# NSCharacterSet Class Cluster

## Class Cluster Description

An NSCharacterSet object represents a set of Unicode characters. NSString and NSScanner objects use NSCharacterSets to group characters together for searching operations, so that they can find any of a particular set of characters during a search. The cluster's two public classes, NSCharacterSet and NSMutableCharacterSet, declare the programmatic interface for static and dynamic character sets, respectively.

The objects you create using these classes are referred to as *character set objects* (and when no confusion will result, merely as *character sets*). Because of the nature of class clusters, character set objects aren't actual instances of the NSCharacterSet or NSMutableCharacterSet classes but of one of their private subclasses. Although a character set object's class is private, its interface is public, as declared by these abstract superclasses, NSCharacterSet and NSMutableCharacterSet. (See "Class Clusters" in the introduction to the Foundation Kit for more information on class clusters and on creating subclasses within a cluster.) The character set classes adopt the NSCopying and NSMutableCopying protocols, making it convenient to convert a character set of one type to the other.

### Building a Character Set

NSCharacterSet defines class methods that return commonly used character sets, such as letters (uppercase or lowercase), decimal digits, whitespace, and so on. These "standard" character sets are always immutable, even if created by sending a message to NSMutableCharacterSet. See "Standard Character Sets and Unicode Definitions" below for more information on standard character sets.

You can use a standard character set as a starting point for building a custom set by making a mutable copy of it and changing that. (You can also start from scratch by creating a mutable character set with **alloc** and **init** and adding characters to it.) For example, this fragment creates a character set containing letters, digits, and basic punctuation:

```
NSMutableCharacterSet *workingSet;
NSCharacterSet *finalCharSet;

workingSet = [[NSCharacterSet alphanumericCharacterSet] mutableCopy];
[workingSet addCharactersInString:@";:,."];
finalCharSet = [workingSet copy];
[workingSet release];
```

For performance reasons (explained in "Using a Character Set"), always finish by converting the working mutable character set into an immutable set. If you need to keep changing the character set after you've created it, of course, you should just use the mutable set.

If your application frequently uses a custom character set, you'll want to save its definition in a resource file and load that instead of explicitly adding individual characters each time you need to create the set. You can save a character set by getting its bitmap representation (an NSData object) and saving that object to a file:

```
NSString *filename;    /* Assume this exists. */
NSString *absolutePath;
NSData *charSetRep;
BOOL result;

absolutePath = [filename stringByStandardizingPath];
charSetRep = [finalCharSet bitmapRepresentation];
result = [charSetRep writeToFile:absolutePath atomically:YES];
```

Character set filenames by convention use the extension **.bitmap**. If you intend for others to use your character set files, you should follow this convention. To read a character set file with a **.bitmap** extension, simply use the **characterSetWithContentsOfFile:** method.

## Using a Character Set

A character set object doesn't perform any tasks; it simply holds a set of character values to limit operations on strings. The NSString and NSScanner classes define methods that take NSCharacterSets as arguments to find any of several characters. For example, this code excerpt finds the range of the first uppercase letter in **myString**:

```
NSString *myString = @"some text in an NSString...";
NSRange letterRange;

letterRange = [myString rangeOfCharacterFromSet:[NSCharacterSet
    uppercaseLetterCharacterSet]];
```

After this fragment executes, **letterRange.location** is equal to the index of the first "N" in "NSString" after **rangeOfCharacterFromSet:** is invoked. If the first letter of the string were "S" then **letterRange.location** would be 0.

Because character sets often participate in performance-critical code, you should be aware of the aspects of their use that can affect the performance of your application. Mutable character sets are generally much more expensive than immutable character sets. They consume more memory and are costly to invert (an operation often performed in scanning a string). Because of this, you should follow these guidelines:

- Create as few mutable character sets as possible.
- Cache character sets (in a global dictionary, perhaps) instead of continually recreating them.
- When creating a custom set that doesn't need to change after creation, make an immutable copy of the final character set for actual use, and dispose of the working mutable character set. Alternatively, create a character set file as described in "Building a Character Set" and store it in your application's main bundle.

- Similarly, avoid archiving character set objects; store them in character set files instead. Archiving can result in a character set being duplicated in different archive files, resulting in wasted disk space and duplicates in memory for each separate archive read.

## Standard Character Sets and Unicode Definitions

The standard character sets, such as that returned by **letterCharacterSet**, are formally defined in terms of the normative and informative categories established by the Unicode standard, such as Uppercase Letter, Combining Mark, and so on. The formal definition of a standard character set is in most cases given as one or more of the categories defined in the standard. For example, the set returned by **lowercaseLetterCharacterSet** include all characters in normative category Lowercase Letters, while the set returned by **letterCharacterSet** includes the characters in all of the Letter categories.

Note that the definitions of the categories themselves may change with new versions of the Unicode standard. You can download the files that define category membership from **http://www.unicode.org/**.

# NSCharacterSet

| | |
|---|---|
| **Inherits From:** | NSObject |
| **Conforms To:** | NSCoding<br>NSCopying<br>NSMutableCopying<br>NSObject (NSObject) |
| **Declared In:** | Foundation/NSCharacterSet.h |

## Class Description

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

## Adopted Protocols

| | |
|---|---|
| NSCoding | – encodeWithCoder:<br>– initWithCoder: |
| NSCopying | – copyWithZone: |
| NSMutableCopying | – mutableCopyWithZone: |

## Method Types

| | |
|---|---|
| Creating a standard character set | + alphanumericCharacterSet<br>+ controlCharacterSet<br>+ decimalDigitCharacterSet<br>+ decomposableCharacterSet<br>+ illegalCharacterSet<br>+ letterCharacterSet<br>+ lowercaseLetterCharacterSet<br>+ nonBaseCharacterSet<br>+ punctuationCharacterSet<br>+ uppercaseLetterCharacterSet<br>+ whitespaceAndNewlineCharacterSet<br>+ whitespaceCharacterSet |
| Creating a custom character set | + characterSetWithRange:<br>+ characterSetWithCharactersInString:<br>+ characterSetWithBitmapRepresentation: |
| Opening a character set file | + characterSetWithContentsOfFile: |
| Testing set membership | – characterIsMember: |
| Inverting a character set | – invertedSet |
| Getting a binary representation | – bitmapRepresentation |

## Class Methods

### alphanumericCharacterSet

+ (NSCharacterSet *)**alphanumericCharacterSet**

Returns a character set containing the characters in the categories Letters, Marks, and Numbers. Informally, this is the set of all characters used as basic units of alphabets, syllabaries, ideographs, and digits.

**See also:** + **letterCharacterSet**, + **decimalDigitCharacterSet**

### characterSetWithBitmapRepresentation:

+ (NSCharacterSet *)**characterSetWithBitmapRepresentation:**(NSData *)*data*

Returns a character set containing characters determined by the bitmap representation *data*. This method is useful for creating a character set object with data from a file or other external data source.

A raw bitmap representation of a character set is a byte array of $2^{16}$ bits (that is, 8192 bytes). The value of the bit at position *n* represents the presence in the character set of the character with decimal Unicode value *n*. To add a character with decimal Unicode value *n* to a raw bitmap representation, use a statement such as:

```
unsigned char bitmapRep[8192];

bitmapRep[n >> 3] |= (((unsigned int)1) << (n & 7));
```

To remove that character:

```
bitmapRep[n >> 3] &= ~(((unsigned int)1) << (n & 7));
```

**See also:** – **bitmapRepresentation**, + **characterSetWithContentsOfFile:**

### characterSetWithCharactersInString:

+ (NSCharacterSet *)**characterSetWithCharactersInString:**(NSString *)*aString*

Returns a character set containing the characters in *aString*. Returns an empty character set if *aString* is empty.

### NS+ characterSetWithContentsOfFile:

+ (NSCharacterSet *)**characterSetWithContentsOfFile:**(NSString *)*path*

Returns a character set read from the bitmap representation stored in the file at *path*, which must end with the extension **.bitmap**. To read a bitmap representation from any file, use NSData's **dataWithContentsOfFile:** method and pass the result to **characterSetWithBitmapRepresentation:**.

This method doesn't perform filename-based uniquing of the character sets it creates. To prevent duplication of character sets in memory, cache them and make them available through an API that checks whether the requested set has already been loaded.

### characterSetWithRange:

+ (NSCharacterSet *)**characterSetWithRange:**(NSRange)*aRange*

Returns a character set containing characters whose Unicode values are given by *aRange*. *aRange.location* is the value of the first character, and *aRange.location* + *aRange.length* – 1 is the value of the last. Returns an empty character set if *aRange.length* is 0.

This code excerpt creates a character set object containing the lowercase English alphabetic characters:

```
NSRange lcEnglishRange;
NSCharacterSet *lcEnglishLetters;
```

```
lcEnglishRange.location = (unsigned int)'a';
lcEnglishRange.length = 26;
lcEnglishLetters = [NSCharacterSet
    characterSetWithRange:lcEnglishRange];
```

## controlCharacterSet

+ (NSCharacterSet *)**controlCharacterSet**

Returns a character set containing the characters in the categories of Control or Format Characters. These are specifically the Unicode values U+0000 to U+001F and U+007F to U+009F.

**See also:** + **illegalCharacterSet**

## decimalDigitCharacterSet

+ (NSCharacterSet *)**decimalDigitCharacterSet**

Returns a character set containing the characters in the category of Decimal Numbers. Informally, this is the set of all characters used to represent the decimal values 0 through 9. These include, for example, the decimal digits of the Indic scripts and Arabic.

**See also:** + **alphanumericCharacterSet**

## decomposableCharacterSet

+ (NSCharacterSet *)**decomposableCharacterSet**

Returns a character set containing all individual Unicode characters that can also be represented as composed character sequences (such as for letters with accents), by the definition of "standard decomposition" in version 1.1 of the Unicode character encoding standard. These include compatibility characters as well as precomposed characters.

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

**See also:** + **nonBaseCharacterSet**

## illegalCharacterSet

+ (NSCharacterSet *)**illegalCharacterSet**

Returns a character set containing values in the category of Non-Characters, or that have not yet been defined in version 2.0 of the Unicode standard.

**See also:** + **controlCharacterSet**

## letterCharacterSet

+ (NSCharacterSet *)**letterCharacterSet**

Returns a character set containing the characters in the categories Letters and Marks. Informally, this is the set of all characters used as letters of alphabets and ideographs.

**See also:** + **alphanumericCharacterSet**, + **lowercaseLetterCharacterSet**, + **uppercaseLetterCharacterSet**

## lowercaseLetterCharacterSet

+ (NSCharacterSet *)**lowercaseLetterCharacterSet**

Returns a character set containing the characters in the category of Lowercase Letters. Informally, this is the set of all characters used as lowercase letters in alphabets which make case distinctions.

**See also:** + **uppercaseLetterCharacterSet**, + **letterCharacterSet**

## nonBaseCharacterSet

+ (NSCharacterSet *)**nonBaseCharacterSet**

Returns a character set containing the characters in the category of Marks. This set is also defined as all legal Unicode characters with a non-spacing priority greater than zero. Informally, this is the set of all characters used as modifiers of base characters.

**See also:** + **decomposableCharacterSet**

## punctuationCharacterSet

+ (NSCharacterSet *)**punctuationCharacterSet**

Returns a character set containing the characters in the category of Punctuation. Informally, this is the set of all non-whitespace characters used to separate linguistic units in scripts, such as periods, dashes, parentheses, and so on.

### uppercaseLetterCharacterSet

+ (NSCharacterSet *)**uppercaseLetterCharacterSet**

Returns a character set containing the characters in the category of Uppercase Letters. Informally, this is the set of all characters used as uppercase letters in alphabets which make case distinctions.

**See also:** + **lowercaseLetterCharacterSet**, + **letterCharacterSet**

### whitespaceAndNewlineCharacterSet

+ (NSCharacterSet *)**whitespaceAndNewlineCharacterSet**

Returns a character set containing only the whitespace characters space (U+0020) and tab (U+0009) and the newline character (U+000A).

**See also:** + **whitespaceCharacterSet**

### whitespaceCharacterSet

+ (NSCharacterSet *)**whitespaceCharacterSet**

Returns a character set containing only the in-line whitespace characters space (U+0020) and tab (U+0009). This set doesn't contain the newline or carriage return characters.

**See also:** + **whitespaceAndNewlineCharacterSet**

## Instance Methods

### characterIsMember:

– (BOOL)**characterIsMember:**(unichar)*aCharacter*

Returns YES if *aCharacter* is in the receiving character set, NO if it isn't.

### bitmapRepresentation

– (NSData *)**bitmapRepresentation**

Returns an NSData object encoding the receiving character set in binary format. This format is suitable for saving to a file or otherwise transmitting or archiving.

A raw bitmap representation of a character set is a byte array of $2^{16}$ bits (that is, 8192 bytes). The value of the bit at position *n* represents the presence in the character set of the character with decimal Unicode value

*n*. To test for the presence of a character with decimal Unicode value *n* in a raw bitmap representation, use an expression such as:

```
unsigned char bitmapRep[8192];

if (bitmapRep[n >> 3] & (((unsigned int)1) << (n & 7))) {
    /* Character is present. */
}
```

**See also:** + **characterSetWithBitmapRepresentation:**

## invertedSet

– (NSCharacterSet *)**invertedSet**

Returns a character set containing only characters that *don't* exist in the receiver. Inverting an immutable character set is much more efficient that inverting a mutable character set.

**See also:** – **invert** (NSMutableCharacterSet)

# NSMutableCharacterSet

| | |
|---|---|
| **Inherits From:** | NSCharacterSet : NSObject |
| **Conforms To:** | NSCoding (NSCharacterSet)<br>NSCopying (NSCharacterSet)<br>NSMutableCopying (NSCharacterSet)<br>NSObject (NSObject) |
| **Declared In:** | Foundation/NSCharacterSet.h |

## Class Description

The NSMutableCharacterSet class declares the programmatic interface to objects that manage a modifiable set of Unicode characters. You can add or remove characters from a mutable character set as numeric values in NSRanges or as character values in strings; combine character sets by union or intersection; and invert a character set.

Mutable character sets are less efficient to use than immutable character sets. If you don't need to change a character set after creating it, create an immutable copy with **copy** and use that.

NSMutableCharacterSet defines no primitive methods. Subclasses must implement all methods declared by this class in addition to the primitives of NSCharacterSet. They must also implement **mutableCopyWithZone:**.

## Method Types

| | |
|---|---|
| Adding and removing characters | – addCharactersInRange:<br>– removeCharactersInRange:<br>– addCharactersInString:<br>– removeCharactersInString: |
| Combining character sets | – formIntersectionWithCharacterSet:<br>– formUnionWithCharacterSet: |
| Inverting a character set | – invert |

## Instance Methods

### addCharactersInRange:

– (void)**addCharactersInRange:**(NSRange)*aRange*

Adds the characters whose integer values are given by *aRange* to the receiver. *aRange.location* is the value of the first character to add, *aRange range.location* + *aRange.length* – 1 is the value of the last. If *aRange.length* is 0 this method has no effect.

**See also:** – **removeCharactersInRange:**, – **addCharactersInString:**

### addCharactersInString:

– (void)**addCharactersInString:**(NSString *)*aString*

Adds the characters in *aString* to those in the receiver. This method has no effect if *aString* is empty.

**See also:** – **removeCharactersInString:**, – **addCharactersInRange:**

### formIntersectionWithCharacterSet:

– (void)**formIntersectionWithCharacterSet:**(NSCharacterSet *)*otherSet*

Modifies the receiver so that it contains only characters that exist in both the receiver and in *otherSet*.

**See also:** – **formUnionWithCharacterSet:**

### formUnionWithCharacterSet:

– (void)**formUnionWithCharacterSet:**(NSCharacterSet *)*otherSet*

Modifies the receiver so that it contains all characters that exist in either the receiver or *otherSet*.

**See also:** – **formIntersectionWithCharacterSet:**

### invert

– (void)**invert**

Replaces all of the characters in the receiver with all the characters it didn’t previously contain. Inverting a mutable character set, whether by **invert** or by **invertedSet**, is much less efficient that inverting an immutable character set with **invertedSet**.

**See also:** – **invertedSet** (NSCharacterSet)

### removeCharactersInRange:

– (void)**removeCharactersInRange:**(NSRange)*aRange*

Removes from the receiver the characters whose integer values are given by *aRange*. *aRange.location* is the value of the first character to remove, and *aRange.location* + *aRange.length* – 1 is the value of the last. If *aRange.length* is 0 this method has no effect.

**See also:** – **addCharactersInRange:**, – **removeCharactersInString:**

### removeCharactersInString:

– (void)**removeCharactersInString:**(NSString *)*aString*

Removes the characters in *aString* from those in the receiver. This method has no effect if *aString* is empty.

**See also:** – **addCharactersInString:**, – **removeCharactersInRange:**