MiscKit Release 1.2.5, MiscString Version 1.95 Copyright ©1993, 1994 by Don Yacktman. All Rights Reserved.

MiscString

Inherits From: Object

Conforms To: NXTransport

Declared In: misckit/MiscString.h

Class Description

A MiscString object contains a simple text string and provides methods for its manipulation, encompassing all the functions available in <strings.h>. The MiscString object automatically handles the freeing and copying of character strings. Although it simplifies string operations, it does not yet incorporate the benefits of the NXAtom type, which you may wish to use instead. Certain tradeoffs have been made between speed and robustness, typically in favor of robustness. If there is enough demand, a MiscFastString class may someday appear. Another free string class, RCString, is also available (under the GNU General Public License) which incorporates regular expression matching, reference counting, and other useful functions. Some of these features will probably eventually appear in the MiscString class to a certain degree. Depending on your needs, you may find

the RCString class to better suit your needs. Another free string class is the MOString by Mike Ferris, which is a part of the MOKit. There are also now commercial string classes available from several sources, which you may want to consider. This particular class is free and may be used in commercial applications, so the price is very attractive.

A MiscString is created through the normal process of **±alloc** and **±init**. It may be set to a specific string by means of the **±setStringValue:** and **±takeStringValue:** methods. As a shortcut, +**newWithString:** is also available. To copy an existing MiscString, use the **±copy** and **±copyFromZone:** methods. To copy a portion of a string, use the **±subStringLeft:**, **±subStringRight:**, **±left:...**, **±midFrom:...**, and **±right...:** methods. Use the **±concatenate:...** and **±cat:...** methods to concatenate another string (or many strings) onto the end of the string in the buffer. If the current buffer is too small, it is enlarged. To tokenize or create copies of substrings, use the ±**extractPart:...**, ±**fileName...**, ±**pathName...**, ±**subStringLeft**, and ±**subStringRight** methods. Using ±**encrypt:** creates a MiscString encrypted by the crypt(3) function. You can use ±**reverse** to reverse the MiscString's contents. Use **±free** to free a MiscString and it's buffer or **±freeString** to free just the buffer.

Inserting characters or strings into a MiscString may be performed via any of the ±**insert:at:**, ±**insertChar:at:**, ±**insertString:at:**, ±**padToLength:withChar:**, and ±**addChar:** methods. Deleting a portion of the MiscString is performed by the ±**trimLeadSpaces**, ±**trimTailSpaces**, ±**trimSpaces**, ±**squashSpaces**, and ±**removeFrom:...** methods. A whole series of ±**replace...** methods provide flexible substring and character replacement options.

The **±length** method returns the length of the string currently in the buffer and **±stringValue** returns a pointer to the string itself. The ±**emptyString** method returns YES if the MiscString is empty, which would correspond to a string of zero length or a NULL pointer in C. The ±**index:...** methods return a pointer to the n^{th} occurrence in the buffer of a specific character and ±**rindex:...** return a pointer to the n^{th} occurrence from the end (right to left). The ±**spotOf:...** and ±**rspotOf:** methods work similarly, but return an integer which gives the character number of the occurrence. You can find out about what types of characters are in a MiscString by using ±**hasType:** and ±**isAllOfType:**. The constants that can be passed to request various lexical groups are described in the ±**hasType:** method documentation below.

You may compare strings to each other by means of the various **±isEqual:...**, **±cmp:...**, **± casecmp:...**, **±endcmp:...**, **± endcasecmp:...**, **±compareTo:...**, and **±endCompareTo:...** methods. The **±isEqual:...**, **±compareTo:...**, and **±compareTo:...** methods are preferred, since they use NXStringOrderTables to make the comparison and are therefore more accurate with respect to international, accented, and ligature characters. If you need to use a table different from the default, use the **±setStringOrderTable:** method.

Use the ±**numWords** method to count words in the MiscString, and ±**wordNum:** to create a new MiscString containing a specific word. Note that these methods do more than tokenizing via spaces; all whitespaces (space, tab, return, linefeed; as recognized by NXIsSpace()) delimit the words. Consecutive whitespace characters are treated as a single delimiter.

Use the ±**toUpper** and ±**toLower** methods to change all characters in the MiscString to upper or lower case. Note that the NX...() functions are used to perform this conversion, so it should work even with international character sets. Use ±**reverse** to reverse all the characters in the MiscString's buffer. The ±**charAt:** method returns a single character from a given location in the MiscString's buffer.

A MiscString may be archived by means of the **±read:** and **±write:** methods. (Call NXReadObject() and NXWrite[Root]Object() functions and not the **±read:** and **±write:** methods directly.)

There are several methods which are designed to handle regular expression searching, matching, and so on. To determine if a MiscString matches a regular expression, use the ±**grep:...** or ±**grepString:...** methods; you can use ±**grep...before:middle:after:** variants to split up a MiscString into the matched section and the parts before and after the match. To replace a section of a MiscString which matches a regular expression, simply use one of the ±**replaceRegex...** and ±**replaceEveryOccurrenceOfRegex...** methods. The only difference between the ±**replacePattern** methods is whether the arguments are objects or character pointers. You can find out if a MiscString matches a regular expression with ±**matchesRegex:...** and find the number of individual matches with the ±**numOfRegex:...** methods. You can find where regular expression matches occur with ±**spotOfRegex:...** and ±**rspotOfRegex...** as well.

Two conventions with char * buffers are followed by the MiscString class. First, any method which has an

argument of type ªconst char *º will either make a copy of the argument or discard the pointer upon exit of the method, so it is safe for you to free it any time afterward. Second, any pointer returned as a ªconst char *º by a method could be freed at any time by the MiscString object, so you ought to copy it yourself if you intend to keep it around for any length of time. When a MiscString object gives out such a pointer, it assumes that it won't be cached by the caller. If you violate this assumption, you will most definitely create mysterious crashing bugs when you start accessing freed pointers and such. The compiler's -Wall flag will catch most mistakes of this nature for you.

Although not included below, the MiscString object also implements ±**getIBImage** and ±**getInspectorClassName** messages to support Interface Builder palettes. A palette is included in this distribution which allows you to save a MiscString in your .nib files with it pre-initialized to an arbitrary string value.

Disclaimer and other notes: If you have any problems with the MiscString class or wish to suggest improvements, the author may be contacted via e-mail to Don_Yacktman@byu.edu. Since this object is free, please understand that the author cannot be held responsible for any problems this code may cause. You use it at your own risk. (The author himself uses this code, too, if that's any consolation.) Also remember that the author's ability to support this software is highly dependent upon free time available, which is often quite scarce. If you wish to use this, but for some reason require support and some sort of ªcommercialº standing for this class, contact the author; support can be bought if you need it. (Why you'd need or want to pay for support for such a simple object is beyond me, though!) Many thanks are due to Carl Lindberg who has contributed many of the methods that were not available in versions before version 1.1., as well as every one else who has taken the time to send in feedback. You may find it worth noting some of the comments in the source file MiscString.m; several trade-offs have been made, typically in favor of making the code more maintainable and robust, but at the expense of speed. One important thing to note is that any method which takes an object as an argument (even if the description claims it should be a MiscString) will work for ***any*** object which responds to ±**stringValue** messages, but will work faster for MiscString objects. This can be very useful when working with NeXT's AppKit, since many of those objects respond to ±**stringValue**.

If you are not careful, you can get memory leaks with the MiscString class. If you nest a method that returns a new MiscString instance inside of a method that does not return **self**, you will leak memory. The most common occurrence of this, by far, is to call a method that creates a MiscString and follow it with a ±**stringValue** message. (For example, the message `[[aString pathName] stringValue]` will leak memory.) The current workaround for this is to use the ±**stringValueAndFree** method. Thus, the example message ought to be `[[aString pathName] stringValueAndFree]` to avoid the memory leak. Unfortunately, you must know when to use this method. To help you remember which methods might cause you trouble, here is a list of methods which return new MiscString instances: ±**encrypt:**, ±**extractPart:useAsDelimiter:**, ±**extractPart:useAsDelimiter:caseSensitive:**, ±**extractPart:useAsDelimiter:caseSensitive:fromZone:**, ±**extractPart:useAsDelimiter:fromZone:**, ±**fileName**, ±**fileNameFromZone:**, ±**left:**, ±**left:fromZone:**, ±**midFrom:length:** ±**midFrom:to:**, ±**midFrom:to:fromZone:**, ±**pathName**, ±**pathNameFromZone:**, ±**wordNum:**, ±**wordNum:fromZone:**, ±**right:**, ±**right:fromZone:**, ±**subStringLeft:**, and ±**subStringRight:**.

You can load a MiscString from a file, stream, or stdin by using ±**fgets:**, ±**fgets:keepNewline:**, ±**loadFromFile:** ±**gets:**, ±**streamGets:**, or ± **streamGets:keepNewline:**.

Although this class description is quite extensive, there are a few methods of the MiscString which are not yet documented. A list of these methods is at the end of this document. If any of these methods sound interesting to you, feel free to look at the implementation in the source code which should give you a good idea of how they work.

Instance Variables

int **length**;
int **_length**;
char ***buffer**;

length	Length of string currently in storage
_length	Length in bytes of allocated buffer
buffer	Stored character string

Method Types

Initializing and freeing a MiscString
± init
± initString:
± allocateBuffer:
± allocateBuffer:fromZone:
± free
± freeString
+ initialize
+ new
+ newWithString:

Copying a MiscString
± copyFromZone:
± extractPart:useAsDelimiter:
± extractPart:useAsDelimiter:caseSensitive:
± extractPart:useAsDelimiter:caseSensitive:fromZone:
± extractPart:useAsDelimiter:fromZone:
± fileName
± fileNameFromZone:

	± left: ± left:fromZone: ± right: ± right:fromZone: ± midFrom:to: ± midFrom:to:fromZone: ± midFrom:length: ± midFrom:length:fromZone: ± pathName ± pathNameFromZone: ± subStringLeft: ± subStringRight: ± wordNum:
File I/O with a MiscString	± fgets: ± fgets:keepNewLine: ± gets ± loadFromFile: ± streamGets: ± streamGets:keepNewLine:
Manipulating a MiscString	± addChar: ± capitalizeEachWord ± cat: ± cat:n: ± cat:fromZone: ± cat:n:fromZone: ± catStrings:

± concatenate:
± concatenate:n:
± concatenate:fromZone:
± concatenate:n:fromZone:
± concatenateStrings:
± insert:
± insert:at:
± insertChar:
± insertChar:at:
± insertString:
± insertString:at:
± invertCases
± padFrontToLength:withChar:
± padToLength:withChar:
± removeFrom:length:
± removeFrom:to:
± replace:with:
± replace:with:caseSensitive:
± replace:with:occurrenceNum:
± replace:with:occurrenceNum:caseSensitive:
± replace:with:overlap:
± replace:with:caseSensitive:overlap:
± replace:with:occurrenceNum:overlap:
± replace:with:occurrenceNum:caseSensitive:overlap:
± replace:withChar:
± replace:withChar:caseSensitive:
± replace:withChar:occurrenceNum:
± replace:withChar:occurrenceNum:caseSensitive:

± replace:withChar:overlap:
± replace:withChar:caseSensitive:overlap:
± replace:withChar:occurrenceNum:overlap:
± replace:withChar:occurrenceNum:caseSensitive:overlap:
± replace:withString:
± replace:withString:caseSensitive:
± replace:withString:occurrenceNum:
± replace:withString:occurrenceNum:caseSensitive:
± replace:withString:overlap:
± replace:withString:caseSensitive:overlap:
± replace:withString:occurrenceNum:overlap:
± replace:withString:occurrenceNum:caseSensitive:overlap:
± replaceCharAt:with:
± replaceEveryOccurrenceOf:with:
± replaceEveryOccurrenceOf:with:caseSensitive:
± replaceEveryOccurrenceOf:with:overlap:
± replaceEveryOccurrenceOf:with:caseSensitive:overlap:
± replaceEveryOccurrenceOf:withChar:
± replaceEveryOccurrenceOf:withChar:caseSensitive:
± replaceEveryOccurrenceOf:withChar:overlap:
± replaceEveryOccurrenceOf:withChar:caseSensitive:overlap:
± replaceEveryOccurrenceOf:withString:
± replaceEveryOccurrenceOf:withString:caseSensitive:
± replaceEveryOccurrenceOf:withString:overlap:
± replaceEveryOccurrenceOf:withString:caseSensitive:overlap:
± replaceEveryOccurrenceOfChar:with:
± replaceEveryOccurrenceOfChar:with:caseSensitive:
± replaceEveryOccurrenceOfChar:withChar:

± replaceEveryOccurrenceOfChar:withChar:caseSensitive:
± replaceEveryOccurrenceOfChar:withString:
± replaceEveryOccurrenceOfChar:withString:caseSensitive:
± replaceEveryOccurrenceOfChars:with:
± replaceEveryOccurrenceOfChars:with:caseSensitive:
± replaceEveryOccurrenceOfChars:withChar:
± replaceEveryOccurrenceOfChars:withChar:caseSensitive:
± replaceEveryOccurrenceOfChars:withString:
± replaceEveryOccurrenceOfChars:withString:caseSensitive:
± replaceEveryOccurrenceOfRegex:with:
± replaceEveryOccurrenceOfRegex:with:caseSensitive:
± replaceEveryOccurrenceOfRegex:withChar:
± replaceEveryOccurrenceOfRegex:withChar:caseSensitive:
± replaceEveryOccurrenceOfRegex:withString:
± replaceEveryOccurrenceOfRegex:withString:caseSensitive:
± replaceEveryOccurrenceOfString:with:
± replaceEveryOccurrenceOfString:with:caseSensitive:
± replaceEveryOccurrenceOfString:with:overlap:
± replaceEveryOccurrenceOfString:with:caseSensitive:overlap:
± replaceEveryOccurrenceOfString:withChar:
± replaceEveryOccurrenceOfString:withChar:caseSensitive:
± replaceEveryOccurrenceOfString:withChar:overlap:
± replaceEveryOccurrenceOfString:withChar:caseSensitive:overlap:
± replaceEveryOccurrenceOfString:withString:
± replaceEveryOccurrenceOfString:withString:caseSensitive:
± replaceEveryOccurrenceOfString:withString:overlap:
± replaceEveryOccurrenceOfString:withString:caseSensitive:overlap:
± replaceEveryOccurrenceOfType:with:

± replaceEveryOccurrenceOfType:withChar:
± replaceEveryOccurrenceOfType:withString:
± replaceFrom:length:with:
± replaceFrom:length:withChar:
± replaceFrom:length:withString:
± replaceFrom:to:with:
± replaceFrom:to:withChar:
± replaceFrom:to:withString:
± replaceHomeWithTilde
± replaceRegex:with:
± replaceRegex:with:caseSensitive:
± replaceRegex:with:occurrenceNum:
± replaceRegex:with:occurrenceNum:caseSensitive:
± replaceRegex:withChar:
± replaceRegex:withChar:caseSensitive:
± replaceRegex:withChar:occurrenceNum:
± replaceRegex:withChar:occurrenceNum:caseSensitive:
± replaceRegex:withString:
± replaceRegex:withString:caseSensitive:
± replaceRegex:withString:occurrenceNum:
± replaceRegex:withString:occurrenceNum:caseSensitive:
± replaceString:with:
± replaceString:with:caseSensitive:
± replaceString:with:occurrenceNum:
± replaceString:with:occurrenceNum:caseSensitive:
± replaceString:with:overlap:
± replaceString:with:caseSensitive:overlap:
± replaceString:with:occurrenceNum:overlap:

± replaceString:with:occurrenceNum:caseSensitive:overlap:
± replaceString:withChar:
± replaceString:withChar:caseSensitive:
± replaceString:withChar:occurrenceNum:
± replaceString:withChar:occurrenceNum:caseSensitive:
± replaceString:withChar:overlap:
± replaceString:withChar:caseSensitive:overlap:
± replaceString:withChar:occurrenceNum:overlap:
± replaceString:withChar:occurrenceNum:caseSensitive:overlap:
± replaceString:withString:
± replaceString:withString:caseSensitive:
± replaceString:withString:occurrenceNum:
± replaceString:withString:occurrenceNum:caseSensitive:
± replaceString:withString:overlap:
± replaceString:withString:caseSensitive:overlap:
± replaceString:withString:occurrenceNum:overlap:
± replaceString:withString:occurrenceNum:caseSensitive:overlap:
± replaceTildeWithHome
± replaceType:with:
± replaceType:with:occurrenceNum:
± replaceType:withChar:
± replaceType:withChar:occurrenceNum:
± replaceType:withString:
± replaceType:withString:occurrenceNum:
± setStringValue:
± setStringValue:fromZone:
± squashSpaces
± takeStringValueFrom:

	± takeStringValueFrom:fromZone: ± toLower ± toUpper ± trimLeadSpaces ± trimLeadWhiteSpaces ± trimSpaces ± trimTailSpaces ± trimTailWhiteSpaces ± trimWhiteSpaces
Querying attributes	± charAt: ± cmp: ± cmp:n: ± casecmp: ± casecmp:n: ± compareTo: ± compareTo:n: ± compareTo:caseSensitive: ± compareTo:n:caseSensitive: ± emptyString ± endcasecmp: ± endcasecmp:n: ± endcmp: ± endcmp:n: ± endCompareTo: ± endCompareTo:caseSensitive: ± endCompareTo:n: ± endCompareTo:n:caseSensitive:

± grep:
± grep:caseSensitive:
± grep:occurrenceNum:
± grep:occurrenceNum:caseSensitive:
± grep:before:middle:after:
± grep:caseSensitive:before:middle:after:
± grep:occurrenceNum:before:middle:after:
± grep:occurrenceNum:caseSensitive: before:middle:after:
± hasType:
± index:
± index:caseSensitive:
± index:occurrenceNum:
± index:occurrenceNum:caseSensitive:
± indexOfChars:
± indexOfChars:caseSensitive:
± indexOfChars:occurrenceNum:
± indexOfChars:occurrenceNum:caseSensitive:
± isAllOfType:
± isEqual:
± length
± matchesRegex:
± matchesRegex: caseSensitive:
± numOf:
± numOf:caseSensitive:
± numOf:overlap:
± numOf:caseSensitive:overlap:
± numOfChar:
± numOfChar:caseSensitive:

± numOfChars:
± numOfChars:caseSensitive:
± numOfRegex:
± numOfRegex: caseSensitive:
± numOfString:
± numOfString:caseSensitive:
± numOfString:overlap:
± numOfString:caseSensitive:overlap:
± numOfType:
± numWords
± rindex:
± rindex:caseSensitive:
± rindex:occurrenceNum:
± rindex:occurrenceNum:caseSensitive:
± rindexOfChars:
± rindexOfChars:caseSensitive:
± rindexOfChars:occurrenceNum:
± rindexOfChars:occurrenceNum:caseSensitive:
± rspotOf:
± rspotOf:caseSensitive:
± rspotOf:occurrenceNum:
± rspotOf:occurrenceNum:caseSensitive:
± rspotOfChars:
± rspotOfChars:caseSensitive:
± rspotOfChars:occurrenceNum:
± rspotOfChars:occurrenceNum:caseSensitive:
± rspotOfRegex:
± rspotOfRegex:caseSensitive:

± rspotOfRegex:occurrenceNum:
± rspotOfRegex:occurrenceNum:caseSensitive:
± rspotOfRegex:length:
± rspotOfRegex:caseSensitive:length:
± rspotOfRegex:occurrenceNum:length:
± rspotOfRegex:occurrenceNum:caseSensitive:length:
± rspotOfStr:
± rspotOfStr:caseSensitive:
± rspotOfStr:occurrenceNum:
± rspotOfStr:occurrenceNum:caseSensitive:
± rspotOfStr:overlap:
± rspotOfStr:caseSensitive:overlap:
± rspotOfStr:occurrenceNum:overlap:
± rspotOfStr:occurrenceNum:caseSensitive:overlap:
± rspotOfString:
± rspotOfString:caseSensitive:
± rspotOfString:occurrenceNum:
± rspotOfString:occurrenceNum:caseSensitive:
± rspotOfString:overlap:
± rspotOfString:caseSensitive:overlap:
± rspotOfString:occurrenceNum:overlap:
± rspotOfString:occurrenceNum:caseSensitive:overlap:
± rspotOfType:
± rspotOfType:occurrenceNum:
± rstrstr:
± rstrstr:caseSensitive:
± rstrstr:occurrenceNum:
± rstrstr:occurrenceNum:caseSensitive:

± rstrstr:overlap:
± rstrstr:caseSensitive:overlap:
± rstrstr:occurrenceNum:overlap:
± rstrstr:occurrenceNum:caseSensitive:overlap:
± rstrString:
± rstrString:caseSensitive:
± rstrString:occurrenceNum:
± rstrString:occurrenceNum:caseSensitive:
± rstrString:overlap:
± rstrString:caseSensitive:overlap:
± rstrString:occurrenceNum:overlap:
± rstrString:occurrenceNum:caseSensitive:overlap:
± setStringOrderTable:
± spotOf:
± spotOf:caseSensitive:
± spotOf:occurrenceNum:
± spotOf:occurrenceNum:caseSensitive:
± spotOfChars:
± spotOfChars:caseSensitive:
± spotOfChars:occurrenceNum:
± spotOfChars:occurrenceNum:caseSensitive:
± spotOfStr:
± spotOfStr:caseSensitive:
± spotOfStr:occurrenceNum:
± spotOfStr:occurrenceNum:caseSensitive:
± spotOfStr:overlap:
± spotOfStr:caseSensitive:overlap:
± spotOfStr:occurrenceNum:overlap:

± spotOfStr:occurrenceNum:caseSensitive:overlap:
± spotOfString:
± spotOfString:caseSensitive:
± spotOfString:occurrenceNum:
± spotOfString:occurrenceNum:caseSensitive:
± spotOfString:overlap:
± spotOfString:caseSensitive:overlap:
± spotOfString:occurrenceNum:overlap:
± spotOfString:occurrenceNum:caseSensitive:overlap:
± spotOfType:
± spotOfType:occurrenceNum:
± spotOfRegex:
± spotOfRegex: caseSensitive:
± spotOfRegex: occurrenceNum:
± spotOfRegex: occurrenceNum: caseSensitive:
± spotOfRegex: length:
± spotOfRegex: caseSensitive: length:
± spotOfRegex: occurrenceNum: length:
± spotOfRegex: occurrenceNum: caseSensitive: length:
± stringOrderTable
± stringValue
± strstr:
± strstr:caseSensitive:
± strstr:occurrenceNum:
± strstr:occurrenceNum:caseSensitive:
± strstr:overlap:
± strstr:caseSensitive:overlap:
± strstr:occurrenceNum:overlap:

	± strstr:occurrenceNum:caseSensitive:overlap: ± strString: ± strString:caseSensitive: ± strString:occurrenceNum: ± strString:occurrenceNum:caseSensitive: ± strString:overlap: ± strString:caseSensitive:overlap: ± strString:occurrenceNum:overlap: ± strString:occurrenceNum:caseSensitive:overlap:
Archiving	± read: ± write:

Class Methods

extensionSeparator
+ (char)**extensionSeparator**

Returns the character currently being used by all MiscStrings to separate a file's name from its extension.

See also: +**pathSeparator**, +**setExtensionSeparator:**, and +**setPathSeparator:**

initialize
+ **initialize**

Initializes the MiscString class. Returns **self**.

new
+ **new**

Returns a new, empty MiscString object. Same as `[[MiscString alloc] init]`.

newWithString:
+ **newWithString:**(char *)*aString*

Returns a new MiscString object containing *aString*.

pathSeparator
+ (char)**pathSeparator**

Returns the character currently being used by all MiscStrings to separate path elements.

See also: +**extensionSeparator**, +**setExtensionSeparator:**, and +**setPathSeparator:**

setExtensionSeparator:
+ **setExtensionSeparator:**(char)*c*

Changes the character used to separate separate a file's name from its extension by all MiscString. By default, MiscString assumes `.' is between a filename and the extension. Returns *self*.

See also: +**extensionSeparator**, +**pathSeparator**, and +**setPathSeparator:**

setPathSeparator:
+ **setPathSeparator:**(char)*c*

Changes the character used to separate path elements by all MiscString. By default, MiscString assumes `/' separates path elements. Returns *self*.

See also: +**extensionSeparator**, +**pathSeparator**, and +**setExtensionSeparator:**

Instance Methods

addChar:
- **addChar:**(char)*aChar*

Appends *aChar* to the end of *buffer*. Returns **self**.

addExtensionIfNeeded:
- **addExtensionIfNeeded:**(const char *)*aString*

Adds the extension *aString* if it is not already present on the end of the receiving string. Returns *self*.

See also: **-x**

allocateBuffer:

allocateBuffer:fromZone:

- **allocateBuffer:**(int)*size* **fromZone:**(NXZone *)*zone*

If the current buffer is less than *size* bytes, then it is freed and a new buffer is allocated from *zone* to be *size* bytes in length. Returns **self**. You do not need to directly call this method, since the ±**copyFromZone:** and other methods do this automatically. However, you may wish to call this method after calling ±**init** for MiscString objects which will dynamically change in size often. By allocating a buffer which is as least as large as you expect the MiscString to grow to during it's lifetime, your application may run faster. This is because the MiscString object won't have to dynamically grow as often, an operation which can slow things down. In the degenerate method, *zone* defaults to the receiver's zone.

See also: -**copyFromZone:**, and -**setStringValue:fromZone:**

buildInstanceImageIn:

- **buildInstanceImageIn:**(char *)*buf*

Places a string into *buf* which gives information about the MiscString: length, capacity, string buffer address, and contents. Returns *self*.

See also: -**printForDebugger:**, and ± **printToStdErr:**

capacity

- (unsigned)**capacity**

Returns the capacity of the currently allocated buffer, which will always be at least one greater than the length of the actual string.

See also: **-x**

capitalizeEachWord
- **capitalizeEachWord**

Capitalizes the first character of every word in *buffer*. Words are recognized in the same way as in **±numWords** and **-wordNum**. Returns **self**.

casecmp:
- (int)**cmp:**(const char *)*aString*

Calls **strcasecmp()** to perform a case insensitive comparison of *buffer* and *aString*. Return values follow the same rules as **strcasecmp()**. This method is provided for those cases in which MiscString objects are not in use, and therefore only a char pointer is available. It is also useful with constant strings. The ±**compareTo:** methods are preferred for use whenever possible, since they work with objects and use the current string ordering table.

See also: **- casecmp:n:**, **- cmp:**, **- cmp:n:**, and **- compareTo:...**

casecmp:n:
- (int)**cmp:**(const char *)*aString* **n:**(int)*n*

Calls **strncasecmp()** to perform a case insensitive comparison of at most the first *n* characters *buffer* and *aString*. Return values follow the same rules as **strncasecmp()**. This method is provided for those cases in which MiscString objects are not in use, and therefore only a char pointer is available. It is also useful with

constant strings. The ±**compareTo:** methods should be used whenever possible, since they work with objects and use the current string ordering table.

See also: -**casecmp:**, -**cmp:**, -**cmp:n:**, and -**compareTo:...**

catFromFormat:

- **catFromFormat:**(const char *)*format, ...*

This method is like a combination of sprintf() followed by strcat(). It uses *format* and the arguments which follow to create a string according to the user's specification. Then it concatenates this string to the end of the current string. Returns *self*.

See also: -**x**

catFromFormat:valist:

- **catFromFormat:**(const char *)*format*
  valist:(va_list)*param_list*

This method has the same function as ±**catFromFormat:** but with different argument types.

See also: -**x**

cat:
cat:fromZone:
cat:n:
cat:n:fromZone:

- **cat:**(const char *)*aString* **n:**(int)*n* **fromZone:**(NXZone *)*zone*

Calls **strncat()** to concatenate *buffer* and up to the first *n* bytes of *aString*. If the current size of *buffer* is not large enough to fit the concatenation of the two strings, then a new, larger *buffer* is allocated *zone*. Returns **nil**. This method is provided for those cases in which MiscString objects are not in use, and therefore only a char pointer is available. It is also useful with constant strings. The ±**concatenate:** methods are preferred for use whenever possible, mainly because they work with objects. In the degenerate methods, *n* defaults to the length of *aString* and *zone* defaults to the receiver's zone.

See also: -**catStrings:**, -**concatenate:...**, and -**concatenateStrings:**

catStrings:

- **catStrings:**(const char *)*aString*, ¼

Works like ±**cat:** but accepts many string pointers as an argument. They are added to the end of *buffer* in the same order as they are encountered in the argument list. The last argument in the list ***must*** be NULL or unpredictable results (such as segmentation faults) will occur. Returns **self**.

See also: -**cat:**, -**cat:fromZone:**, -**cat:n:**, -**cat:n:fromZone:**, -**concatenate:**, -**concatenate:fromZone:**, -**concatenate:n:**, -**concatenate:n:fromZone:**, and -**concatenateStrings:**

catStrings:valist:

- **catStrings:**(const char *)*strings*
  valist:(va_list)*ptr*

This method has the same function as ±**catStrings:** but with different argument types.

See also: **-x**

charAt:

- (char)**charAt:**(int)*index*

Returns the *index*th character of *buffer*. Returns **0** if *index* is out of range.

cmp:

- (int)**cmp:**(const char *)*aString*

Calls **strcmp()** to compare *buffer* and *aString*. Return values follow the same rules as **strcmp()**. This method is provided for those cases in which MiscString objects are not in use, and therefore only a char pointer is available. It is also useful with constant strings. The ±**compareTo:** methods are preferred for use whenever possible, mainly because they work with objects and use the current string ordering table.

See also: **-casecmp:...**, **-cmp:n:**, and **-compareTo:...**

cmp:n:

- (int)**cmp:**(const char *)*aString* **n:**(int)*n*

Calls **strncmp()** to compare at most the first *n* characters *buffer* and *aString*. Return values follow the same rules as **strncmp()**. This method is provided for those cases in which MiscString objects are not in use, and therefore only a char pointer is available. It is also useful with constant strings. The ±**compareTo:** methods are preferred for use whenever possible, since they work with objects and use the current string ordering table.

See also: **- casecmp:...**, **- cmp:**, and **-compareTo:...**

compareTo:
compareTo:caseSensitive:
compareTo:n:
compareTo:n:caseSensitive:
- (int)**compareTo:**(id)*sender* **n:**(int)*n* **caseSensitive:**(BOOL)*sense*

Compares the string in *buffer* to the **±stringValue** of *sender*. No more than the first *n* characters are used to make the comparison. If *n* is -1, the full lengths of the strings is used. (If the strings differ in length, the longer string will be considered ªgreaterº.) If *sense* is YES, then the comparison is case sensitive. If *sense* is NO, then the comparison ignores case. The value returned is zero if the strings are equal, -1 if the receiver is less than *sender*, and 1 otherwise. The current string ordering table is used to make the comparison. This method is basically a cover for NXOrderStrings(). In the degenerate methods, *sense* defaults to YES and *n* defaults to -1.

See also: - **cmp:...** and - **casecmp:...**

concatenate:
concatenate:fromZone:
concatenate:n:
concatenate:n:fromZone:
- **concatenate:**(id)*sender* **n:**(int)*n* **fromZone:**(NXZone *)*zone*

Adds up to the first *n* bytes of the MiscString *sender* to the end of the string in *buffer*. If the current size of *buffer* is not large enough to fit the concatenation of the two strings, then a new, larger *buffer* is allocated from *zone*. Returns **self**. In the degenerate methods, *n* defaults to the **-length** of *sender* and *zone* defaults to the receiver's zone.

See also: **-cat:...**, **-catStrings:**, and **-concatenateStrings:**

concatenateStrings:

- **concatenateStrings:**(id)*sender*, ¼

Works like ±**concatenate:** but accepts many MiscString objects as an argument. They are added to the end of *buffer* in the same order as they are encountered in the argument list. The last argument in the list ***must*** be **nil** or unpredictable results (such as segmentation faults) will occur. Returns **self**.

See also: **-cat:...**, **-catStrings:**, and **-concatenate:...**

convertToCaseInsensitiveSearchString

- **convertToCaseInsensitiveSearchString**

Converts the MiscString into a Sybase-compatible case-insensitive search string. Returns *self*.

See also: **-x**

convertUnixWildcardsToSybase

- **convertUnixWildcardsToSybase**

Converts the MiscString into a Sybase-compatible search string with any standard UNIX wildcard symbols converted to their Sybase counterparts. Returns *self*.

See also: **-x**

copyFromZone:

- **copyFromZone:**(NXZone *)*zone*

Returns a new MiscString. Memory for the new MiscString is allocated from *zone*. The string stored in *buffer* is copied.

doesExistInFileSystem

- (BOOL)**doesExistInFileSystem**

Returns YES is the receiver contains the path to a file which exists in the current file system. Returns NO otherwise.

See also: **-x**

doubleValue

- (double)**doubleValue**

Returns the double precision floating point value of the receiver, if the receiving string begins with a properly formed floating point number. Any trailing non-numeric characters are ignored. If the receiving string doesn't begin with a number, a zero is returned.

See also: **-intValue** and **-floatValue**

emptyString

- (BOOL)**emptyString**

Returns YES if the MiscString is empty and NO otherwise.

See also: **-length**

encrypt:

- **encrypt:***salt*

Encrypts the receiving string using the UNIX password encryption algorithm with *salt* as the ªsalt.º The ªsaltº argument should be an object that responds to ±**stringValue** and the receiving string should be no longer than eight characters in length. (If it is longer, the extra characters are ignored.) Returns *self*.

See also: **-x**

endcasecmp:
endcasecmp:n:

- **endcasecmp:**(const char *)*aString* **n:**(int)*n*

Performs a case insensitive comparison of the last *n* characters of *buffer* with the last *n* characters of *aString*. If *n* is -1 or *n* is greater than the length of either string, *n* is set to the length of the shorter string. Return values follow those of the **-compareTo:** methods. In the degenerate method, *n* defaults to -1.

See also: **-endcmp:...** and **-endCompareTo:...**

endcmp:
endcmp:n:

- **endcmp:**(const char *)*aString* **n:**(int)*n*

Performs a case sensitive comparison of the last *n* characters of *buffer* with the last *n* characters of *aString*. If *n* is -1 or *n* is greater than the length of either string, *n* is set to the length of the shorter string. Return values follow those of the **-compareTo:** methods. In the degenerate method, *n* defaults to -1.

See also: **-endcasecmp:...** and **-endCompareTo:...**

endCompareTo:
endCompareTo:caseSensitive:
endCompareTo:n:
endCompareTo:n:caseSensitive:
- **endCompareTo:**(id)*sender* **n:**(int)*n* **caseSensitive:**(BOOL)*sense*

Compares the last *n* characters of *sender*'s **-stringValue** with the last *n* characters of *buffer*. If *n* is -1 or *n* is greater than the length of either string, *n* is set to the length of the shorter string. If *sense* is NO, the comparison is case insensitive. Return values follow those of the **-compareTo:** methods. In the degenerate methods, *sense* defaults to YES and *n* defaults to -1.

See also: **-endcmp:...** and **-endcasecmp:...**

extractPart:useAsDelimiter:
extractPart:useAsDelimiter:caseSensitive:
extractPart:useAsDelimiter:fromZone:
extractPart:useAsDelimiter:caseSensitive:fromZone:
- **extractPart:**(int)*n* **useAsDelimiter:**(char)*c* **caseSensitive:**(BOOL)*sense* **fromZone:**(NXZone *)*zone*

This method allows you to extract substring from strings which have fields delimited by a particular character. This is useful for getting at tab-delimited fields, entries from files like /etc/passwd (delimited by ª:º) and parts of UNIX paths. The first field is part number one; you can also use the constants MISC_STRING_FIRST and MISC_STRING_LAST to specify the first and last fields, respectively. The character used to delimit fields is specified by *c*. By setting *sense* to YES or NO, you can control whether or not the delimiter is case sensitive. A new MiscString is returned, allocated from *zone*. If the field specified does not exist (i.e. you specified field 7 when there are only 6 fields, etc.) then **nil** is returned. In the degenerate methods, *sense* defaults to YES and *zone* defaults to the receiver's zone.

See also: -**fileName**, -**fileNameFromZone:**, -**pathName**, -**pathNameFromZone:**, and -**wordNum:**

fgets:
fgets:keepNewline:

- (int)**fgets:**(FILE *)*fd* **keepNewline:**(BOOL)*keepit*
- (int)**fgets:**(FILE *)*fd*

Reads in one character at a time from the file *fd* until a newline or EOF character is reached, and sets receiver's ±**stringValue** to the resulting line of text. If *keepit* is YES, the newline character is left on at the end (though not the EOF character). In the degenerate method, *keepit* defaults to YES in keeping with the fgets() function. Returns EOF if that character is encountered, **0** otherwise. NOTE: In reading in the line, the receiver's allocated space is doubled each time it fills up. This can mean that the resulting string takes up substantially more space than is needed. If this is a problem, you can use ±**fixStringLength** to remove any excess allocated memory.

See also: -**gets** and -**streamGets:...**

fileBasename

- **fileBasename**

Returns a new MiscString which is the basename of the file represented by the receiver. All path elements and the last extension are stripped off, as determined by the current path and extension separators. The new MiscString is allocated from the receiver's zone.

See also: **-x**

fileBasenameFromZone:

- **fileBasenameFromZone:**(NXZone *)*zone*

Returns a new MiscString which is the basename of the file represented by the receiver. All path elements and the last extension are stripped off, as determined by the current path and extension separators. The new MiscString is allocated from *zone*.

See also: **-x**

fileExtension

- **fileExtension**

Returns a new MiscString which is the last extension of the file represented by the receiver. All path elements, the base name, and the all but the last extension are stripped off, as determined by the current path and extension separators. The new MiscString is allocated from the receiver's zone.

See also: **-x**

fileExtensionFromZone:

- **fileExtensionFromZone:**(NXZone *)*zone*

Returns a new MiscString which is the last extension of the file represented by the receiver. All path elements, the base name, and the all but the last extension are stripped off, as determined by the current path and extension separators. The new MiscString is allocated from *zone*.

See also: **-x**

fileName

- **fileName**

Same as the ±**fileNameFromZone:** method. The new MiscString is in the same zone as the receiver.

See also: **-extractPart:useAsDelimiter:...**, -**fileNameFromZone:**, -**pathName**, -**pathNameFromZone:**, and **-wordNum:**

fileNameFromZone:

- **fileNameFromZone:**(NXZone *)*zone*

Assuming that the receiving MiscString contains a UNIX path name of some sort, this method returns a new MiscString instance which contains the filename portion of a path name. This amounts to the part of the receiver from the character after the last ª/º to the end of the string.

See also: **-extractPart:useAsDelimiter:...**, -**fileName**, -**pathName**, -**pathNameFromZone:**, and **-wordNum:**

fixStringLength

- **fixStringLength**

Truncates the string buffer to the length of the string stored in the MiscString. Returns *self*.

See also: **-x**

fixStringLengthAt:

- **fixStringLengthAt:**(unsigned)*index*

Truncates the string buffer to *index*; if the MiscString contains a string longer than *index*, then it is truncated accordingly. Returns *self*.

See also: **-x**

floatValue

- (float)**floatValue**

Returns the floating point value of the receiver, if the receiving string begins with a properly formed floating point number. Any trailing non-numeric characters are ignored. If the receiving string doesn't begin with a number, a zero is returned.

See also: **-doubleValue** and **-intValue**

free

- **free**

Deallocates the MiscString and the contents of *buffer*.

freeString
- **freeString**

Frees the contents of *buffer* and sets it to NULL. Also sets the length of the MiscString to zero. Returns **self**.

getCopyInto:
- (char *)**getCopyInto:**(char *)*buf*

Copies the receiver's string buffer into the buffer *buf*. It is up to the programmer to be sure that *buf* is large enough to contain the complete contents of the receiver; if not, a memory violation will occur. Returns *buf.* If *buf* is NULL, a new buffer of the proper size will be created and returned instead of *buf* with the receiver's contents copied into it.

See also: **-x**

getIBImage
- (NXImage *)**getIBImage**

This method is used by InterfaceBuilder to obtain the image to display in the objects window when the MiscString has been dragged off of a palette. If the proper image is available, it is returned; otherwise *nil* is returned.

See also: **-x**

getInspectorClassName

- (const char *)**getInspectorClassName**

This method is used by InterfaceBuilder and returns the name of the MiscString's inspector class when it has been dragged off of an InterfaceBuilder palette.

See also: **-x**

gets

- (int)**gets**

Reads in one character at a time from the standard input until a newline or EOF character is reached, and sets receiver's ±**stringValue** to the resulting line of text. The newline character is not left on at the end, keeping with the gets() function. Returns EOF if that character is encountered, **0** otherwise. NOTE: In reading in the line, the receiver's allocated space is doubled each time it fills up. This can mean that the resulting string takes up substantially more space than is needed. If this is a problem, you can use ±**fixStringLength** to remove any excess allocated memory.

See also: **-fgets:...** and **-streamGets:...**

grep:
grep:caseSensitive:
grep:occurrenceNum:
grep:occurrenceNum:caseSensitive:
grep:before:middle:after:
grep:caseSensitive:before:middle:after:
grep:occurrenceNum:before:middle:after:

grep:occurrenceNum:caseSensitive:before:middle:after:

- (int)**grep:**(const char *)*pattern* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense* **before:***bstring* **middle:***mstring* **after:***astring*

Returns **1** if the n^{th} occurrence of the regular expression *pattern* is found in *buffer*, returns **0** if not, and returns **-1** if *pattern* is not a legal regular expression. If *sense* is NO, then the search ignores case. The receiver is split into three strings, the part before the match, the part that matched, and the part after the match. Each part is placed into *bstring*, *mstring*, and *astring* respectively, each of which should be an object which responds to **-setStringValue:**. If any of *bstring*, *mstring*, or *astring* are nil, then that portion is ignored. You may use the constants MISC_STRING_FIRST and MISC_STRING_LAST for *n* to specify the first and last matches, respectively. Usage of MISC_STRING_LAST may take a little longer because it calls **-rspotOfRegex:** to find the last part. This method leaves the receiving MiscString unchanged. In the degenerate methods, *sense* defaults to YES, *n* defaults to 0 (the first occurrence), and *bstring*, *mstring*, and *astring* all default to nil.

See also: **-matchesRegex:...**

grepString:
grepString:caseSensitive:
grepString:caseSensitive:before:middle:after:

- (int)**grepString:***pattern* **caseSensitive:**(BOOL)*caseSens* **before:***bstring* **middle:***mstring* **after:***astring*

Returns 1 if a portion of the receiver matches the regular expression *pattern*, stored in a MiscString, returns 0 if not, and returns -1 if *pattern* is not a legal regular expression. If *caseSens* is true, then the match is case sensitive; if false then the match is not case sensitive. The receiver is split into three strings, the part before the match, the part that matched, and the part after the match. Each part is placed into *bstring*, *mstring*, and *astring* respectively, each of which should be a MiscString (or subclass). If any of *bstring*, *mstring*, or *astring* are nil,

then that portion is ignored. This method leaves the receiving MiscString unchanged. In the degenerate methods, *sense* defaults to YES, and *bstring*, *mstring*, and *astring* all default to nil.

See also: **-grep:...**

hash

- (unsigned int)**hash**

Returns a hash value for the string contained by the MiscString, as determined by the NXStrHash() function.

See also: **-x**

hasType:

- (BOOL)**hasType:**(int)*type*

Returns YES if buffer contains any characters of type *type*, otherwise returns NO. The parameter *type* may be any of the following constants:

MISC_UPPER	an uppercase letter
MISC_LOWER	a lowercase letter
MISC_DIGIT	a digit
MISC_XDIGIT	a hexadecimal digit
MISC_PUNCT	a punctuation character (neither control nor alphanumeric)
MISC_ASCII	an ASCII character (code less than 0x7F)
MISC_CNTRL	a control character (0x00 through 0x1F, 0x7F, 0x80, 0xFE, 0xFF)
MISC_PRINT	a printing character
MISC_SPACE	a space, tab, carriage return, newline, vertical tab, or formfeed
MISC_GRAPH	a printing character; like MISC_PRINT except for space

Also, any bitwise OR (using ª|º) combination of the constants is permitted. MISC_ALPHA and MISC_ALNUM are predefined combinations that you may use.

See also: **-isAllOfType:**

index:
index:caseSensitive:
index:occurrenceNum:
index:occurrenceNum:caseSensitive:
- (const char *)**index:**(char)*aChar* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense*

Returns a pointer to the n^{th} occurrence of *aChar* in *buffer* going from left to right. If sense is NO, then the search ignores case. Returns NULL if the n^{th} occurrence of *aChar* is not found. Occurrences start numbering at zero, not one. In the degenerate methods, *sense* defaults to YES and *n* defaults to 0 (the first occurrence).

See also: **-rindex:...**

indexOfChars:
indexOfChars:caseSensitive:
indexOfChars:occurrenceNum:
indexOfChars:occurrenceNum:caseSensitive:
- (const char *)**indexOfChars:**(const char *)*aString* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense*

Returns a char pointer to the n^{th} occurrence in *buffer* of any of the characters in *aString* going from left to right. Use *n*=0 for the first occurrence. If *sense* is NO, then the search ignores case. Returns NULL if the n^{th}

occurrence is not found. In the degenerate methods, *sense* defaults to YES and *n* defaults to 0 (the first occurrence).

See also: **-rindexOfChars:**..., **-spotOfChars:**..., **-rspotOfChars:**..., **-index:**..., **-rindex:**..., **-spotOf:**..., and **-rspotOf:**...

init
- **init**

Initializes a new MiscString instance. Returns **self**.

See also: ±**initString:**

initCapacity:
initCapacity:fromZone:
- **initCapacity:**(int)*capacity* **fromZone:**(NXZone *)*zone*

Initializes the receiver with a buffer of size *capacity*, taken from the zone *zone*. In the degenerate method, *zone* defaults to the receiver's zone.

See also: **-x**

initDirectory:file:
- **initDirectory:**(const char *)*dir*
  file:(const char *)*file*

Initializes a new MiscString instance to represent the complete path to the file specified by *dir* and *file*.

See also: **-x**

initFromFormat:

- **initFromFormat:**(const char *)*formatStr, ...*

Initializes the receiver and sets it to the string generated from evaluating the format *formatStr* and subsequent arguments. Returns *self*.

See also: **-x**

initString:

- **initString:**(const char *)*aString*

This method calls the ±**init** method and then calls the ±**setStringValue** method with *aString* as the argument.

See also: ±**init**, -**setStringValue:**

insert:
insert:at:

- **insert:**(const char *)*aString* **at:***index*

Inserts *aString* into *buffer* at position *index.* Returns **self**. In the degenerate method, *index* defaults to 0 (the beginning of *buffer*).

See also: **-insertChar:...** and **-insertString:...**

insertAt:fromFormat:

- **insertAt:**(int)*index*
  fromFormat:(const char *)*format, ...*

Creates a new string using *format* and the subsequent arguments and inserts it into the receiver at *index*. Returns *self*.

See also: **-x**

insertAt:fromFormat:valist:

- **insertAt:**(int)*index*
  fromFormat:(const char *)*format*
  valist:(va_list)*param_list*

Same as ±**insertAt:fromFormat:** except that the argument types are different. Returns *self*.

See also: **-x**

insertChar:
insertChar:at:

- **insertChar:**(char)*aChar* **at:***index*

Inserts *aChar* into *buffer* at position *index.* *aChar* can not be 0 (null character); if this is the case, nothing happens. Returns **self**. In the degenerate method, *index* defaults to 0 (the beginning of *buffer*).

See also: **±insert:...** and **-insertString:...**

insertFromFormat:
- **insertFromFormat:**(const char *)*format, ...*

Same as calling ±**insertAt:FromFormat:** with 0 for *index*.

See also: **-x**

insertString:
insertString:at:
- **insertString:**(id)*sender* **at:***index*

Inserts the **±stringValue** of *sender* into *buffer* at position *index.* Returns **self**. In the degenerate method, *index* defaults to 0 (the beginning of *buffer*).

See also: ±**insert:...** and ±**insertChar:...**

intValue
- (int)**intValue**

Returns the integer value of the string; if the string is non-numeric or doesn't begin with a number, then a zero is returned. Any non-numeric characters following the number at the start of the string are ignored.

See also: **±doubleValue** and **-floatValue**

invertCases:

- **invertCases**

Inverts the case of every character in *buffer.* Returns **self**.

See also: **±toLower**, **-toUpper**

isAbsolutePath

- (BOOL)**isAbsolutePath**

If the receiving MiscString begins with the path element separator, then it is considered to be an absolute path and YES is returned. Otherwise a NO is returned.

See also: **-x**

isAllOfType:

- (int)**isAllOfType:**(int)*type*

Returns YES if buffer is entirely made up of characters of type *type*, else returns NO. The parameter *type* may be any of the following constants: MISC_UPPER, MISC_LOWER, MISC_DIGIT, MISC_XDIGIT, MISC_PUNCT, MISC_ASCII, MISC_CNTRL, MISC_PRINT, MISC_SPACE, or MISC_GRAPH. (See **-hasType:** for a description of these constants.) Also, any bitwise OR (using ª|º) combination of the constants is permitted. MISC_ALPHA and MISC_ALNUM are predefined combinations that you may use.

See also: **-hasType:**

isEqual:

- (BOOL)**isEqual:**(id)*anObject*

Returns YES if the string value of *anObject* is the same as the string value of the receiver.

isFileOfType:

- (BOOL)**isFileOfType:**(MiscFileType)*fileType*

Returns YES if the MiscString contains the path to an existing file which is of type *fileType*. See Types and Constants for a listing of valid file types.

See also: **-x**

isRelativePath

- (BOOL)**isRelativePath**

If the receiving MiscString doesn't begin with the path element separator, then it is considered to be a relative path and YES is returned. Otherwise a NO is returned.

See also: **-x**

isRTFText

- (BOOL)**isRTFText**

If the receiving MiscString contains RTF, as determined by the prefix ª{\rtf0º, then YES is returned. Otherwise a NO is returned.

See also: **-x**

left:
left:fromZone:
- **left:**(int)*count* **fromZone:**(NXZone *)*zone*

Returns a new MiscString object which is composed of the first *count* characters of *buffer*. The new object is allocated from *zone*. In the degenerate method, *zone* defaults to the receiver's zone.

See also: **-midFrom:length:...**, **-midFrom:to:...**, and **-right:...**

length
- (int)**length**

Returns the length of the string in *buffer*.

See also: **-emptyString**

loadFromFile:
- **loadFromFile:**(const char *)*aFileName*

Loads the contents of the file *aFileName* into the MiscString and returns **self**. If an error occurs, **nil** is returned and the receiver is left empty.

makeCaseInsensitiveSearchString

- **makeCaseInsensitiveSearchString**

Creates a new MiscString and converts it into a Sybase-compatible case-insensitive search string. Returns *self*.

See also: -**x**

matchesRegex:
matchesRegex:caseSensitive:

- (int)**matchesRegex:**(const char *)*pattern* **caseSensitive:**(BOOL)*sense*

Checks to see if *buffer* matches up directly with the regular expression *pattern*. If *sense* is NO, then the search ignores case. Returns **0** if it does not match, otherwise, returns the length of the matched portion. An error, such as an illegal regular expression, also returns **0**. This makes it possible to use this method as if it returned a BOOL. In the degenerate method, *sense* defaults to YES.

See also: -**grep:...** and -**spotOfRegex:...**

midFrom:length:
midFrom:length:fromZone:

- **midFrom:**(int)*start* **length:**(int)*len* **fromZone:**(NXZone *)*zone*

Returns a new MiscString object which is composed of *len* characters of *buffer* starting with the *start*th character. The new object is allocated from *zone*. In the degenerate method, *zone* defaults to the receiver's zone.

See also: -**left:...**, -**midFrom:to:...**, and -**right:...**

midFrom:to:
midFrom:to:fromZone:
- **midFrom:**(int)*start* **to:**(int)*end* **fromZone:**(NXZone *)*zone*

Returns a new MiscString object which is composed of the characters of *buffer* from the *start*th character to the *end*th character inclusive. The new object is allocated from *zone*. In the degenerate method, *zone* defaults to the receiver's zone.

See also: **-left:...**, **-midFrom:length:**, and **-right:...**

new
- **new**

Returns a new, empty MiscString object of the same class as the receiver.

nthQuotedField
- **nthQuotedField:**(int)*fieldNumber*

Returns a new MiscString which contains the contents of the *fieldNumber*th quoted section. Nested quotes will confuse the parsing since deciding whether or not a section of the string is quoted is determined by whether the opening quotation mark is the nth quotation mark where *n* is odd. Numbering starts at zero.

See also: **-x**

numberOfPathComponents

- (int)**numberOfPathComponents**

Returns the number of path components in the MiscString, assuming that is contains a filename and path, using the current path component separators.

See also: **-x**

numOf:
numOf:caseSensitive:
numOf:overlap:
numOf:caseSensitive:overlap:

- **numOf:**(const char *)*aString* **caseSensitive:**(BOOL)*sense* **overlap:**(BOOL)*overlap*

Returns the number of times the string *aString* occurs in *buffer*. If *sense* is NO, then the search ignores case. If *overlap* is YES, then the search can match overlapping parts of the receiving string. For example, if you search for ªabc abcº in the string ªabc abc abcº, with *overlap* YES you will get two matches and with *overlap* NO you will get one match. In the degenerate methods, *sense* defaults to YES and *overlap* defaults to NO.

See also: **-x**

numOfChar:
numOfChar:caseSensitive:

- (int)**numOfChar:**(char)*aChar* **caseSensitive:**(BOOL)*sense*

Returns the number of times the given character occurs in *buffer*. If *sense* is NO, then the search ignores case. In the degenerate method, *sense* defaults to YES.

See also: -**numOfChars:...**

numOfChars:
numOfChars:caseSensitive:
- (int)**numOfChars:**(const char *)*aString* **caseSensitive:**(BOOL)*sense*

Returns the number of times any character in *aString* occurs in *buffer*. If *sense* is NO, then the search ignores case. In the degenerate method, *sense* defaults to YES.

See also: -**numOfChar:...**

numOfRegex:
numOfRegex:caseSensitive:
- (int)**numOfRegex:**(const char *)*pattern* **caseSensitive:**(BOOL)*sense*

Returns the number of times the regular expression pattern is matched in buffer. Returns **-1** if some kind of error occurred, such as an illegal regular expression. In the degenerate method, *sense* defaults to YES.

See also: -**replaceEveryOccurrenceOfRegex:...** and -**grep:...**

numOfString:
numOfString:caseSensitive:
numOfString:overlap:
numOfString:caseSensitive:overlap:
- **numOfString:**(id)*sender* **caseSensitive:**(BOOL)*sense* **overlap:**(BOOL)*overlap*

Returns the number of times the **±stringValue** of *sender* occurs in *buffer*. If *sense* is NO, then the search ignores case. If *overlap* is YES, then the search can match overlapping parts of the receiving string. For example, if you search for ªabc abcº in the string ªabc abc abcº, with *overlap* YES you will get two matches and with *overlap* NO you will get one match. In the degenerate methods, *sense* defaults to YES and *overlap* defaults to NO.

See also: **-x**

numOfType:

- **numOfType:**(int)*type*

Returns the number of times any character of type *type* occurs in *buffer*. The parameter *type* may be any of the following constants: MISC_UPPER, MISC_LOWER, MISC_DIGIT, MISC_XDIGIT, MISC_PUNCT, MISC_ASCII, MISC_CNTRL, MISC_PRINT, MISC_SPACE, or MISC_GRAPH. (See **-hasType:** for a description of these constants.) Also, any bitwise OR (using ª|º) combination of the constants is permitted. MISC_ALPHA and MISC_ALNUM are predefined combinations that you may use.

See also: **-hasType:**

numWords

- (int)**numWords**

Returns the number of words in *buffer*. Words are separated by any number of spaces, carriage returns, newlines, vertical tabs, or formfeeds (not punctuation characters).

See also: **-wordNum:**

padFrontToLength:withChar:

- **padFrontToLength:**(int)*len* **withChar:**(char)*aChar*

This method pads the beginning of *buffer* to length *len*, using *aChar* to fill in any extra spaces needed. If *len* is less than the length of *buffer*, nothing happens. Returns **self**.

See also: **-x**

padToLength:withChar:

- **padToLength:**(int)*len* **withChar:**(char)*aChar*

This method pads *buffer* to length *len*, using *aChar* to fill in any extra spaces needed. If *len* is less than the length of *buffer*, nothing happens. Returns **self**.

See also: **-x**

pathComponentAt:

- **pathComponentAt:**(int)*index*

Returns a MiscString containing the *index*th path component; the first component is number zero.

See also: **-x**

pathName

- **pathName**

Same as the ±**pathNameFromZone:** method with the returned MiscString coming from the receiver's zone.

See also: **-extractPart:...**, **-fileName**, **-fileNameFromZone:**, **-pathNameFromZone:**, and **-wordNum:**

pathNameFromZone:

- **pathNameFromZone:**(NXZone *)*zone*

Assuming that the receiving MiscString contains a UNIX path name of some sort, this method returns a new MiscString instance which contains the path portion of a path name. This amounts to the part of the receiver from the start of the string up to, but not including, the last ª/º.

See also: **-extractPart:useAsDelimiter:...**, **-fileName**, **-fileNameFromZone:**, **-pathName**, and **-wordNum:**

plainTextForRTF

- (MiscString *)**plainTextForRTF**

If the MiscString contains RTF, this will return a plain ASCII string converted from the RTF. If the MiscString contains ASCII text already, that is returned. This method returns a new string which you must free yourself.

See also: **-x**

printForDebugger:

- **printForDebugger:**(NXStream *)*stream*

Prints information about the receiver onto *stream*. The data printed is identical in content and format to that produced by ±**buildInstanceImageIn:**.

See also: **-buildInstanceImageIn:**, and ± **printToStdErr:**

printToStdErr:

- **printToStdErr:**(const char *)*label*

Prints a line of information about the receiver to stderr with the label *label* and the class and id of the receiver prepended to the output. The data printed is identical in content and format to that produced by ±**buildInstanceImageIn:**.

See also: ±**buildInstanceImageIn**, and **-printForDebugger:**

read:

- **read:**(NXTypedStream *)*stream*

Reads the MiscString from the typed stream *stream*. Returns **self**.

See also: **- write:**

recalcLength

- **recalcLength**

Recalculates the length of the string contained by the MiscString. Under normal circumstances, this should be unnecessary as any message sent to the MiscString which alters the length of the string will update the cached string length. Returns **self**.

See also: **-fixStringLength**, and ±**fixStringLengthAt:**

removeFrom:length:
- **removeFrom:**(int)*start* **length:**(int)*len*

Removes *len* characters from *buffer* starting with the *start*th character. Will not take any action if *len* is zero or if *start* is out of range. Returns **self**.

See also: - **removeFrom:to:**

removeFrom:to:
- **removeFrom:**(int)*start* **to:**(int)*end*

Removes all the characters of *buffer* from the *start*th character to the *end*th character inclusive. Will not take any action if *start* or *end* is out of range. Returns **self**.

See also: - **removeFrom:length:**

replace:with:
replace:with:caseSensitive:
replace:with:occurrenceNum:
replace:with:occurrenceNum:caseSensitive:
replace:with:overlap:
replace:with:caseSensitive:overlap:
replace:with:occurrenceNum:overlap:
replace:with:occurrenceNum:caseSensitive:overlap:

replace:withChar:
replace:withChar:caseSensitive:
replace:withChar:occurrenceNum:
replace:withChar:occurrenceNum:caseSensitive:
replace:withChar:overlap:
replace:withChar:caseSensitive:overlap:
replace:withChar:occurrenceNum:overlap:
replace:withChar:occurrenceNum:caseSensitive:overlap:
replace:withString:
replace:withString:caseSensitive:
replace:withString:occurrenceNum:
replace:withString:occurrenceNum:caseSensitive:
replace:withString:overlap:
replace:withString:caseSensitive:overlap:
replace:withString:occurrenceNum:overlap:
replace:withString:occurrenceNum:caseSensitive:overlap:

- **replace:**(const char *)*aString* **with:**(const char *)*repl* **occurrenceNum:**(int)*n* **caseSensitive:** (BOOL)*sense* **overlap:**(BOOL)*overlap*
- **replace:**(const char *)*aString* **withChar:**(char)*aChar* **occurrenceNum:**(int)*n* **caseSensitive:** (BOOL)*sense* **overlap:**(BOOL)*overlap*
- **replace:**(const char *)*aString* **withString:**(id)*sender* **occurrenceNum:**(int)*n* **caseSensitive:** (BOOL)*sense* **overlap:**(BOOL)*overlap*

Replaces the n^{th} occurrence of the string *aString* with either *repl*, *aChar*, or the **-stringValue** of *sender*. If *sense* is NO, the search ignores case. If *overlap* is YES, then the search can match overlapping parts of the receiving string. For example, if you search for ªabc abcº in the string ªabc abc abcº, with *overlap* YES you will get two occurrences and with *overlap* NO you will get one occurrence. Use *n*=0 for the first occurrence. You may use the constants MISC_STRING_FIRST and MISC_STRING_LAST for *n* to specify the first and last matches,

respectively. Usage of MISC_STRING_LAST may take a little longer because it calls **-rspotOfStr:** to find the last part. If the *n*th occurrence of the search string is not found, then nothing happens. Returns **self**. In the degenerate methods, *n* defaults to 0, *sense* defaults to YES, and *overlap* defaults to NO.

See also: **-x**

replaceCharAt:withChar:
- **replaceCharAt:**(int)*index* **withChar:**(char)*aChar*

Replaces the *index*th character of *buffer* with *aChar*. *aChar* can not be 0 (null character); if this is the case, nothing happens. Returns **self**.

replaceEveryOccurrenceOf:with:
replaceEveryOccurrenceOf:with:overlap:
replaceEveryOccurrenceOf:with:caseSensitive:
replaceEveryOccurrenceOf:with:caseSensitive:overlap:
replaceEveryOccurrenceOf:withChar:
replaceEveryOccurrenceOf:withChar:overlap:
replaceEveryOccurrenceOf:withChar:caseSensitive:
replaceEveryOccurrenceOf:withChar:caseSensitive:overlap:
replaceEveryOccurrenceOf:withString:
replaceEveryOccurrenceOf:withString:overlap:
replaceEveryOccurrenceOf:withString:caseSensitive:
replaceEveryOccurrenceOf:withString:caseSensitive:overlap:
- (int)**replaceEveryOccurrenceOf:**(const char *)*aString* **with:**(const char *)*repl* **caseSensitive:**(BOOL)*sense* **overlap:**(BOOL)*overlap*

- (int)**replaceEveryOccurrenceOf:**(const char *)*aString* **withChar:**(char)*aChar* **caseSensitive:**(BOOL)*sense* **overlap:**(BOOL)*overlap*
- (int)**replaceEveryOccurrenceOf:**(const char *)*aString* **withString:**(id)*sender* **caseSensitive:**(BOOL)*sense* **overlap:**(BOOL)*overlap*

Replaces each occurrence of the string *aString* with either *repl*, *aChar*, or the **-stringValue** of *sender*. If *sense* is NO, then the search ignores case. If *overlap* is YES, then the search can match overlapping parts of the receiving string. For example, if you try to replace ªabcabcº with ªpdqº in the string ªabcabcabc abcabcº, with *overlap* YES you will get ªpdqpdq pdqº and with *overlap* NO you will get ªpdqabc pdqº. Returns the number of replacements made. In the degenerate methods, *sense* defaults to YES and *overlap* defaults to NO.

See also: **-x**

replaceEveryOccurrenceOfString:with:
replaceEveryOccurrenceOfString:with:overlap:
replaceEveryOccurrenceOfString:with:caseSensitive:
replaceEveryOccurrenceOfString:with:caseSensitive:overlap:
replaceEveryOccurrenceOfString:withChar:
replaceEveryOccurrenceOfString:withChar:overlap:
replaceEveryOccurrenceOfString:withChar:caseSensitive:
replaceEveryOccurrenceOfString:withChar:caseSensitive:overlap:
replaceEveryOccurrenceOfString:withString:
replaceEveryOccurrenceOfString:withString:overlap:
replaceEveryOccurrenceOfString:withString:caseSensitive:
replaceEveryOccurrenceOfString:withString:caseSensitive:overlap:
- (int)**replaceEveryOccurrenceOfString:**(id)*aSender* **with:**(const char *)*repl* **caseSensitive:**(BOOL)*sense* **overlap:**(BOOL)*overlap*

- (int)**replaceEveryOccurrenceOfString:**(id)*aSender* **withChar:**(char)*aChar* **caseSensitive:**(BOOL)*sense* **overlap:**(BOOL)*overlap*
- (int)**replaceEveryOccurrenceOfString:**(id)*aSender* **withString:**(id)*sender* **caseSensitive:**(BOOL)*sense* **overlap:**(BOOL)*overlap*

Replaces each occurrence of the **-stringValue** of *aSender* with either *repl*, *aChar*, or the **-stringValue** of *sender*. If *sense* is NO, then the search ignores case. If *overlap* is YES, then the search can match overlapping parts of the receiving string. For example, if you try to replace ªabcabcº with ªpdqº in the string ªabcabcabc abcabcº, with *overlap* YES you will get ªpdqpdq pdqº and with *overlap* NO you will get ªpdqabc pdqº. Returns the number of replacements made. In the degenerate methods, *sense* defaults to YES and *overlap* defaults to NO.

See also: **-x**

replaceEveryOccurrenceOfChar:with:
replaceEveryOccurrenceOfChar:with:caseSensitive:
replaceEveryOccurrenceOfChar:withChar:
replaceEveryOccurrenceOfChar:withChar:caseSensitive:
replaceEveryOccurrenceOfChar:withString:
replaceEveryOccurrenceOfChar:withString:caseSensitive:

- **replaceEveryOccurrenceOfChar:**(char)*aChar* **with:**(const char *)*aString* **caseSensitive:**(BOOL)*sense*
- **replaceEveryOccurrenceOfChar:**(char)*aChar* **withChar:**(char)*replaceChar* **caseSensitive:**(BOOL)*sense*
- **replaceEveryOccurrenceOfChar:**(char)*aChar* **withString:**(id)*sender* **caseSensitive:**(BOOL)*sense*

Replaces each occurrence of *aChar* within *buffer* with either *aString*, *replaceChar*, or the **-stringValue** of *sender*. If *sense* is NO, the search will ignore case--all upper and lower case versions of *aChar* will be replaced. If *replaceChar* is the NULL character, returns **nil**, otherwise, returns **self**. In the degenerate methods, *sense* defaults to YES.

See also: **-replaceEveryOccurrenceOfChars:...**

replaceEveryOccurrenceOfChars:with:
replaceEveryOccurrenceOfChars:with:caseSensitive:
replaceEveryOccurrenceOfChars:withChar:
replaceEveryOccurrenceOfChars:withChar:caseSensitive:
replaceEveryOccurrenceOfChars:withString:
replaceEveryOccurrenceOfChars:withString:caseSensitive:

- **replaceEveryOccurrenceOfChars:**(const char *)*aString* **with:**(const char *)*replaceString* **caseSensitive:** (BOOL)*sense*
- **replaceEveryOccurrenceOfChars:**(const char *)*aString* **withChar:**(char)*replaceChar* **caseSensitive:** (BOOL)*sense*
- **replaceEveryOccurrenceOfChars:**(const char *)*aString* **withString:**(id)*sender* **caseSensitive:** (BOOL)*sense*

Replaces each occurrence of any character in *aString* within *buffer* with either *replaceString*, *replaceChar*, or the **-stringValue** of *sender*. If *sense* is NO, the search will ignore case--all upper and lower case versions of characters in *aString* will be replaced. If *replaceChar* is the NULL character, returns **nil**, otherwise, returns **self**. In the degenerate methods, *sense* defaults to YES.

See also: **-replaceEveryOccurrenceOfChar:...**

replaceEveryOccurrenceOfRegex:with:
replaceEveryOccurrenceOfRegex:with:caseSensitive:
replaceEveryOccurrenceOfRegex:withChar:

replaceEveryOccurrenceOfRegex:withChar:caseSensitive:
replaceEveryOccurrenceOfRegex:withString:
replaceEveryOccurrenceOfRegex:withString:caseSensitive:

- (int)**replaceEveryOccurrenceOfRegex:**(const char *)*pattern* **with:**(const char *)*replacement* **caseSensitive:**(BOOL)*sense*
- (int)**replaceEveryOccurrenceOfRegex:**(const char *)*pattern* **withChar:**(char)*aChar* **caseSensitive:**(BOOL)*sense*
- (int)**replaceEveryOccurrenceOfRegex:**(const char *)*pattern* **withString:**(id)*sender* **caseSensitive:**(BOOL)*sense*

Replaces every occurrence of the regular expression *pattern* with either *replacement*, *aChar*, or the **-stringValue** of *sender*. If *sense* is NO, the search ignores case. If it is an illegal regular expression, then nothing happens and -1 gets returned. Otherwise, returns the number of times a match was replaced. In the degenerate methods, *sense* defaults to YES.

See also: **-replaceRegex:...** and **-grep:...**

replaceEveryOccurrenceOfType:with:
replaceEveryOccurrenceOfType:withChar:
replaceEveryOccurrenceOfType:withString:

- **replaceEveryOccurrenceOfType:**(int)*type* **with:**(const char *)*aString*
- **replaceEveryOccurrenceOfType:**(int)*type* **withChar:**(char)*aChar*
- **replaceEveryOccurrenceOfType:**(int)*type* **withString:**(id)*sender*

Replaces each occurrence of any character of type *type* with either *aString*, *aChar*, or the **±stringValue** of *sender*. The parameter *type* may be any of the following constants: MISC_UPPER, MISC_LOWER, MISC_DIGIT, MISC_XDIGIT, MISC_PUNCT, MISC_ASCII, MISC_CNTRL, MISC_PRINT, MISC_SPACE, or

MISC_GRAPH. (See **-hasType:** for a description of these constants.) Also, any bitwise OR (using ª|º) combination of the constants is permitted. MISC_ALPHA and MISC_ALNUM are predefined combinations that you may use. Returns **self**.

See also: **-hasType:**

replaceFrom:length:with:
replaceFrom:length:withChar:
replaceFrom:length:withString:
- **replaceFrom:**(int)*start* **length:**(int)*len* **with:**(const char *)*aString*
- **replaceFrom:**(int)*start* **length:**(int)*len* **withChar:**(char)*aChar*
- **removeFrom:**(int)*start* **length:**(int)*end* **withString:**(id)*sender*

Replaces *len* characters from *buffer,* starting with the *start*th character, with either *aString*, *aChar*, or the **-stringValue** of *sender*. Returns **self**. Will not take any action if *len* is zero or if *start* is out of range.

See also: **- replaceFrom:to:...**

replaceFrom:to:with:
replaceFrom:to:withChar:
replaceFrom:to:withString:
- **replaceFrom:**(int)*start* **to:**(int)*end* **with:**(const char *)*aString*
- **replaceFrom:**(int)*start* **to:**(int)*end* **withChar:**(char)*achar*
- **removeFrom:**(int)*start* **to:**(int)*end* **withString:**(id)*sender*

Replaces all the characters of *buffer,* from the *start*th character to the *end*th character inclusive, with either

aString, *aChar*, or the **-stringValue** of *sender*. Returns **self**. Will not take any action if *start* or *end* is out of range.

See also: **- replaceFrom:length:with:...**

replaceHomeWithTilde

- replaceHomeWithTilde

If the beginning of *buffer* matches the string returned by NXHomeDirectory(), that portion of the string is replaced with the character '~'. Failing this, setpwent(),getpwent(), and endpwent() are used to see if another user's home directory matches the beginning of *buffer*. If a match is found, that portion of the string is replaced with "~username". Care is taken to avoid matching a portion of a directory name (so there are no problems when one user's username is a prefix of another, like "tom" and "tomsmith"). Also, no action is taken if the home directory being checked is root's ("/"). Returns **self**.

Examples:

"/Users/joe"	-->	"~" (if current user is joe)
"/Users/joe/"	-->	"~/" (if current user is joe)
"/Users/joey/"	-->	"~joey/" (if current user is joe, and not joey)
"/"	-->	"/" (if current user is root)
"/LocalLibrary"	-->	"/LocalLibrary" (if current user is root)
"/Users/juser"	-->	"~juser"
"/Users/juser2/"	-->	"~juser2/"

See also: **-replaceTildeWithHome**

replacePattern:caseSensitive:globally:with:

- (int)**replacePattern:**(const char *)*pattern*
  caseSensitive:(BOOL)*caseSens*
  globally:(BOOL)*glob*
  with:(const char *)*replacement*

Searches the MiscString for segments which match the regular expression *pattern* and replaces them with *replacement*. The search is case sensitive if *caseSens* is true, otherwise it is not case sensitive. If *glob* is true, then all matches will be replaced; if *glob* is false, then only the first match will be replaced. *This method is now considered obsolete. You should use a* ±***replaceRegex:*** *variant instead.*

See also: **-x**

replacePattern:caseSensitive:globally:withString:

- (int)**replacePattern:**(const char *)*pattern*
  caseSensitive:(BOOL)*caseSens*
  globally:(BOOL)*glob*
  withString:*replacement*

Searches the MiscString for segments which match the regular expression *pattern* and replaces them with the MiscString *replacement*. The search is case sensitive if *caseSens* is true, otherwise it is not case sensitive. If *glob* is true, then all matches will be replaced; if *glob* is false, then only the first match will be replaced. *This method is now considered obsolete. You should use a* ±***replaceRegex:*** *variant instead.*

See also: **-x**

replacePatternString:caseSensitive:globally:with:

- (int)**replacePatternString:***patternL*
  caseSensitive:(BOOL)*caseSens*
  globally:(BOOL)*glob*
  with:(const char *)*replacement*

Searches the MiscString for segments which match the regular expression in the MiscString *pattern* and replaces them with *replacement*. The search is case sensitive if *caseSens* is true, otherwise it is not case sensitive. If *glob* is true, then all matches will be replaced; if *glob* is false, then only the first match will be replaced. *This method is now considered obsolete. You should use a* ±***replaceRegex:*** *variant instead.*

See also: **-x**

replacePatternString:caseSensitive:globally:withString:

- (int)**replacePatternString:***pattern*
  caseSensitive:(BOOL)*caseSens*
  globally:(BOOL)*glob*
  withString:*replacement*

Searches the MiscString for segments which match the regular expression in the MiscString *pattern* and replaces them with the MiscString *replacement*. The search is case sensitive if *caseSens* is true, otherwise it is not case sensitive. If *glob* is true, then all matches will be replaced; if *glob* is false, then only the first match will be replaced. *This method is now considered obsolete. You should use a* ±***replaceRegex:*** *variant instead.*

See also: **-x**

replaceRegex:with:
replaceRegex:with:caseSensitive:

replaceRegex:with:occurrenceNum:
replaceRegex:with:occurrenceNum:caseSensitive:
replaceRegex:withChar:
replaceRegex:withChar:caseSensitive:
replaceRegex:withChar:occurrenceNum:
replaceRegex:withChar:occurrenceNum:caseSensitive:
replaceRegex:withString:
replaceRegex:withString:caseSensitive:
replaceRegex:withString:occurrenceNum:
replaceRegex:withString:occurrenceNum:caseSensitive:

- **replaceRegex:**(const char *)*pattern* **with:**(const char *)*replacement* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense*
- **replaceRegex:**(const char *)*pattern* **withChar:**(char)*aChar* **occurrenceNum:**(int)*n* **caseSensitive:** (BOOL)*sense*
- **replaceRegex:**(const char *)*pattern* **withString:**(id)*sender* **occurrenceNum:**(int)*n* **caseSensitive:** (BOOL)*sense*

Replaces the n^{th} occurrence of the regular expression *pattern* with either *replacement*, *aChar*, or the **-stringValue** of *sender*. If *sense* is NO, the search ignores case. If the n^{th} occurrence of *pattern* is not found, then nothing happens. Returns **self**. In the degenerate methods, *sense* defaults to YES and *n* defaults to 0 (the first occurrence).

See also: **-replaceEveryOccurrenceOfRegex:..., -grep:...**

replaceString:with:
replaceString:with:caseSensitive:
replaceString:with:occurrenceNum:

replaceString:with:occurrenceNum:caseSensitive:
replaceString:with:overlap:
replaceString:with:caseSensitive:overlap:
replaceString:with:occurrenceNum:overlap:
replaceString:with:occurrenceNum:caseSensitive:overlap:
replaceString:withChar:
replaceString:withChar:caseSensitive:
replaceString:withChar:occurrenceNum:
replaceString:withChar:occurrenceNum:caseSensitive:
replaceString:withChar:overlap:
replaceString:withChar:caseSensitive:overlap:
replaceString:withChar:occurrenceNum:overlap:
replaceString:withChar:occurrenceNum:caseSensitive:overlap:
replaceString:withString:
replaceString:withString:caseSensitive:
replaceString:withString:occurrenceNum:
replaceString:withString:occurrenceNum:caseSensitive:
replaceString:withString:overlap:
replaceString:withString:caseSensitive:overlap:
replaceString:withString:occurrenceNum:overlap:
replaceString:withString:occurrenceNum:caseSensitive:overlap:

- **replaceString:**(id)*aSender* **with:**(const char *)*repl* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense* **overlap:**(BOOL)*overlap*
- **replaceString:**(id)*aSender* **withChar:**(char)*aChar* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense* **overlap:**(BOOL)*overlap*
- **replaceString:**(id)*aSender* **withString:**(id)*sender* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense* **overlap:**(BOOL)*overlap*

Replaces the *n*th occurrence of the **-stringValue** of *aSender* with either *repl*, *aChar*, or the **-stringValue** of *sender*. If *sense* is NO, the search ignores case. If *overlap* is YES, then the search can match overlapping parts of the receiving string. For example, if you search for ªabc abcº in the string ªabc abc abcº, with *overlap* YES you will get two occurrences and with *overlap* NO you will get one occurrence. Use *n*=0 for the first occurrence. You may use the constants MISC_STRING_FIRST and MISC_STRING_LAST for *n* to specify the first and last matches, respectively. Usage of MISC_STRING_LAST may take a little longer because it calls **-rspotOfString:** to find the last part. If the *n*th occurrence of the search string is not found, then nothing happens. Returns **self**. In the degenerate methods, *n* defaults to 0, *sense* defaults to YES, and *overlap* defaults to NO.

See also: **-x**

replaceType:with:
replaceType:with:occurrenceNum:
replaceType:withChar:
replaceType:withChar:occurrenceNum:
replaceType:withString:
replaceType:withString:occurrenceNum:

- **replaceType:**(int)*type* **with:**(const char *)*aString* **occurrenceNum:**(int)*n*
- **replaceType:**(int)*type* **withChar:**(char)*aChar* **occurrenceNum:**(int)*n*
- **replaceType:**(int)*type* **withString:**(id)*sender* **occurrenceNum:**(int)*n*

Replaces the *n*th occurrence of any character of type *type* with either *aString*, *aChar*, or the **-stringValue** of *sender*. If *sense* is NO, the search ignores case. Use *n*=0 for the first occurrence. If the *n*th occurrence of *type* is not found, then nothing happens. The parameter *type* may be any of the following constants: MISC_UPPER, MISC_LOWER, MISC_DIGIT, MISC_XDIGIT, MISC_PUNCT, MISC_ASCII, MISC_CNTRL, MISC_PRINT, MISC_SPACE, or MISC_GRAPH. (See **-hasType:** for a description of these constants.) Also,

any bitwise OR (using ª|º) combination of the constants is permitted. MISC_ALPHA and MISC_ALNUM are predefined combinations that you may use. Returns **self**. In the degenerate methods, *n* defaults to 0.

See also: **-x**

replaceTildeWithHome
- **replaceTildeWithHome**

If *buffer* is "~", it gets replaced with the string returned by NXHomeDirectory(). If *buffer* starts with "~/", the '~' is replaced by the string returned by NXHomeDirectory(). If the user is root (i.e., NXHomeDirectory() returns "/"), the '~' is simply removed, leaving '/' as the first character. Lastly, if *buffer* starts with "~uname/etc/etc", and 'uname' is a valid username, the "~uname" portion is replaced with that user's home directory. This method will do nothing if the first character of *buffer* is not '~'. Returns **self**.

Examples:

"~"	-->	"/Users/joe" (if current user is joe)
"~"	-->	"/" (if current user is root)
"~/"	-->	"/" (if current user is root)
"~/"	-->	"/Users/joe/" (if current user is joe)
"~news"	-->	"/usr/spool/news"
"~invalidusername"	-->	"~invalidusername"
"~root/usr"	-->	"//usr" (this does work)
"~juser/Library"	-->	"/Users/juser/Library"

See also: **-replaceHomeWithTilde**

reverse
- **reverse**

Reverses the characters in *buffer*. Returns **self**.

right:
right:fromZone:
- **right:**(int)*count* **fromZone:**(NXZone *)*zone*

Returns a new MiscString object which is composed of the last *count* characters of *buffer*. The new object is allocated from *zone*. In the degenerate method, *zone* defaults to the receiver's zone.

See also: **-left:...**, **-midFrom:length:...**, and **-midFrom:to:...**

rindex:
rindex:caseSensitive:
rindex:occurrenceNum:
rindex:occurrenceNum:caseSensitive:
- (const char *)**rindex:**(char)*aChar* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense*

Returns a pointer to the n^{th} occurrence of *aChar* in *buffer* going from right to left. If sense is NO, then the search ignores case. Returns NULL if the n^{th} occurrence of *aChar* is not found. Occurrences start numbering at zero, not one. In the degenerate methods, *sense* defaults to YES and *n* defaults to 0 (the first occurrence).

See also: **-index:...**

rindexOfChars:
rindexOfChars:caseSensitive:
rindexOfChars:occurrenceNum:
rindexOfChars:occurrenceNum:caseSensitive:

- (const char *)**rindexOfChars:**(const char *)*aString* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense*

Returns a char pointer to the n^{th} occurrence in *buffer* of any of the characters in *aString* going from right to left. Use *n*=0 for the first occurrence. If *sense* is NO, then the search ignores case. Returns NULL if the n^{th} occurrence is not found. In the degenerate methods, *sense* defaults to YES and *n* defaults to 0 (the first occurrence).

See also: **-indexOfChars:**..., **-spotOfChars:**..., **-rspotOfChars:**..., **-index:**..., **-rindex:**..., **-spotOf:**..., and **-rspotOf:**...

rspotOf:
rspotOf:caseSensitive
rspotOf:occurrenceNum:
rspotOf:occurrenceNum:caseSensitive:

- (int)**rspotOf:**(char)*aChar* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense*

Returns the position number of the n^{th} occurrence of *aChar* in *buffer* going from right to left. If *sense* is NO, then the search ignores case. Returns **-1** if the n^{th} occurrence of *aChar* is not found. Occurrences start numbering at zero, not one. In the degenerate methods, *sense* defaults to YES and *n* defaults to 0 (the first occurrence).

See also: **-spotOf:...**

rspotOfChars:
rspotOfChars:caseSensitive:
rspotOfChars:occurrenceNum:
rspotOfChars:occurrenceNum:caseSensitive:

- (int)**rspotOfChars:**(const char *)*aString* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense*

Returns the position number of the *n*th occurrence in *buffer* of any of the characters in *aString* going from right to left. Use *n*=0 for the first occurrence. If *sense* is NO, then the search ignores case. Returns **-1** if the *n*th occurrence is not found. In the degenerate methods, *sense* defaults to YES and *n* defaults to 0 (the first occurrence).

See also: **-spotOfChars:...**, and **-rspotOfChars:...**

rspotOfRegex:
rspotOfRegex:caseSensitive:
rspotOfRegex:occurrenceNum:
rspotOfRegex:occurrenceNum:caseSensitive:
rspotOfRegex:length:
rspotOfRegex:caseSensitive:length:
rspotOfRegex:occurrenceNum:length:
rspotOfRegex:occurrenceNum:caseSensitive:length:

- (int)**rspotOfRegex:**(const char *)*pattern* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense*
  length:(int *)*matchlen*

Returns the position number of the *n*th occurrence of the regular expression *pattern* in *buffer* going from right to left. This method works by first calling **-numOfRegex:** to find out how many times pattern occurs, then uses **-spotOfRegex:** to pluck out the correct part. The primary advantage of this is that it is symmetrical with

-spotOfRegex: -- the last part able to be plucked out by that method will be the first part returned with this method. However, this also means that this method is slower, since it must make an extra pass through the string. Use n=0 for the first occurrence. If *sense* is NO, then the search ignores case. Returns **-1** if the n^{th} occurrence is not found, and **-2** on an illegal regular expression. If not NULL, the integer pointed to by *matchlen* gets set to the length of the matched portion. In the degenerate methods, *sense* defaults to YES, *n* defaults to 0, and *matchlen* defaults to NULL.

See also: **-grep:...** and **-matchesRegex:...**

rspotOfStr:
rspotOfStr:caseSensitive:
rspotOfStr:occurrenceNum:
rspotOfStr:occurrenceNum:caseSensitive:
rspotOfStr:overlap:
rspotOfStr:caseSensitive:overlap:
rspotOfStr:occurrenceNum:overlap:
rspotOfStr:occurrenceNum:caseSensitive:overlap:

- **rspotOfStr:**(const char *)*aString* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense* **overlap:**(BOOL)*overlap*

Returns the position number of the n^{th} occurrence in *buffer* of the string *aString* going from right to left. Use *n*=0 for the first occurrence. Returns **-1** if the n^{th} occurrence is not found. If *sense* is NO, then the search ignores case. If *overlap* is YES, then the search can match overlapping parts of the receiving string. For example, if you search for ªabc abcº in the string ªabc abc abcº, with *overlap* YES you will get two occurrences and with *overlap* NO you will get one occurrence. This method works by first calling **-numOf:** to find out how many times the search string occurs, then uses **-spotOfStr:** to pluck out the correct part. The primary advantage of this is that it is symmetrical with **-spotOfStr:** -- the last part able to be plucked out by that method will be the first part

returned with this method. However, this also means that this method is slower, since it must make an extra pass through the string. In the degenerate methods, *n* defaults to 0, *sense* defaults to YES, and *overlap* defaults to NO.

See also: **-x**

rspotOfString:
rspotOfString:caseSensitive:
rspotOfString:occurrenceNum:
rspotOfString:occurrenceNum:caseSensitive:
rspotOfString:overlap:
rspotOfString:caseSensitive:overlap:
rspotOfString:occurrenceNum:overlap:
rspotOfString:occurrenceNum:caseSensitive:overlap:

- (int)**rspotOfString:**(id)*sender* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense* **overlap:** (BOOL)*overlap*

Returns the position number of the n^{th} occurrence in *buffer* of the **±stringValue** of *sender* going from right to left. Use *n*=0 for the first occurrence. Returns **-1** if the n^{th} occurrence is not found. If *sense* is NO, then the search ignores case. If *overlap* is YES, then the search can match overlapping parts of the receiving string. For example, if you search for ªabc abcº in the string ªabc abc abcº, with *overlap* YES you will get two occurrences and with *overlap* NO you will get one occurrence. This method works by first calling **-numOfString:** to find out how many times the search string occurs, then uses **-spotOfString:** to pluck out the correct part. The primary advantage of this is that it is symmetrical with **-spotOfString:** -- the last part able to be plucked out by that method will be the first part returned with this method. However, this also means that this method is slower, since it must make an extra pass through the string. In the degenerate methods, *n* defaults to 0, *sense* defaults

to YES, and *overlap* defaults to NO.

See also: **-x**

rspotOfType:
rspotOfType:occurrenceNum:

- **rspotOfType:**(int)*type* **occurrenceNum:**(int)*n*

Returns the position number of the *n*th occurrence in *buffer* of any character of type *type* going from right to left. Use *n*=0 for the first occurrence. Returns **-1** if the *n*th occurrence is not found. The parameter *type* may be any of the following constants: MISC_UPPER, MISC_LOWER, MISC_DIGIT, MISC_XDIGIT, MISC_PUNCT, MISC_ASCII, MISC_CNTRL, MISC_PRINT, MISC_SPACE, or MISC_GRAPH. (See **-hasType:** for a description of these constants.) Also, any bitwise OR (using a|o) combination of the constants is permitted. MISC_ALPHA and MISC_ALNUM are predefined combinations that you may use. In the degenerate method, *n* defaults to 0.

See also: **-hasType:**

rstrstr:
rstrstr:caseSensitive:
rstrstr:occurrenceNum:
rstrstr:occurrenceNum:caseSensitive:
rstrstr:overlap:
rstrstr:caseSensitive:overlap:
rstrstr:occurrenceNum:overlap:
rstrstr:occurrenceNum:caseSensitive:overlap:

- (const char *)**rstrstr:**(const char *)*aString* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense*

overlap:(BOOL)*overlap*

Returns a char pointer to the *n*th occurrence in *buffer* of the string *aString* going from right to left. Use *n*=0 for the first occurrence. Returns NULL if the *n*th occurrence is not found. If *sense* is NO, then the search ignores case. If *overlap* is YES, then the search can match overlapping parts of the receiving string. For example, if you search for ªabc abcº in the string ªabc abc abcº, with *overlap* YES you will get two occurrences and with *overlap* NO you will get one occurrence. This method works by first calling **-numOf:** to find out how many times the search string occurs, then uses **-strstr:** to pluck out the correct part. The primary advantage of this is that it is symmetrical with **-strstr:** -- the last part able to be plucked out by that method will be the first part returned with this method. However, this also means that this method is slower, since it must make an extra pass through the string. In the degenerate methods, *n* defaults to 0, *sense* defaults to YES, and *overlap* defaults to NO.

See also: **-x**

rstrString:
rstrString:caseSensitive:
rstrString:occurrenceNum:
rstrString:occurrenceNum:caseSensitive:
rstrString:overlap:
rstrString:caseSensitive:overlap:
rstrString:occurrenceNum:overlap:
rstrString:occurrenceNum:caseSensitive:overlap:

- (const char *)**rstrString:**(id)*sender* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense* **overlap:**(BOOL)*overlap*

Returns a char pointer to the *n*th occurrence in *buffer* of the **±stringValue** of *sender* going from right to left. Use *n*=0 for the first occurrence. Returns NULL if the *n*th occurrence is not found. If *sense* is NO, then the search

ignores case. If *overlap* is YES, then the search can match overlapping parts of the receiving string. For example, if you search for ªabc abcº in the string ªabc abc abcº, with *overlap* YES you will get two occurrences and with *overlap* NO you will get one occurrence. This method works by first calling **-numOfString:** to find out how many times the search string occurs, then uses **-strString:** to pluck out the correct part. The primary advantage of this is that it is symmetrical with **-strString:** -- the last part able to be plucked out by that method will be the first part returned with this method. However, this also means that this method is slower, since it must make an extra pass through the string. In the degenerate methods, *n* defaults to 0, *sense* defaults to YES, and *overlap* defaults to NO.

See also: **-x**

setCapacity:

- **setCapacity:**(unsigned)*newCapacity*

Enlarges the buffer capacity of the receiver to *newCapacity* if necessary. Returns *self*.

See also: **-x**

setDirectory:file:

- **setDirectory:**(const char *)*dir*
  file:(const char *)*file*

Sets the string value of the receiver to a full path name consisting of the path *dir* and file name *file*. Returns *self*.

See also: **-x**

setDoubleValue:

- **setDoubleValue:**(double)*val*

Sets the receiver to be the string which represents the double precision floating point number *val*. Returns *self*.

See also: **-x**

setFloatValue:

- **setFloatValue:**(float)*val*

Sets the receiver to be the string which represents the floating point number *val*. Returns *self*.

See also: **-x**

setFromFormat:

- **setFromFormat:**(const char *)*formatStr, ...*

The same as the ANSI sprintf(), sets the receiver's string buffer using the format specified in *formatStr* and subsequent arguments. Returns *self*.

See also: **-x**

setIntValue:

- **setIntValue:**(int)*val*

Sets the receiver to be the string which represents the integer *val*. Returns *self*.

See also: **-x**

setStringOrderTable:

- **setStringOrderTable:**(NXStringOrderTable *)*table*

Sets the NXStringOrderTable used by the ±**compareTo:** methods. Returns **self**. If not programmatically set using this method, an instance of MiscString will use the default system string table.

See also: **-stringOrderTable:** and **-compareTo:...**

setStringValue:
setStringValue:fromZone:
setStringValue:n:
setStringValue:n:fromZone:

- **setStringValue:**(const char *)*aString* **n:**(int)*n* **fromZone:**(NXZone *)*zone*

Sets the receiver's string value to be the first *n* characters of *aString*. If the receiver's capacity needs to be expanded to fit the new string value, then *zone* is used to allocate the needed memory. Returns **self**. In the degenerate methods, *n* defaults to the length of *aString* and *zone* defaults to the receiver's zone.

See also: **-x**

sprintf:

- **sprintf:**(const char *)*formatStr, ...*

Uses *format* and subsequent arguments to create a new string, as specified by *format* following the ANSI spec

for formats. Replaces any previous string value. Returns *self*.

See also: **-x**

spotOf:
spotOf:caseSensitive:
spotOf:occurrenceNum:
spotOf:occurrenceNum:caseSensitive:
- (int)**spotOf:**(char)*aChar* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense*

Returns the position number of the n^{th} occurrence of *aChar* in *buffer* going from left to right. If *sense* is NO, then the search ignores case. Returns **-1** if the n^{th} occurrence is not found. Occurrences start numbering at zero, not one. In the degenerate methods, *sense* defaults to YES and *n* defaults to 0 (the first occurrence).

See also: **-rspotOf:...**

spotOfChars:
spotOfChars:caseSensitive:
spotOfChars:occurrenceNum:
spotOfChars:occurrenceNum:caseSensitive:
- (int)**spotOfChars:**(const char *)*aString* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense*

Returns the position number of the n^{th} occurrence in *buffer* of any of the characters in *aString* going from left to right. Use *n*=0 for the first occurrence. If *sense* is NO, then the search ignores case. Returns **-1** if the n^{th} occurrence is not found. In the degenerate methods, *sense* defaults to YES and *n* defaults to 0 (the first occurrence).

See also: -**spotOfChars:...** and -**rspotOfChars:...**

spotOfRegex:
spotOfRegex:caseSensitive:
spotOfRegex:occurrenceNum:
spotOfRegex:occurrenceNum:caseSensitive:
spotOfRegex:length:
spotOfRegex:caseSensitive:length:
spotOfRegex:occurrenceNum:length:
spotOfRegex:occurrenceNum:caseSensitive:length:

- (int)**spotOfRegex:**(const char *)*pattern* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense* **length:**(int *)*matchlen*

Returns the position number of the n^{th} occurrence of the regular expression *pattern* in *buffer* going from left to right. Use *n*=0 for the first occurrence. If *sense* is NO, then the search ignores case. Returns **-1** if the n^{th} occurrence is not found, and **-2** on an illegal regular expression. If not NULL, the integer pointed to by *matchlen* gets set to the length of the matched portion. In the degenerate methods, *sense* defaults to YES, *n* defaults to 0, and *matchlen* defaults to NULL.

See also: -**grep:...** and -**matchesRegex:...**

spotOfStr:
spotOfStr:caseSensitive:
spotOfStr:occurrenceNum:
spotOfStr:occurrenceNum:caseSensitive:
spotOfStr:overlap overlap:

spotOfStr:caseSensitive:overlap:
spotOfStr:occurrenceNum:overlap:
spotOfStr:occurrenceNum:caseSensitive:overlap:

- **spotOfStr:**(const char *)*aString* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense* **overlap:**(BOOL)*overlap*

Returns the position number of the *n*th occurrence in *buffer* of the string *aString* going from left to right. Use *n*=0 for the first occurrence. Returns **-1** if the *n*th occurrence is not found. If *sense* is NO, then the search ignores case. If *overlap* is YES, then the search can match overlapping parts of the receiving string. For example, if you search for ªabc abcº in the string ªabc abc abcº, with *overlap* YES you will get two occurrences and with *overlap* NO you will get one occurrence. In the degenerate methods, *n* defaults to 0, *sense* defaults to YES, and *overlap* defaults to NO.

See also: **-x**

spotOfString:
spotOfString:caseSensitive:
spotOfString:occurrenceNum:
spotOfString:occurrenceNum:caseSensitive:
spotOfString:overlap:
spotOfString:caseSensitive:overlap:
spotOfString:occurrenceNum:overlap:
spotOfString:occurrenceNum:caseSensitive:overlap:

- **spotOfString:**(id)*sender* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense* **overlap:**(BOOL)*overlap*

Returns the position number of the *n*th occurrence in *buffer* of the **±stringValue** of *sender* going from left to right.

Use *n*=0 for the first occurrence. Returns **-1** if the *n*th occurrence is not found. If *sense* is NO, then the search ignores case. If *overlap* is YES, then the search can match overlapping parts of the receiving string. For example, if you search for ªabc abcº in the string ªabc abc abcº, with *overlap* YES you will get two occurrences and with *overlap* NO you will get one occurrence. In the degenerate methods, *n* defaults to 0, *sense* defaults to YES, and *overlap* defaults to NO.

See also: **-x**

spotOfType:
spotOfType:occurrenceNum:

- **spotOfType:**(int)*type* **occurrenceNum:**(int)*n*

Returns the position number of the *n*th occurrence in *buffer* of any character of type *type* going from left to right. Use *n*=0 for the first occurrence. Returns **-1** if the *n*th occurrence is not found. The parameter *type* may be any of the following constants: MISC_UPPER, MISC_LOWER, MISC_DIGIT, MISC_XDIGIT, MISC_PUNCT, MISC_ASCII, MISC_CNTRL, MISC_PRINT, MISC_SPACE, or MISC_GRAPH. (See **-hasType:** for a description of these constants.) Also, any bitwise OR (using ª|º) combination of the constants is permitted. MISC_ALPHA and MISC_ALNUM are predefined combinations that you may use. In the degenerate method, *n* defaults to 0.

See also: **-x**

squashSpaces

- **squashSpaces**

This method will remove any redundant spaces in *buffer*. It first calls **-trimSpaces**, and then goes through *buffer* and leaves at most one space between words, except following a period or colon, in which case two

spaces will be left. Note that this method only checks for spaces, so a sequence such as space-tab-space will be left as is. Returns **self**.

See also: -**trimSpaces,** -**trimWhiteSpaces,** ±**trimLeadSpaces,** ±**trimLeadWhiteSpaces,** ±**trimTailSpaces,** and ±**trimTailWhiteSpaces**

streamGets:
streamGets:keepNewline:
- (int)**streamGets:**(NXStream *)*stream* **keepNewline:**(BOOL)*keepit*

Reads in one character at a time from the NXStream *stream* until a newline, EOF, or NX_EOS character is reached, and sets receiver's ±**stringValue** to the resulting line of text. If *keepit* is YES, the newline character is left on at the end (though not the EOF or NX_EOS character). In the degenerate method, *keepit* defaults to YES. Returns EOF if that character or NX_EOS is encountered, **0** otherwise. NOTE: In reading in the line, the receiver's allocated space is doubled each time it fills up. This can mean that the resulting string takes up substantially more space than is needed. If this is a problem, you can use ±**fixStringLength** to remove any excess allocated memory.

See also: -**gets** and -**fgets:...**

stringOrderTable
- (NXStringOrderTable *)**stringOrderTable**

Returns the NXStringOrderTable used to make comparisons between strings.

See also: -**compareTo:n:caseSensitive:** and ±**setStringOrderTable:**

stringValue

- (const char *)**stringValue**

Returns *buffer*, a pointer to the string value.

See also: -**stringValueAndFree**

stringValueAndFree

- (const char *)**stringValueAndFree**

Returns the string value of the receiver, just like ±**stringValue**, and then frees the receiver. The returned buffer will remain valid only until the next time this method is called.

See also: -**stringValue**

strstr:
strstr:caseSensitive:
strstr:occurrenceNum:
strstr:occurrenceNum:caseSensitive:
strstr:overlap:
strstr:caseSensitive:overlap:
strstr:occurrenceNum:overlap:
strstr:occurrenceNum:caseSensitive:overlap:

- (const char *)**strstr:**(const char *)*aString* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense* **overlap:**(BOOL)*overlap*

Returns a char pointer to the *n*th occurrence in *buffer* of the string *aString* going from left to right. Use *n*=0 for the first occurrence. Returns NULL if the *n*th occurrence is not found. If *sense* is NO, then the search ignores case. If *overlap* is YES, then the search can match overlapping parts of the receiving string. For example, if you search for ªabc abcº in the string ªabc abc abcº, with *overlap* YES you will get two occurrences and with *overlap* NO you will get one occurrence. In the degenerate methods, *n* defaults to 0, *sense* defaults to YES, and *overlap* defaults to NO.

See also: **-x**

strString:
strString:caseSensitive:
strString:occurrenceNum:
strString:occurrenceNum:caseSensitive:
strString:overlap:
strString:caseSensitive:overlap:
strString:occurrenceNum:overlap:
strString:occurrenceNum:caseSensitive:overlap:

- (const char *)**strString:**(id)*sender* **occurrenceNum:**(int)*n* **caseSensitive:**(BOOL)*sense* **overlap:**(BOOL)*overlap*

Returns a char pointer to the *n*th occurrence in *buffer* of the **±stringValue** of *sender* going from left to right. Use *n*=0 for the first occurrence. Returns NULL if the *n*th occurrence is not found. If *sense* is NO, then the search ignores case. If *overlap* is YES, then the search can match overlapping parts of the receiving string. For example, if you search for ªabc abc°a in the string ªabc abc abcº, with *overlap* YES you will get two occurrences and with *overlap* NO you will get one occurrence. In the degenerate methods, *n* defaults to 0, *sense* defaults to YES, and *overlap* defaults to NO.

See also: **-x**

subStringLeft:
- **subStringLeft:***subString*

Returns a new MiscString object which is created from the receiving MiscString object's string from it's start up to the start of *subString*. Note that *subString* could be any object with a ±stringValue. (For example, if a MiscString object with the value ªThis is a string.º is sent the message `subStringLeft:key`, where `key` is an object for which a ±stringValue message returns ªaº, then it would return a new MiscString object with the value ªThis is º) Returns **nil** if *subString* doesn't respond to ±stringValue.

See also: **-strstr:** and ±**subStringRight:**

subStringRight:
- **subStringRight:***subString*

Returns a new MiscString object which is created from the receiving MiscString object, starting with *subString* and continuing up to the end of the receiving string. Note that *subString* could be any object with a ±stringValue. (For example, if a MiscString object with the value ªThis is a string.º is sent the message `subStringRight:key`, where `key` is an object for which a ±stringValue message returns ªaº, then it would return a new MiscString object with the value ªa string.º) Returns **nil** if *subString* doesn't respond to ±stringValue.

See also: **-strstr:** and ±**subStringLeft:**

takeDoubleValueFrom:

- **takeDoubleValueFrom:***sender*

Sets the string to the double precision floating point value of *sender*. By convention, *sender* must respond to the ±**doubleValue** message. Returns *self*.

See also: **-x**

takeFloatValueFrom:

- **takeFloatValueFrom:***sender*

Sets the string to the floating point value of *sender*. By convention, *sender* must respond to the ±**floatValue** message. Returns *self*.

See also: **-x**

takeIntValueFrom:

- **takeIntValueFrom:***sender*

Sets the string to the integer value of *sender*. By convention, *sender* must respond to the ±**intValue** message. Returns *self*.

See also: **-x**

takeStringValueFrom:
takeStringValueFrom:fromZone:

- **takeStringValue:**(id)*sender* **fromZone:**(NXZone *)*zone*

Copies the string value of *sender* into *buffer*. If *buffer* is not large enough, the old buffer is freed and a new buffer is allocated from *zone*. Returns **self**. In the degenerate method, *zone* defaults to the receiver's zone.

See also: **-setStringValue:...**

tokenize:into:

- **tokenize:**(const char *)*breakChars*
  into:*aList*

Uses strtok() to break the receiving string up into a series of MiscStrings, one string for each field as delimited by *breakChars*. The new MiscStrings are placed into *aList* in the order that they are created (from start of the receiving string to end) and *aList* is returned. If *aList* is NULL, then a new List object is created and is returned. It is your responsibility to free the tokenized MiscStrings returned in the List.

See also: **-extractPart:...**

toLower

- **toLower**

Converts every uppercase character in *buffer* to lowercase. Returns **self**.

See also: **-toUpper**

toUpper

- **toUpper**

Converts every lowercase character in *buffer* to uppercase. Returns **self**.

See also: **-toLower**

trimLeadSpaces
- trimLeadSpaces

Removes any leading spaces from *buffer*. Returns **self**. Note that this method will only remove spaces (not tabs, linefeeds, etc).

See also: **-trimSpaces, -trimWhiteSpaces, ±trimLeadWhiteSpaces, ±trimTailSpaces, ±trimTailWhiteSpaces,** and **±squashSpaces**

trimLeadWhiteSpaces
- trimLeadWhiteSpaces

Removes any leading white spaces (space, tab, carriage return) from *buffer*. Returns **self**.

See also: **-trimSpaces, -trimWhiteSpaces, ±trimLeadSpaces, ±trimTailSpaces, ±trimTailWhiteSpaces,** and **±squashSpaces**

trimSpaces
- trimSpaces

Removes all leading and trailing spaces from *buffer*. Returns **self**. Note that this method will only remove spaces (not tabs, linefeeds, etc).

See also: **-trimWhiteSpaces, ±trimLeadSpaces, ±trimLeadWhiteSpaces, ±trimTailSpaces, ±trimTailWhiteSpaces,** and **±squashSpaces**

trimTailSpaces
- trimTailSpaces

Removes any trailing spaces from *buffer*. Returns **self**. Note that this method will only remove spaces (not tabs, linefeeds, etc).

See also: **-trimSpaces, -trimWhiteSpaces, ±trimLeadSpaces, ±trimLeadWhiteSpaces, ±trimTailWhiteSpaces,** and **±squashSpaces**

trimTailWhiteSpaces
- trimTailWhiteSpaces

Removes any trailing whitespaces (space, tab, carriage return) from *buffer*. Returns **self**.

See also: **-trimSpaces, -trimWhiteSpaces, ±trimLeadSpaces, ±trimLeadWhiteSpaces, ±trimTailSpaces,** and **±squashSpaces**

trimWhiteSpaces
- trimWhiteSpaces

Removes all leading and trailing whitespaces (space, tab, carriage return) from *buffer*. Returns **self**.

See also: **-trimSpaces, ±trimLeadSpaces, ±trimLeadWhiteSpaces, ±trimTailSpaces,**

±**trimTailWhiteSpaces,** and ±**squashSpaces**

uniqueStringValue

- (NXAtom)**uniqueStringValue**

Returns a pointer to a unique string which is identical to the receiver's string buffer, as determined by the NXUniqueString() function.

See also: **-x**

wordNum:
wordNum:fromZone:

- **wordNum:**(int)*num* **fromZone:**(NXZone *)*zone*

Returns a new MiscString object which contains the *num*th word of *buffer*. Words are separated by any number of spaces, carriage returns, newlines, vertical tabs, or formfeeds (not punctuation characters). The first word is word number ᵃzeroº, following typical C conventions. Returns **nil** if the *num*th word does not exist in *buffer*. In the degenerate method, *zone* defaults to the receiver's zone.

See also: **-wordNum:**

write:

- **write:**(NXTypedStream *)*stream*

Writes the MiscString to the typed stream *stream*. Returns **self**.

See also: **- read:**

Left to do in the documentation:

- Update introduction section to talk about more of the available methods.
- Re-do Method-types section so that it is more useful and is more complete.
- Finish writing and check all ªsee alsoº references to be sure that they are complete.
- Proofread all sections.
- A spelling check.