NSString

The NSString class and its mutable subclass, [NSMutableString](https://developer.apple.com/library/prerelease/mac/documentation/Cocoa/Reference/Foundation/Classes/NSMutableString_Class/index.html#//apple_ref/occ/cl/NSMutableString), provide an extensive set of APIs for working with strings, including methods for comparing, searching, and modifying strings. NSString objects are used extensively throughout Foundation and other Cocoa frameworks, serving as the basis for all textual and linguistic functionality on the platform.

NSString is “toll-free bridged” with its Core Foundation counterpart, [CFStringRef](https://developer.apple.com/library/prerelease/mac/documentation/CoreFoundation/Reference/CFStringRef/index.html" \l "//apple_ref/c/tdef/CFStringRef). See “[Toll-Free Bridging](https://developer.apple.com/library/prerelease/mac/documentation/General/Conceptual/CocoaEncyclopedia/Toll-FreeBridgin/Toll-FreeBridgin.html#//apple_ref/doc/uid/TP40010810-CH2)” for more information.

### String Objects

An NSString object encodes a Unicode-compliant text string, represented as a sequence of UTF–16 code units. All lengths, character indexes, and ranges are expressed in terms of 16-bit platform-endian values, with index values starting at 0.

An NSString object can be initialized from or written to a C buffer, an NSData object, or the contents of an NSURL. It can also be encoded and decoded to and from ASCII, UTF–8, UTF–16, UTF–32, or any other string encoding represented by [NSStringEncoding](https://developer.apple.com/library/prerelease/mac/documentation/Cocoa/Reference/Foundation/Classes/NSString_Class/#//apple_ref/c/tdef/NSStringEncoding).

**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 UTF–16 code units (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 a sequence of UTF–16 code units. You can determine how many UTF-16 code units a string object contains with the length method and can retrieve a specific UTF-16 code unit with the characterAtIndex: 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.

Localized string comparisons are based on the Unicode Collation Algorithm, as tailored for different languages by CLDR (Common Locale Data Repository). Both are projects of the Unicode Consortium. Unicode is a registered trademark of Unicode, Inc.

**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 unichar values, the returned string is always native-endian, since the array always contains UTF–16 code units in native byte order.

**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).

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). 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 and characterAtIndex:. 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: 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.

**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.

**Creating and Initializing Strings**

+ string

- init

Designated Initializer

- initWithBytes:length:encoding:

- initWithBytesNoCopy:length:encoding:freeWhenDone:

- initWithCharacters:length:

- initWithCharactersNoCopy:length:freeWhenDone:

- initWithString:

- initWithCString:encoding:

- initWithUTF8String:

- initWithFormat:

- initWithFormat:arguments:

- initWithFormat:locale:

- initWithFormat:locale:arguments:

- initWithData:encoding:

+ stringWithFormat:

+ localizedStringWithFormat:

+ stringWithCharacters:length:

+ stringWithString:

+ stringWithCString:encoding:

+ stringWithUTF8String:

+ stringWithCString:

(OS X v10.4)

- initWithCString:

(OS X v10.4)

+ stringWithCString:length:

(OS X v10.4)

**Creating and Initializing a String from an URL**

+ stringWithContentsOfURL:encoding:error:

- initWithContentsOfURL:encoding:error:

+ stringWithContentsOfURL:usedEncoding:error:

- initWithContentsOfURL:usedEncoding:error:

+ stringWithContentsOfURL:

(OS X v10.4)

- initWithContentsOfURL:

(OS X v10.4)

**Getting a String’s Length**

length

Property

- lengthOfBytesUsingEncoding:

- maximumLengthOfBytesUsingEncoding:

**Getting Characters and Bytes**

- characterAtIndex:

- getCharacters:

(OS X v10.6)

- getCharacters:range:

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

**Getting C Strings**

- cStringUsingEncoding:

- getCString:maxLength:encoding:

UTF8String

Property

**Combining Strings**

- stringByAppendingFormat:

- stringByAppendingString:

- stringByPaddingToLength:withString:startingAtIndex:

**Dividing Strings**

- componentsSeparatedByString:

- componentsSeparatedByCharactersInSet:

- stringByTrimmingCharactersInSet:

- substringFromIndex:

- substringWithRange:

- substringToIndex:

**Finding Characters and Substrings**

- containsString:

- localizedCaseInsensitiveContainsString:

- localizedStandardContainsString:

- localizedStandardRangeOfString:

- rangeOfCharacterFromSet:

- rangeOfCharacterFromSet:options:

- rangeOfCharacterFromSet:options:range:

- rangeOfString:

- rangeOfString:options:

- rangeOfString:options:range:

- rangeOfString:options:range:locale:

- enumerateLinesUsingBlock:

- enumerateSubstringsInRange:options:usingBlock:

**Replacing Substrings**

- stringByReplacingOccurrencesOfString:withString:

- stringByReplacingOccurrencesOfString:withString:options:range:

- stringByReplacingCharactersInRange:withString:

**Getting a Shared Prefix**

- commonPrefixWithString:options:

**Changing Case**

capitalizedString

Property

- capitalizedStringWithLocale:

lowercaseString

Property

- lowercaseStringWithLocale:

uppercaseString

Property

- uppercaseStringWithLocale:

**Working with Encodings**

+ availableStringEncodings

+ defaultCStringEncoding

+ stringEncodingForData:encodingOptions:convertedString:usedLossyConversion:

+ localizedNameOfStringEncoding:

- canBeConvertedToEncoding:

- dataUsingEncoding:

- dataUsingEncoding:allowLossyConversion:

description

Property

fastestEncoding

Property

smallestEncoding

Property

**Data Types**

unichar

NSStringCompareOptions

NSStringEncodingConversionOptions

NSStringEncoding

**Inherits From:**

[NSObject](https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSObject_Class/index.html#//apple_ref/occ/cl/NSObject)

NSString

[NSMutableString](https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSMutableString_Class/index.html#//apple_ref/occ/cl/NSMutableString)

**Import Statement:**

* @import Foundation;

**Availability:**

* Available in OS X v10.0 and later

**Advantages:**

* It stores any type of character.
* It has many built in methods.

**Disadvantage:**

* It is immutable.

**References:**

* <https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSArray_Class/index.html#//apple_ref/occ/cl/NSArray>

NSMutableString

The NSMutableString class declares the programmatic interface to an object that manages a mutable string—that is, a string whose contents can be edited—that conceptually represents an array of Unicode characters. To construct and manage an immutable string—or a string that cannot be changed after it has been created—use an object of the [NSString](https://developer.apple.com/library/prerelease/mac/documentation/Cocoa/Reference/Foundation/Classes/NSString_Class/index.html#//apple_ref/occ/cl/NSString) class.

The NSMutableString class adds one primitive method—[replaceCharactersInRange:withString:](https://developer.apple.com/library/prerelease/mac/documentation/Cocoa/Reference/Foundation/Classes/NSMutableString_Class/index.html" \l "//apple_ref/occ/instm/NSMutableString/replaceCharactersInRange:withString:)—to the basic string-handling behavior inherited from NSString. All other methods that modify a string work through this method. For example, [insertString:atIndex:](https://developer.apple.com/library/prerelease/mac/documentation/Cocoa/Reference/Foundation/Classes/NSMutableString_Class/index.html" \l "//apple_ref/occ/instm/NSMutableString/insertString:atIndex:) simply replaces the characters in a range of 0 length, while[deleteCharactersInRange:](https://developer.apple.com/library/prerelease/mac/documentation/Cocoa/Reference/Foundation/Classes/NSMutableString_Class/index.html#//apple_ref/occ/instm/NSMutableString/deleteCharactersInRange:) replaces the characters in a given range with no characters.

NSMutableString is “toll-free bridged” with its Core Foundation counterpart, [CFMutableStringRef](https://developer.apple.com/library/prerelease/mac/documentation/CoreFoundation/Reference/CFMutableStringRef/index.html" \l "//apple_ref/c/tdef/CFMutableStringRef). See [Toll-Free Bridging](https://developer.apple.com/library/prerelease/mac/documentation/General/Conceptual/CocoaEncyclopedia/Toll-FreeBridgin/Toll-FreeBridgin.html#//apple_ref/doc/uid/TP40010810-CH2) for more information.

**Creating and Initializing a Mutable String**

+ stringWithCapacity:

- initWithCapacity:

**Modifying a String**

- appendFormat:

- appendString:

- applyTransform:reverse:range:updatedRange:

- deleteCharactersInRange:

- insertString:atIndex:

- replaceCharactersInRange:withString:

- replaceOccurrencesOfString:withString:options:range:

- setString:

**Inherits From:**

[NSObject](https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSObject_Class/index.html#//apple_ref/occ/cl/NSObject)

NSString

[NSMutableString](https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSMutableString_Class/index.html#//apple_ref/occ/cl/NSMutableString)

**Import Statement:**

* @import Foundation;

**Availability:**

* Available in OS X v10.0 and later

**Advantages:**

* It stores any type of character.
* It has many built in methods.

**Disadvantage:**

* Any subclass of NSString must override the primitive instance methods [length](https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSString_Class/index.html#//apple_ref/occ/instp/NSString/length) and [characterAtIndex:](https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSString_Class/index.html#//apple_ref/occ/instm/NSString/characterAtIndex:)

**References:**

* <https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSArray_Class/index.html#//apple_ref/occ/cl/NSArray>