NSCharacterSet Class Cluster

Class Cluster Description

An NSCharacterSet object represents a set of Unicode characters. The NSString and NSScanner classes 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 are not 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 ªClass Clustersº in the introduction to the Foundation Kit for more information on class clusters and 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.

Using a Character Set

Character set objects are value objects, in that they don't perform any tasks. The NSString and NSScanner classes define methods that take NSCharacterSets as arguments so that they can 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]];
```

letterRange.**location** is equal to the index of the first ªNº in ªNSStringº after **rangeOfCharacterFromSet:** is invoked. If the first letter of the string were ªSº then **letterRange.location** would be 0.

See the NSScanner class cluster specification for an example using an NSScanner.

Building a Character Set

NSCharacterSet provides methods to quickly create ªstandardº character sets, such as letters (uppercase or lowercase), decimal digits, whitespace, and so on. You can use a standard character set as a starting point for building your own custom set by creating an immutable standard set and making a mutable copy of it. For example, to create a character set containing letters, decimal digits, and basic punctuation, you could use this code:

```
myCharSet = [[NSCharacterSet alphanumericCharacterSet] mutableCopy];
[myCharSet addCharactersInString:@":;,.";];
```

You can also start from scratch by using **alloc** and **init** to create an empty character 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 = @"/some/file";
NSData *charSetRep = [myCharSet bitmapRepresentation];
[charSetRep writeToFile:filename atomically:YES];
```

To read a character set file, load it into an NSData object and use **characterSetWithBitmapRepresentation:**:

```
charSetRep = [NSData dataWithContentsOfFile:filename];
myCharSet = [NSCharacterSet
    characterSetWithBitmapRepresentation:charSetRep];
```

Notes on Unicode Support

The NSCharacterSet classes don't fully support Unicode at this time. Only the low 256 character values, corresponding to the NEXTSTEP character set, are implemented. The definitions of the standard character sets defined by NSCharacterSet will change in the future to include the full set of Unicode characters. String objects created from C strings work properly with character set objects as they're currently implemented, and both will continue to work as NEXTSTEP support for the Unicode character encoding increases.

NSCharacterSet

Inherits From:	NSObject
Conforms To:	NSCopying NSMutableCopying
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 two primitive methodsÐ**characterIsMember:** and **bitmapRepresentation**Ðprovide the basis for all other instance methods in its interface. A subclass of NSCharacterSet needs only to override these methods for proper behavior.

Adopted Protocols

NSCopying	- copyWithZone: - copy
NSMutableCopying	- mutableCopyWithZone: - mutableCopy

Method Types

Creating a standard character set

+ alphanumericCharacterSet
+ controlCharacterSet
+ decimalDigitCharacterSet
+ decomposableCharacterSet
+ illegalCharacterSet
+ letterCharacterSet
+ lowercaseLetterCharacterSet

	+ nonBaseCharacterSet + uppercaseLetterCharacterSet + whitespaceCharacterSet + whitespaceAndNewlineCharacterSet
Creating a custom character set	+ characterSetWithRange: + characterSetWithCharactersInString: + characterSetWithBitmapRepresentation:
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 uppercase and lowercase NEXTSTEP alphabetic characters (a-z, A-Z, other alphabetic characters such as Ý, ‰, Û, ‡, and so on) and the decimal digit characters (0-9).

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.

See also: - **bitmapRepresentation**

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. If *aRange.length* is 0, an empty character set is returned.

For example, this code excerpt creates a character set object containing the lowercase English alphabetic characters:

```
NSCharacterSet *lcLetters;

lcLetters = [NSCharacterSet
    characterSetWithRange:(NSRange){(unsigned int) 'a', 26}];
```

characterSetWithCharactersInString:

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

Returns a character set containing the characters in *aString*. If *aString* is empty, an empty character set is returned. *aString* must not be **nil**.

controlCharacterSet

+ (NSCharacterSet *)**controlCharacterSet**

Returns a character set containing the control characters (characters with decimal Unicode values 0 to 31 and 127 to 159).

decimalDigitCharacterSet

+ (NSCharacterSet *)**decimalDigitCharacterSet**

Returns a character set containing only decimal digit characters (0-9).

See also: - **alphanumericCharacterSet**

decomposableCharacterSet

+ (NSCharacterSet *)**decomposableCharacterSet**

Returns a character set containing all individual Unicode characters that can also be represented as composed character sequences. Composed character sequences are simply letters with accents for the currently supported subset of Unicode (decimal values 0 through 255). See the NSString class cluster description for a brief introduction to composed character sequences.

See also: - **nonBaseCharacterSet**

illegalCharacterSet

+ (NSCharacterSet *)**illegalCharacterSet**

Returns a character set containing the illegal Unicode values. See *The Unicode Standard: Worldwide Character Encoding* for details on illegal Unicode values.

letterCharacterSet

+ (NSCharacterSet *)**letterCharacterSet**

Returns a character set containing the uppercase and lowercase NEXTSTEP alphabetic characters (a-z, A-Z, other alphabetic characters such as Ý, ‰, Û, ‡, and so on).

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

lowercaseLetterCharacterSet

+ (NSCharacterSet *)**lowercaseLetterCharacterSet**

Returns a character set containing only lowercase NEXTSTEP alphabetic characters (a-z, other alphabetic characters such as Ý, Û, and so on).

See also: - **uppercaseLetterCharacterSet**, - **letterCharacterSet**

nonBaseCharacterSet

+ (NSCharacterSet *)**nonBaseCharacterSet**

Returns an empty character set. There are no non-base characters in the subset of Unicode currently supported.

See also: - **decomposableCharacterSet**

uppercaseLetterCharacterSet

+ (NSCharacterSet *)**uppercaseLetterCharacterSet**

Returns a character set containing only uppercase NEXTSTEP alphabetic characters (A-Z, other alphabetic characters such as ‰, ‡, and so on).

See also: - **lowercaseLetterCharacterSet**, - **letterCharacterSet**

whitespaceAndNewlineCharacterSet

+ (NSCharacterSet *)**whitespaceAndNewlineCharacterSet**

Returns a character set containing only whitespace characters (space and tab) and the newline character.

See also: - **whitespaceCharacterSet**

whitespaceCharacterSet

+ (NSCharacterSet *)**whitespaceCharacterSet**

Returns a character set containing only in-line whitespace characters (space and tab). 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 bitmap representation is an byte array of 2^{16} bits (that is, 8192 bytes). The value of the bit at position 2^n represents the presence of the character with decimal Unicode value *n*. To add a character with decimal Unicode value *n* to a bitmap representation, use a statement such as:

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

To remove that character:

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

To test for the presence of that character, use an expression such as:

```
(bitmapRep[n >> 3] & (((unsigned)1) << (n & 7)))
```

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:	NSCopying (NSCharacterSet) NSMutableCopying (NSCharacterSet)
Declared In:	foundation/NSCharacterSet.h

Class Description

The NSMutableCharacterSet class declares the programmatic interface to objects that manage a modifiable set of Unicode characters. NSMutableCharacterSet defines no primitive methods; subclasses must override all methods declared by this class.

Adopted Protocols

NSCopying	- copyWithZone: - copy
NSMutableCopying	- mutableCopyWithZone: - mutableCopy

Method Types

Adding and removing characters	- addCharactersInRange: - removeCharactersInRange: - addCharactersInString: - removeCharactersInString:
Combining character sets	- formIntersectionWithCharacterSet: - 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, and *aRange.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. If *aString* is empty, this method has no effect. *aString* must not be **nil**.

See also: - **removeCharactersInString:**, - **addCharactersInRange:**

formIntersectionWithCharacterSet:

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

Modifies the receiver so that it contains only those 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*, barring duplicates.

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 is much less efficient that inverting an immutable character set.

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 add, 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. If *aString* is empty, this method has no effect. *aString* must not be **nil**.

See also: - **addCharactersInString:**, - **removeCharactersInRange:**