initWithCString:encoding: (page 58)

Declared In
NSString.h

lossyCString

Returns a representation of the receiver as a C string in the default C-string encoding, possibly losing information in converting to that encoding. (Deprecated in Mac OS X v10.4. Use cStringUsingEncoding: (page 41) or dataUsingEncoding:allowLossyConversion: (page 42) instead.)

```
- (const char *)lossyCString
```

Discussion
This method does not raise an exception if the conversion is lossy. The returned C string will be automatically freed just as a returned object would be released; your code should copy the C string or use getCString: (page 112) if it needs to store the C string outside of the autorelease context in which the C string is created.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
- cStringUsingEncoding: (page 41)
- dataUsingEncoding:allowLossyConversion: (page 42)

Declared In
NSString.h

writeToFile:atomically:

Writes the contents of the receiver to the file specified by a given path. (Deprecated in Mac OS X v10.4. Use writeToFile:atomically:encoding:error: (page 99) instead.)

```
- (BOOL)writeToFile:(NSString *)path atomically:(BOOL)flag
```

Return Value
YES if the file is written successfully, otherwise NO.

Discussion
Writes the contents of the receiver to the file specified by *path* (overwriting any existing file at *path*). *path* is written in the default C-string encoding if possible (that is, if no information would be lost), in the Unicode encoding otherwise.

If *flag* is YES, the receiver is written to an auxiliary file, and then the auxiliary file is renamed to *path*. If *flag* is NO, the receiver is written directly to *path*. The YES option guarantees that *path*, if it exists at all, won't be corrupted even if the system should crash during writing.

If *path* contains a tilde (~) character, you must expand it with `stringByExpandingTildeInPath` (page 89) before invoking this method.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
`- writeToFile:atomically:encoding:error:` (page 99)

Related Sample Code
bMoviePalette
bMoviePaletteCocoa
Cropped Image
Monochrome Image
RGB Image

Declared In
`NSString.h`

writeToURL:atomically:

Writes the contents of the receiver to the location specified by a given URL. (Deprecated in Mac OS X v10.4. Use `writeToURL:atomically:encoding:error:` (page 100) instead.)

```
- (BOOL)writeToURL:(NSURL *)aURL atomically:(BOOL)atomically
```

Return Value
`YES` if the location is written successfully, otherwise `NO`.

Discussion
If *atomically* is `YES`, the receiver is written to an auxiliary location, and then the auxiliary location is renamed to *aURL*. If *atomically* is `NO`, the receiver is written directly to *aURL*. The `YES` option guarantees that *aURL*, if it exists at all, won't be corrupted even if the system should crash during writing.

The *atomically* parameter is ignored if *aURL* is not of a type that can be accessed atomically.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

See Also
`- writeToURL:atomically:encoding:error:` (page 100)

Declared In
`NSString.h`

Document Revision History

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

Date	Notes
2009-02-04	Changed the abstract of getCharacters: to more closely reflect that of getCharacters:range:.
2008-10-15	Corrected return value of longLongValue. Rephrased the abstracts of all the rangeOfCharacter... and rangeOfString... methods, and corrected name mismatches in their parameter descriptions.
	Rewrote description for the constant `NSWidthInsensitiveSearch` (page 103). Fixed bad cross reference in `NSStringEncodingConversionOptions` (page 103) discussion. Added explanations of the effect of locale option on string comparison operations of `compare:options:range:locale:` (page 38), `rangeOfString:options:range:locale:` (page 81), and `stringByFoldingWithOptions:locale:` (page 90).
2008-03-11	Added paragraph to introduction describing the byte-order assumptions of factory and initialization methods taking UTF-16 input.
	Clarified descriptions of `getLineStart:end:contentsEnd:forRange:` (page 50), `getParagraphStart:end:contentsEnd:forRange:` (page 51), `componentsSeparatedByCharactersInSet:` (page 40), and `componentsSeparatedByString:` (page 40).
2008-02-08	Corrected a typographical error.
2007-10-31	Clarified the effect of the stringByStandardizingPath method.
2007-08-23	Added discussion notes regarding comparison of strings to be presented to the user.
2007-07-19	Augmented the description of required buffer sizes in getCharacters: and getCString:. Added a warning about passing NULL to stringWithUTF8String: and initWithUTF8String:.
2007-03-06	Corrected discussion of initWithCharacters methods; clarified behavior of noCopy: creation methods on failure.
2007-02-08	Enhanced the parameter descriptions for several methods.
2006-11-07	Warned not to pass NULL into stringWithUTF8String:.
2006-10-03	Augmented the description of required buffer sizes in the getCharacters: and getCString: methods.

Date	Notes
2006-06-28	Clarified the return value of dataUsingEncoding:.
2006-05-23	Incorporated constants from Foundation Constants article.

Index

H

I

L

M

N

P

R

S

U

W