String

Inherits From: Object

Declared In: String.h

Class Description

A String object contains a simple text string and provides methods for it's manipulation, encompassing all the functions available in <strings.h>. The String 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.

A String is created through the normal process of **±alloc** and **±init**. It may be set to a specific string by means of the **±setString:** and **±setStringValue:** methods. To copy an existing String, use the **±copy** and **±copyFromZone:** methods. To copy a portion of a string, use the **±left:**, **±midFrom:**, and **±right:** methods. Use the **±concatenate:** and **±cat:** methods to concatenate another string onto the end of the string in the buffer. If the current buffer is too small, it is enlarged. Use **±free** to free a String and it's buffer or **±freeString** to free just the buffer.

The **±length** method returns the length of the string currently in the buffer and **±stringValue** returns a pointer to the string itself. The ±**index:** method returns a pointer to the first occurence in the buffer of a specific character and ±**rindex** returns a pointer to the last occurence.

You may compare strings to each other by means of the various **±isEqual:**, **±cmp:**, **± casecmp:**, and **±compareTo:** methods. The **±isEqual:** 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.

A String my be archived by means of the **±read:** and **±write:** methods.

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 String	± init ± allocateBuffer: ± allocateBuffer:fromZone: ± free ± freeString
Copying a String	± copyFromZone: ± left:

	± left:fromZone: ± right: ± right:fromZone: ± midFrom:to: ± midFrom:to:fromZone: ± midFrom:length: ± midFrom:length:fromZone:
Manipulating a String	± cat: ± cat:n: ± cat:fromZone: ± cat:n:fromZone: ± concatenate: ± concatenate:n: ± concatenate:fromZone: ± concatenate:n:fromZone: ± setString: ± setString:fromZone: ± setStringValue: ± setStringValue:fromZone:

Querying attributes	± cmp:
	± cmp:n:
	± casecmp:
	± casecmp:n:
	± compareTo:
	± compareTo:n:
	± compareTo:caseSensitive:
	± compareTo:n:caseSensitive:
	± index:
	± isEqual:
	  ± length
	  ± rindex:

± setStringOrderTable:
± stringOrderTable
± stringValue

Archiving ± read:
± write:

Instance Methods

± allocateBuffer:
± allocateBuffer:fromZone:

allocateBuffer:
- **allocateBuffer:**(int)*size*

If the current buffer is less than *size* bytes, then it is freed and a new buffer is allocated from the receiver's zone to be *size* bytes in length. Returns *self*.

See also: **-allocateBuffer:fromZone:**

allocateBuffer:fromZone:
- **allocateBuffer:**(int)*count* **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 String objects which will dynamically change in size often. By allocating a buffer which is as least as large as you expect the String to grow to during it's lifetime, your application may run faster. This is because the String object won't have to dynamically grow as often, an operation which can slow things down.

See also: **-allocateBuffer:**, **-copyFromZone:**, and **-setString:fromZone:**

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 String 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:**, - **compareTo:**, - **compareTo:caseSensitive:**, - **compareTo:n:**, and - **compareTo:n:caseSensitive:**

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 String 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:**, -**casecmp:n:**, -**cmp:n:**, -**compareTo:**, -**compareTo:caseSensitive:**, -**compareTo:n:** and -**compareTo:n:caseSensitive:**

cat:

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

Calls **strcat()** to concatenate *buffer* and *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 from the String's zone. Returns *self*. This method is provided for those cases in which String 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.

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

cat:fromZone:

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

Calls **strcat()** to concatenate *buffer* and *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 from *zone*. Returns *self*. This method is provided for those cases in which String 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.

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

cat:n:

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

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 from the String's zone. Returns *self*. This method is provided for those cases in which String 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.

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

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 *self*. This method is provided for those cases in which String 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.

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

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 String 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:**, -**casecmp:n:**, -**cmp:n:**, -**compareTo:**, -**compareTo:caseSensitive:**, -**compareTo:n:** and -**compareTo:n:caseSensitive:**

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 String 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:**, - **casecmp:n:**, - **cmp:**, - **compareTo:**, - **compareTo:caseSensitive:**, - **compareTo:n:**, and - **compareTo:n:caseSensitive:**

compareTo:

- (int)**compareTo:**(id)*sender*

Identical to calling the **±compareTo:caseSensitive:** method with a YES as the value of *sense*.

See also: - **compareTo:caseSensitive:**, - **compareTo:n:** and - **compareTo:n:caseSensitive:**

compareTo:caseSensitive:

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

Identical to the **±compareTo:n:caseSensitive:** method, but the entire length of the shortest string is used to make the comparison. This is like calling **±compareTo:n:caseSensitive:** with *n* set to -1.

See also: - **compareTo:**, - **compareTo:n:** and - **compareTo:n:caseSensitive:**

compareTo:n:

- (int)**compareTo:**(id)*sender* **n:**(int)*n*

Identical to calling the **±compareTo:n:caseSensitive:** method with a YES as the value of *sense*.

See also: - **compareTo:**, - **compareTo:caseSensitive:** and - **compareTo:n:caseSensitive:**

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, it is as if the method were called with *n* set to the length of the shorter of the two strings. 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().

See also: - **compareTo:**, - **compareTo:caseSensitive:**, and - **compareTo:n:**

concatenate:
- **concatenate:**(id)*sender*

Adds the String *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 the String's zone. Returns *self*.

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

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

Adds the String *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*.

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

concatenate:n:
- **concatenate:**(id)*sender* **n:**(int)*n*

Adds up to the first *n* bytes of the String *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 the String's zone. Returns *self*.

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

concatenate:n:fromZone:

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

Adds up to the first *n* bytes of the String *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 *zone*. Returns *self*.

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

copyFromZone:

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

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

free

- **free**

Deallocates the String and the contents of *buffer*.

freeString

- **freeString**

Frees the contents of *buffer* and sets the length of the String to zero.

index:

- (const char *)**index:**(char)*aChar*

Returns a pointer to the first occurence of *aChar* in the buffer.

See also: **-rindex:**

init
- **init**

Initializes a new String. Returns **self**.

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

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

left:
- **left:**(int)*count*

Returns a new String object which is composed of the first *count* characters of *buffer*. The new object is allocated from the receiver's zone.

See also: **-left:fromZone:**, **-midFrom:length:**, **-midFrom:length:fromZone:**, **-midFrom:to:**, **-midFrom:to:fromZone:**, **-right:** and **-right:fromZone:**

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

Returns a new String object which is composed of the first *count* characters of *buffer*. The new object is allocated from *zone*.

See also: **-left:**, **-midFrom:length:**, **-midFrom:length:fromZone:**, **-midFrom:to:**, **-midFrom:to:fromZone:**, **-right:**, and **-right:fromZone:**

length
- (int)**length**

Returns the length of the string in *buffer*.

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

Returns a new String object which is composed of *len* characters of *buffer* starting with the *start*th character. The new object is allocated from the receiver's zone.

See also: **-left:**, **-left:fromZone:**, **-midFrom:length:fromZone:**, **-midFrom:to:**, **-midFrom:to:fromZone:**, **-right:**, and **-right:fromZone:**

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

Returns a new String object which is composed of *len* characters of *buffer* starting with the *start*th character. The new object is allocated from *zone*.

See also: **-left:**, **-left:fromZone:**, **-midFrom:length:**, **-midFrom:to:**, **-midFrom:to:fromZone:**, **-right:**, and **-right:fromZone:**

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

Returns a new String 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 the receiver's zone.

See also: **-left:**, **-left:fromZone:**, **-midFrom:length:**, **-midFrom:length:fromZone:**, **-midFrom:to:fromZone:**, **-right:**, and **-right:fromZone:**

midFrom:to:fromZone:

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

Returns a new String 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*.

See also: **-left:**, **-left:fromZone:**, **-midFrom:length:**, **-midFrom:length:fromZone:**, **-midFrom:to:**, **-right:**, and **-right:fromZone:**

read:

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

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

See also: **- write:**

right:

- **right:**(int)*count*

Returns a new String object which is composed of the last *count* characters of *buffer*. The new object is allocated from the receiver's zone.

See also: **-left:**, **-left:fromZone:**, **-midFrom:length:**, **-midFrom:length:fromZone:**, **-midFrom:to:**, **-midFrom:to:fromZone:**, and **-right:fromZone:**

right:fromZone:

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

Returns a new String object which is composed of the last *count* characters of *buffer*. The new object is allocated from *zone*.

See also: **-left:**, **-left:fromZone:**, **-midFrom:length:**, **-midFrom:length:fromZone:**, **-midFrom:to:**, **-midFrom:to:fromZone:**, and **-right:**

rindex:

- (const char *)r**index:**(char)*aChar*

Returns a pointer to the last occurence of *aChar* in the buffer.

See also: **-index:**

setString:

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

Copies *aString* into *buffer*. If *buffer* is not large enough, the old buffer is freed and a new buffer is allocated from the receiver's zone. Returns *self*.

See also: **-setString:fromZone:**, **-setStringValue:**, and **±setStringValue:fromZone:**

setString:fromZone:

- **setString:**(const char *)*aString* **fromZone:**(NXZone *)*zone*

Copies *aString* into *buffer*. If *buffer* is not large enough, the old buffer is freed and a new buffer is allocated from *zone*. Returns *self*.

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

setStringOrderTable:

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

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

See also: -**stringOrderTable:** and -**compareTo:n:caseSensitive:**

setStringValue:

- **setStringValue:**(id)*sender*

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 the receiver's zone. Returns *self*.

See also: -**setString:**, ±**setString:fromZone:**, and -**setStringValue:fromZone:**

setStringValue:fromZone:

- **setStringValue:**(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*.

See also: -**setString:**, ±**setString:fromZone:**, and -**setStringValue:**

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.

write:

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

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

See also: - **read:**