NSScanner Class Cluster

Class Cluster Description

An NSScanner object interprets and converts the characters of an NSString into number and string values. You assign the scanner's string on creating it, and the scanner progresses through the characters of that string from beginning to end as you request items. This cluster has a single public class, NSScanner.

The objects you create using this class are referred to as *scanner objects* (and when no confusion will result, merely as *scanners*). Because of the nature of class clusters, scanner objects aren't actual instances of the NSScanner class but one of its private subclasses. Although a scanner object's class is private, its interface is public, as declared by this abstract superclass, NSScanner. (See "Class Clusters" in the introduction to the Foundation Kit for more information on class clusters and on creating subclasses within a cluster.)

Scanning Behavior

Generally, you instantiate a scanner object by invoking the **scannerWithString:** or **localizedScannerWithString**: class methods. Either method returns a scanner object initialized with the string you pass to it. The newly created scanner starts at the beginning of its string, progressing through the characters as you request values with **scan...** methods. You can change the implicit scan location with the **setScanLocation:** method, to re-scan a portion of the string after an error or to skip ahead a certain number of characters. Scan operations start at the scan location and advance the scanner to just past the last character in the scanned value representation (if any). For example, after scanning an integer from the string "137 small cases of bananas", a scanner's location will be 3, indicating the space immediately after the number.

You can configure a scanner to skip a set of characters with the **setCharactersToBeSkipped:** method. A scanner ignores characters to be skipped at the beginning of any scan operation. Once it finds a scannable character, however, it includes all characters matching the request. Scanners skip whitespace and newline characters by default. If you continue with the previous example's string and use **scanUpToString:intoString:** to find the substring before "of", the scanner skips the space character before the word "small" but includes the space before "of" in its result unless you include a space in the search string:

Search String	Result String
"of" (no space before)	"small cases " (includes the space following)
" of" (space before)	"small cases" (stops before the space)

You can also configure a scanner to consider or ignore case using the **setCaseSensitive:** method. By default a scanner ignores case. Note that case is always considered with regard to characters to be skipped. To skip all English vowels, for example, you must set the characters to be skipped to those in the string "AEIOUaeiou".

A scanner bases some of its scanning behavior on a locale, which specifies a language and conventions for value representations. NSScanner uses only the locale's definition for the decimal separator (given by the key named NSDecimalSeparator). You can create a scanner with the user's locale by using **localizedScannerWithString:**, or set the locale explicitly using **setLocale:**. If you use a method that doesn't specify a locale, the scanner assumes the default locale values. See "Locales" in the "Other Features" section of the Foundation Kit documentation for more information on locales.

For an example of using a scanner, suppose you have a string containing lines such as:

> Product: Acme Potato Peeler; Cost: 0.98
> Product: Chef Pierre Pasta Fork; Cost: 0.75
> Product: Chef Pierre Colander; Cost: 1.27

This method scans such a string to extract the product information for each line:

```
- (BOOL)scanProductString:(NSString *)string
{
    NSCharacterSet *semicolonSet;
    NSScanner *theScanner;
    NSString *PRODUCT = @"Product:";
    NSString *COST = @"Cost:";
    NSString *productName;
    float productCost;

    semicolonSet = [NSCharacterSet
        characterSetWithCharactersInString:@";"];
    theScanner = [NSScanner scannerWithString:string];

    while ([theScanner isAtEnd] == NO) {

        if ([theScanner scanString:PRODUCT intoString:NULL] &&
            [theScanner scanUpToCharactersFromSet:semicolonSet
                        intoString:&productName] &&
            [theScanner scanString:@";" intoString:NULL] &&
            [theScanner scanString:COST intoString:NULL] &&
            [theScanner scanFloat:&productCost]) {

            /* Do something with productName and productCost. */
        }
        else return NO;
    }

    return YES;
}
```

This method uses alternating scan operations to skip the expected substrings "Product:" and "Cost:", as well as the semicolon, and to read the values for the product name and cost (read as a **float** for simplicity's sake). It returns NO if an error occurs on any scan operation, and YES if it successfully scans and processes all lines. Note that because a scanner skips whitespace and newlines by default, the loop does no special processing for them.

NSScanner

Inherits From:	NSObject
Conforms To:	NSCopying NSObject (NSObject)
Declared In:	Foundation/NSScanner.h Foundation/NSDecimalNumber.h

Class Description

The NSScanner class declares the programmatic interface for an object that scans values from an NSString object. NSScanner's primitive methods are **string** and all of the methods listed below under "Configuring an NSScanner." Using an NSScanner is explained in the class cluster description.

Adopted Protocols

NSCopying	– copyWithZone:

Method Types

Creating an NSScanner	+ scannerWithString: + localizedScannerWithString – initWithString:
Getting an NSScanner's string	– string
Configuring an NSScanner	– setScanLocation: – scanLocation – setCaseSensitive: – caseSensitive – setCharactersToBeSkipped: – charactersToBeSkipped – setLocale: – locale

Scanning a string	– scanCharactersFromSet:intoString: – scanUpToCharactersFromSet:intoString: – scanDecimal: – scanDouble: – scanFloat: – scanInt: – scanHexInt: – scanLongLong: – scanString:intoString: – scanUpToString:intoString: – isAtEnd

Class Methods

localizedScannerWithString:

+ (id)**localizedScannerWithString:**(NSString *)*aString*

Returns an NSScanner that scans *aString* according to the user’s default locale (set with **setLocale:**). Sets the string to scan by invoking **initWithString:** with *aString*. See “Locales” in the “Other Features” section of the Foundation Kit documentation for more information on locales.

scannerWithString:

+ (id)**scannerWithString:**(NSString *)*aString*

Returns an NSScanner that scans *aString*. Sets the string to scan by invoking **initWithString:** with *aString*.

Instance Methods

caseSensitive

– (BOOL)**caseSensitive**

Returns YES if the scanner distinguishes case in the characters it scans, NO otherwise. NSScanners are *not* case sensitive by default. Note that case sensitivity doesn’t apply to the characters to be skipped.

See also: – **setCaseSensitive:**, – **setCharactersToBeSkipped:**

charactersToBeSkipped

– (NSCharacterSet *)**charactersToBeSkipped**

Returns a character set containing the characters that the scanner ignores when looking for a scannable element. For example, if a scanner ignores spaces and you send it a **scanInt:** message, it skips spaces until it finds a decimal digit or other character. While an element is being scanned, however, no characters are skipped. If you scan for something made of characters in the set to be skipped (for example, using **scanInt:** when the set of characters to be skipped is the decimal digits), the result is undefined.

The default set to skip is the whitespace and newline character set.

See also: – **setCharactersToBeSkipped:**, + **whitespaceAndNewlineCharacterSet** (NSCharacterSet)

initWithString:

– (id)**initWithString:**(NSString *)*aString*

Initializes a newly allocated NSScanner to scan *aString* from the beginning. Returns **self**.

See also: + **localizedScannerWithString:**, + **scannerWithString:**

isAtEnd

– (BOOL)**isAtEnd**

Returns YES if the scanner has exhausted all significant characters in its string, NO if there are characters left to scan. If only characters from the set to be skipped remain, this method returns YES.

See also: – **charactersToBeSkipped**

locale

– (NSDictionary *)**locale**

Returns the scanner's locale, or **nil** if it has none. A scanner's locale affects the way it interprets numeric values from the string. In particular, a scanner uses the locale's decimal separator to distinguish the integer and fractional parts of floating-point representations. A scanner with no locale set uses the default locale values.

See "Locales" in the "Other Features" section of the Foundation Kit documentation for more information on locales.

See also: – **setLocale:**

scanCharactersFromSet:intoString:

– (BOOL)**scanCharactersFromSet:**(NSCharacterSet *)*scanSet*
  intoString:(NSString **)*stringValue*

Scans the string as long as characters from *scanSet* are encountered, accumulating characters into a string that's returned by reference in *stringValue*. Returns YES if the scanner scans any characters; otherwise returns NO.

Invoke this method with NULL as *stringValue* to simply scan past a given set of characters.

See also: – **scanUpToCharactersFromSet:intoString:**

NS+ scanDecimal:

– (BOOL)**scanDecimal:**(NSDecimal *)*decimalValue*

Scans an NSDecimal value if possible, returning it by reference in *decimalValue*. Returns YES if the scanner finds a valid NSDecimal representation, NO otherwise. See the NSDecimalNumber class specification for more information.

Invoke this method with NULL as *decimalValue* to simply scan past an NSDecimal representation.

scanDouble:

– (BOOL)**scanDouble:**(double *)*doubleValue*

Scans a **double** value if possible, returning it by reference in *doubleValue*. Returns YES if the scanner finds a valid floating-point representation, NO otherwise. Returns HUGE_VAL or –HUGE_VAL by reference in *value* on overflow, or 0.0 on underflow. Skips past excess digits in the case of overflow, so that the scanner's position is past the entire floating-point representation.

Invoke this method with NULL as *doubleValue* to simply scan past a **double** value representation.

Note: Floating-point representations are assumed to be IEEE compliant.

See also: – **doubleValue** (NSString)

scanFloat:

– (BOOL)**scanFloat:**(float *)*floatValue*

Scans a **float** value if possible, returning it by reference in *floatValue*. Returns YES if the scanner finds a valid floating-point representation, NO otherwise. Returns HUGE_VAL or –HUGE_VAL by reference in *floatValue* on overflow, or 0.0 on underflow. Skips past excess digits in the case of overflow, so that the scanner's position is past the entire floating-point representation.

Invoke this method with NULL as *floatValue* to simply scan past a **float** value representation.

Note: Floating-point representations are assumed to be IEEE compliant.

See also: – **floatValue** (NSString)

scanHexInt:

– (BOOL)**scanHexInt:**(unsigned int *)*intValue*

Scans an **int** value from a hexadecimal representation if possible, returning it by reference in *intValue*. The hexadecimal integer representation may optionally be preceded by "0x" or "0X". Returns YES if the scanner finds a valid hexadecimal integer representation, NO otherwise. Returns INT_MAX or INT_MIN by reference in *intValue* on overflow. Skips past excess digits in the case of overflow, so that the scanner's position is past the entire hexadecimal representation.

Invoke this method with NULL as *intValue* to simply scan past a hexadecimal integer representation.

scanInt:

– (BOOL)**scanInt:**(int *)*intValue*

Scans an **int** value from a decimal representation if possible, returning it by reference in *intValue*. Returns YES if the scanner finds a valid decimal integer representation, NO otherwise. Returns INT_MAX or INT_MIN by reference in *intValue* on overflow. Skips past excess digits in the case of overflow, so that the scanner's position is past the entire decimal representation.

Invoke this method with NULL as *intValue* to simply scan past a decimal integer representation.

See also: – **intValue** (NSString)

scanLocation

– (unsigned int)**scanLocation**

Returns the character position at which the scanner begins its next scanning operation.

See also: – **setScanLocation:**

scanLongLong:

– (BOOL)**scanLongLong:**(long long *)*longLongValue*

Scans a **long long int** value from a decimal representation if possible, returning it by reference in *value*. Returns YES if the scanner finds a valid decimal integer representation, NO otherwise. Returns

LONG_LONG_MAX or LONG_LONG_MIN by reference in *longLongValue* on overflow. All overflow digits are skipped. Skips past excess digits in the case of overflow, so that the scanner's position is past the entire decimal representation.

Invoke this method with NULL as *longLongValue* to simply scan past a long decimal integer representation.

scanString:intoString:

– (BOOL)**scanString:**(NSString *)*string*
  intoString:(NSString **)*stringValue*

Scans for *string*, and, if a match is found, returns an equivalent string object by reference in *stringValue*. Returns YES if *stringValue* matches the characters at the scan location; otherwise returns NO.

Invoke this method with NULL as *value* to simply scan past a given string.

See also: – **scanUpToString:intoString:**

scanUpToCharactersFromSet:intoString:

– (BOOL)**scanUpToCharactersFromSet:**(NSCharacterSet *)*stopSet*
  intoString:(NSString **)*stringValue*

Scans the string until a character from *stopSet* is encountered, accumulating characters into a string that's returned by reference in *stringValue*. Returns YES if the scanner scans any characters; otherwise returns NO.

Invoke this method with NULL as *stringValue* to simply scan up to a given set of characters.

See also: – **scanCharactersFromSet:intoString:**

scanUpToString:intoString:

– (BOOL)**scanUpToString:**(NSString *)*stopString*
  intoString:(NSString **)*stringValue*

Scans the string until *stopString* is encountered, accumulating characters into a string that's returned by reference in *stringValue*. Returns YES if the scanner scans any characters; otherwise returns NO.

Invoke this method with NULL as *stringValue* to simply scan up to a given string.

See also: – **scanString:intoString:**

setCaseSensitive:

– (void)**setCaseSensitive:**(BOOL)*flag*

If *flag* is YES, the scanner will distinguish case when scanning characters. If *flag* is NO, it will ignore case distinctions. NSScanners are by default *not* case sensitive. Note that case sensitivity doesn't apply to the characters to be skipped.

See also: – **caseSensitive**, – **setCharactersToBeSkipped:**

setCharactersToBeSkipped:

– (void)**setCharactersToBeSkipped:**(NSCharacterSet *)*skipSet*

Sets the scanner to ignore the characters in *skipSet* when scanning its string for a value representation. For example, if a scanner ignores spaces and you send it a **scanInt:** message, it skips spaces until it finds a decimal digit or other character. While an element is being scanned, however, no characters are skipped. If you scan for something made of characters in the set to be skipped (for example, using **scanInt:** when the set of characters to be skipped is the decimal digits), the result is undefined.

The characters to be skipped are treated literally as single values. A scanner doesn't apply its case sensitivity setting to these characters, and doesn't attempt to match composed character sequences with anything in the set of characters to be skipped (though it does match precomposed characters individually). If you want to skip all vowels while scanning a string, for example, you can set the characters to be skipped to those in the string "AEIOUaeiou" (plus any accented variants with precomposed characters).

The default set of characters to skip is the whitespace and newline character set.

See also: – **charactersToBeSkipped**, + **whitespaceAndNewlineCharacterSet** (NSCharacterSet)

setLocale:

– (void)**setLocale:**(NSDictionary *)*aLocale*

Sets the scanner's locale to *aLocale*. A scanner's locale affects the way it interprets values from the string. In particular, a scanner uses the locale's decimal separator to distinguish the integer and fractional parts of floating-point representations. A new scanner's locale is by default **nil**, which causes it to use the default locale values.

See "Locales" in the "Other Features" section of the Foundation Kit documentation for more information on locales.

See also: – **locale**

setScanLocation:

– (void)**setScanLocation:**(unsigned int)*index*

Sets the location at which the next scan operation begins to *index*. This method is useful for backing up to re-scan after an error. Raises an NSRangeException if *index* is beyond the end of the string being scanned.

Rather than setting the scan location directly to skip known sequences of characters, use **scanString:intoString:** or **scanCharactersFromSet:intoString:**, which allow you to verify that the expected substring (or set of characters) is in fact present.

See also: – **scanLocation**

string

– (NSString *)**string**

Returns the string that the scanner was created or initialized with.

See also: – **locale**