NSScanner

Inherits From: NSObject

Conforms To: NSCopying

Declared In: foundation/NSScanner.h

Class Description

An NSScanner interprets and converts text in an NSString into number and string values. You assign a string object to a scanner on creation, and the scanner progresses through the characters of that string from beginning to end as you request items. An NSScanner can be configured to note or ignore case distinctions and to skip certain characters while scanning. Its scan location can also be set, so that you can re-scan a portion of the string, or skip forward a certain amount of characters.

NSScanner is implemented as a class cluster, although it has only one public class. See ªClass Clustersº in the introduction to the Foundation Kit for a description of how to subclass a member of a class cluster.

As an example of using an NSScanner, suppose you have a string object containing lines of the form:

Product: *product name*; Cost: *cost in dollars*

You could scan each line like this:

```
- (BOOL)scanProductString:(NSString *)aString
{
    NSCharacterSet *semicolonSet;
    NSScanner *theScanner;

    semicolonSet = [NSCharacterSet
        characterSetWithCharactersInString:@";"];

    theScanner = [NSScanner scannerWithString:aString];

    while ([theScanner isAtEnd] == NO) {
        NSString *PRODUCT = @"Product: ";
        NSString *COST = @"Cost: ";
        NSString *prodName;
        int prodCost;
        BOOL scanResult;

        /* Skip the "Product: " part. */
        scanResult = [theScanner scanString:PRODUCT intoString:NULL];
        if (scanResult == NO) return NO;

        /* Read in the product name, up to a semicolon. */
        scanResult = [theScanner
            scanUpToCharactersFromSet:semicolonSet
            intoString:&prodName];
        if (scanResult == NO) return NO;

        /* Now go past the semicolon we scanned up to. */
        scanResult = [theScanner scanString:@";" intoString:NULL];
        if (scanResult == NO) return NO;

        /* Skip the "Cost: " part. Because a scanner by default
         * ignores whitespace we don't have to worry about the
         * space between the previous entry and "Cost: ".
         */
        scanResult = [theScanner scanString:COST intoString:NULL];
        if (scanResult == NO) return NO;

        /* Read in the product cost. */
        scanResult = [theScanner scanInt:&prodCost];
```

```
            if (scanResult == NO) return NO;

            /* Do something with prodName and prodCost. */

            /* Because a scanner by default ignores whitespace we
             * don't have to worry about the newline.
             */
        }

    return YES;
}
```

Adopted Protocols

NSCopying	- copyWithZone: - copy

Method Types

Creating an NSScanner	- initWithString: + scannerWithString:
Getting an NSScanner's String	- string
Configuring an NSScanner	- setScanLocation: - scanLocation - setCaseSensitive: - caseSensitive - setCharactersToBeSkipped: - charactersToBeSkipped
Scanning a String	- scanInt: - scanLongLong: - scanFloat: - scanDouble:

- scanString:intoString:
- scanCharactersFromSet:intoString:
- scanUpToString:intoString:
- scanUpToCharactersFromSet:intoString:
- isAtEnd

Class Methods

scannerWithString:

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

Creates and returns an NSScanner that scans *aString*.

See also: - **initWithString:**

Instance Methods

caseSensitive

- (BOOL)**caseSensitive**

Returns YES if the scanner distinguishes case, NO otherwise. NSScanners are by default *not* case sensitive.

See also: - **setCaseSensitive:**

charactersToBeSkipped

- (NSCharacterSet *)**charactersToBeSkipped**

Returns a character set containing those characters that the scanner ignores when looking for a scannable element. For example, if a scanner ignores spaces and you ask it to **scanInt:**, it will skip 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 is the whitespace and newline character set.

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

initWithString:

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

Initializes the receiver, a newly allocated NSScanner, to scan *aString*. Returns **self**.

See also: + **scannerWithString:**

isAtEnd

- (BOOL)**isAtEnd**

Returns YES if the scanner has exhausted all characters in its string, NO if there are characters left to scan. Characters the scanner would skip are ignored; if only skipped characters remain, this method returns YES.

See also: - **charactersToBeSkipped**

scanCharactersFromSet:intoString:

- (BOOL)**scanCharactersFromSet:**(NSCharacterSet *)*aSet* **intoString:**(NSString **)*value*

Scans the string as long as characters from *aSet* are encountered, accumulating characters into a string that's returned by reference in *value*. If any characters are scanned, returns YES; otherwise returns NO.

This method may be invoked with **nil** as *value* to simply scan past a given set of characters.

See also: - **scanUpToCharactersFromSet:intoString:**

scanDouble:

- (BOOL)**scanDouble:**(double *)*value*

Scans a **double** into *value* if possible. Returns YES if a valid floating-point expression was scanned, NO otherwise. HUGE_VAL or -HUGE_VAL is put in *value* on overflow, 0.0 on underflow.

scanFloat:

- (BOOL)**scanFloat:**(float *)*value*

Scans a **float** into *value* if possible. Returns YES if a valid floating-point expression was scanned, NO otherwise. HUGE_VAL or -HUGE_VAL is put in *value* on overflow, 0.0 on underflow.

scanInt:

- (BOOL)**scanInt:**(int *)*value*

Scans an **int** into *value* if possible. Returns YES if a valid integer expression was scanned, NO otherwise. INT_MAX or INT_MIN is put in *value* on overflow.

scanLocation

- (unsigned)**scanLocation**

Returns the character index at which the scanner will begin its next scanning operation.

See also: - **setScanLocation:**

scanLongLong:

- (BOOL)**scanLongLong:**(long long *)*value*

Scans a **long long int** into *value* if possible. Returns YES if a valid integer expression was scanned, NO otherwise. LONG_LONG_MAX or LONG_LONG_MIN is put in *value* on overflow.

scanString:intoString:

- (BOOL)**scanString:**(NSString *)*aString* **intoString:**(NSString **)*value*

Scans for *aString*, and if a match is found returns (by reference) in *value* a string object equal to it. If *aString* matches the characters at the scan location, returns YES; otherwise returns NO.

This method may be invoked with **nil** as *value* to simply scan past a given string.

See also: - **scanUpToString:intoString:**

scanUpToCharactersFromSet:intoString:

- (BOOL)**scanUpToCharactersFromSet:**(NSCharacterSet *)*aSet* **intoString:**(NSString **)*value*

Scans the string until a character from *aSet* is encountered, accumulating characters encountered into a string that's returned by reference in *value*. If any characters are scanned, returns YES; otherwise returns NO.

This method may be invoked with **NULL** as *value* to simply scan up to a given set of characters.

See also: - **scanCharactersFromSet:intoString:**

scanUpToString:intoString:

- (BOOL)**scanUpToString:**(NSString *)*aString* **intoString:**(NSString **)*value*

Scans the string until *aString* is encountered, accumulating characters encountered into a string that's returned by reference in *value*. The receiver's scan location will then be at the beginning of *aString* (or at the end of the string being scanned if *aString* isn't found). If any characters are scanned, returns YES; otherwise returns NO.

This method may be invoked with **NULL** as *value* 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.

See also: - **caseSensitive**

setCharactersToBeSkipped:

- (void)**setCharactersToBeSkipped:**(NSCharacterSet *)*aSet*

Sets the scanner to ignore characters from *aSet* when scanning its string. Such characters are simply passed by during scanning. For example, if a scanner ignores spaces and you ask it to **scanInt:**, it will skip 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 is the whitespace and newline character set.

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

setScanLocation:

- (void)**setScanLocation:**(unsigned)*index*

Sets the location at which the next scan will begin to *index*. This method is useful for backing up to re-scan after an error. You should use **scanString:intoString:** or **scanCharactersFromSet:intoString:** to skip ahead past known sequences of characters, as this allows you to check for errors in a way that setting the scan location ahead doesn't.

See also: - **scanLocation**

string

- (NSString *)**string**

Returns the string object that the scanner was created with.