DAYString

Inherits From: Object

Conforms To: NXTransport

Declared In: DAYString.h

Class Description

A DAYString object contains a simple text string and provides methods for it's manipulation, encompassing all the functions available in <strings.h>. The DAYString 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 DAYFastString 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 DAYString class to a certain degree. Depending on your needs, you may find the RCString class to better suit your needs. 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 DAYString 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 DAYString, 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 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 DAYString encrypted by the crypt(3) function. Use **±free** to free a DAYString and it's buffer or **±freeString** to free just the buffer.

Inserting characters or strings into a DAYString may be performed via any of the ±**insert:at:**, ±**insertChar:at:**, ±**insertString:at:**, and ±**addChar:** methods. Deleting a portion of the DAYString is performed by the ±**trimLeadSpaces**, ±**trimTailSpaces**, ±**trimSpaces**, and ±**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 ±**index:...** methods return a pointer to the n^{th} occurence in the buffer of a specific character and ±**rindex:...** return a pointer to the n^{th} occurence from the end (right to left). The ±**spotOf:...** and ±**rspotOf:** methods work similarly, but return an integer which gives the character number of the occurence.

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 DAYString, and ±**wordNum:** to create a new DAYString 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 simgle delimiter.

Use the ±**toUpper** and ±**toLower** methods to change all characters in the DAYString 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 DAYString's buffer. The ±**charAt:** method returns a single character from a given location in the DAYString's buffer.

A DAYString my 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.)

Two conventions with char * buffers are followed by the DAYString 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 DAYString object, so you ought to copy it yourself if you intend to keep it around for any length of time. When a DAYString 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 DAYString 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 DAYString in your .nib files with it pre-initialized to an arbitrary string value.

Disclaimer and other notes: If you have any problems with the DAYString 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. You may find it worth noting some of the comments in the source file DAYString.m; several trade-offs have been made, typically in favor of making the code more maintainable and robust, but at the expense of speed.

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 DAYString ± init
± initString:
± allocateBuffer:
± allocateBuffer:fromZone:
± free
± freeString
+ newWithString:

Copying a DAYString	± 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:
Manipulating a DAYString	± addChar: ± cat: ± cat:n: ± cat:fromZone: ± cat:n:fromZone: ± concatenate: ± concatenate:n:

	± concatenate:fromZone: ± concatenate:n:fromZone: ± insert:at: ± insertChar:at: ± insertString:at: ± removeFrom:length: ± removeFrom:to: ± replace:with: ± replace:withString: ± replaceCharAt:with: ± replaceFrom:length:with: ± replaceFrom:length:withChar: ± replaceFrom:length:withString: ± replaceFrom:to:with: ± replaceFrom:to:withChar: ± replaceFrom:to:withString: ± setStringValue: ± setStringValue:fromZone: ± squashSpaces ± takeStringValue: ± takeStringValue:fromZone: ± toLower ± toUpper ± trimLeadSpaces ± trimSpaces ± trimTailSpaces
Querying attributes	± charAt: ± cmp: ± cmp:n:

± casecmp:
± casecmp:n:
± compareTo:
± compareTo:n:
± compareTo:caseSensitive:
± compareTo:n:caseSensitive:
± endcasecmp:
± endcasecmp:n:
± endcmp:
± endcmp:n:
± endCompareTo:
± endCompareTo:caseSensitive:
± endCompareTo:n:
± endCompareTo:n:caseSensitive:
± index:
± index:caseSensitive:
± index:occurenceNum:
± index:occurenceNum:caseSensitive:
± isEqual:
± length
± numWords
± rindex:
± rindex:caseSensitive:
± rindex:occurenceNum:
± rindex:occurenceNum:caseSensitive:
± rspotOf:
± rspotOf:caseSensitive:
± rspotOf:occurenceNum:
± rspotOf:occurenceNum:caseSensitive:
± setStringOrderTable:

	± spotOf: ± spotOf:caseSensitive: ± spotOf:occurenceNum: ± spotOf:occurenceNum:caseSensitive: ± stringOrderTable ± stringValue ± strstr:
Archiving	± read: ± write:

Instance Methods

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

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

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

See also: -**allocateBuffer:**, -**copyFromZone:**, and -**setStringValue: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 DAYString 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 DAYString 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 DAYString's zone. Returns **self**. This method is provided for those cases in which DAYString 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 DAYString 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 DAYString's zone. Returns **self**. This method is provided for those cases in which DAYString 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 **nil**. This method is provided for those cases in which DAYString 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:**

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 DAYString 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 DAYString 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 DAYString *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 DAYString'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 DAYString *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 DAYString *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 DAYString'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 DAYString *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 DAYString. Memory for the new DAYString is allocated from *zone*. The string stored in *buffer* is copied.

endcasecmp:

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

Performs a case insensitive comparison of *aString* with the end of *buffer*. Return values follow those of the -**compareTo:** methods. This is like calling -**endcasecmp:n:** with *n* set to -1.

See also: -**endcasecmp:n**, -**endcmp:**, -**endcmp:n:**, -**endCompareTo:**, -**endCompareTo:caseSensitive:**, -**endCompareTo:n:**, and -**endCompareTo:n:caseSensitive:**

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.

See also: **-endcasecmp:**, **-endcmp:**, **-endcmp:n:**, **-endCompareTo:**, **-endCompareTo:caseSensitive:**, **-endCompareTo:n:**, and **-endCompareTo:n:caseSensitive:**

endcmp:

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

Performs a case sensitive comparison of *aString* with the end of *buffer*. Return values follow those of the **-compareTo:** methods. This is like calling **-endcmp:n:** with *n* set to -1.

See also: **-endcmp:n**, **-endcasecmp:**, **-endcasecmp:n:**, **-endCompareTo:**, **-endCompareTo:caseSensitive:**, **-endCompareTo:n:**, and **-endCompareTo:n:caseSensitive:**

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.

See also: **-endcmp:**, **-endcasecmp:**, **-endcasecmp:n:**, **-endCompareTo:**, **-endCompareTo:caseSensitive:**, **-endCompareTo:n:**, and **-endCompareTo:n:caseSensitive:**

endCompareTo:

- **endCompareTo:**(id)*sender*

Performs a case sensitive comparison of the **-stringValue** of *sender* with the end of *buffer*. Return values follow those of the **-compareTo:** methods. This is like calling the **-endCompareTo:n:caseSensitive:** with *n* set to -1 and *sense* set to YES.

See also: **-endCompareTo:caseSensitive:**, **-endCompareTo:n:**, **-endCompareTo:n:caseSensitive:**, **-endcmp:**, **-endcmp:n:**, **-endcasecmp:**, and **-endcasecmp:n:**

endCompareTo:caseSensitive:

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

Compares the last *n* characters of *sender*'s **-stringValue** with the last *n* characters of *buffer*. If *sense* is NO, the comparison is case insensitive. Return values follow those of the **-compareTo:** methods. This is like calling the **-endCompareTo:n: caseSensitive:** method with *n* set to -1.

See also: **-endCompareTo:**, **-endCompareTo:n:**, **-endCompareTo:n:caseSensitive:**, **-endcmp:**, **-endcmp:n:**, **-endcasecmp:**, and **-endcasecmp:n:**

endCompareTo:n:

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

Performs a case sensitive comparison of 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. Return values follow those of the **-compareTo:** methods. This is like calling the **-endCompareTo:n:caseSensitive:** method with *sense* set to YES.

See also: -**endCompareTo:**, -**endCompareTo:caseSensitive:**, -**endCompareTo:n:caseSensitive:**, -**endcmp:**, -**endcmp:n:**, -**endcasecmp:**, and -**endcasecmp: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.

See also: -**endCompareTo:**, -**endCompareTo:caseSensitive:**, -**endCompareTo:n:**, -**endcmp:**, -**endcmp:n:**, -**endcasecmp:**, and -**endcasecmp:n:**

extractPart:useAsDelimiter:
- **extractPart:**(int)*n* **useAsDelimiter:**(char)*c*

Same as ±**extractPart:useAsDelimiter:caseSensitive:fromZone:** using the same zone as the DAYString which received the message and with *sense* set to YES.

See also: -**extractPart:useAsDelimiter:caseSensitive:**, -**extractPart:useAsDelimiter:caseSensitive:fromZone:**, -**extractPart:useAsDelimiter:fromZone:**, -**fileName**, -**fileNameFromZone:**, -**pathName**, -**pathNameFromZone:**, and -**wordNum:**

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

Same as ±**extractPart:useAsDelimiter:caseSensitive:fromZone:** using the same zone as the DAYString which received the message.

See also: **-extractPart:useAsDelimiter:**, **-extractPart:useAsDelimiter:caseSensitive:fromZone:**, **-extractPart:useAsDelimiter:fromZone:**, **-fileName**, **-fileNameFromZone:**, **-pathName**, **-pathNameFromZone:**, and **-wordNum:**

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 DAY_FIRST and DAY_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 DAYString is returned, allocated from *zone*. If the field specified does not exist (i.e. you specified field 7 when there are only 6, etc.) then **nil** is returned.

See also: **-extractPart:useAsDelimiter:**, **-extractPart:useAsDelimiter:caseSensitive:**, **-extractPart:useAsDelimiter:fromZone:**, **-fileName**, **-fileNameFromZone:**, **-pathName**, **-pathNameFromZone:**, and **-wordNum:**

extractPart:useAsDelimiter:fromZone:

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

Same as ±**extractPart:useAsDelimiter:caseSensitive:fromZone:** with *sense* set to YES.

See also: **-extractPart:useAsDelimiter:**, **-extractPart:useAsDelimiter:caseSensitive:**, **-extractPart:useAsDelimiter:caseSensitive:fromZone:**, **-fileName**, **-fileNameFromZone:**, **-pathName**, **-pathNameFromZone:**, and **-wordNum:**

fileName
- **fileName**

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

See also: -**extractPart:useAsDelimiter:**, -**extractPart:useAsDelimiter:caseSensitive:**, ±**extractPart:useAsDelimiter:caseSensitive:fromZone:**, -**extractPart:useAsDelimiter:fromZone:**, -**fileNameFromZone:**, -**pathName**, -**pathNameFromZone:**, and -**wordNum:**

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

Assuming that the receiving DAYString contains a UNIX path name of some sort, this method returns a new DAYString 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:**, -**extractPart:useAsDelimiter:caseSensitive:**, ±**extractPart:useAsDelimiter:caseSensitive:fromZone:**, -**extractPart:useAsDelimiter:fromZone:**, -**fileName**, -**pathName**, -**pathNameFromZone:**, and -**wordNum:**

free
- **free**

Deallocates the DAYString and the contents of *buffer*.

freeString
- **freeString**

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

index:

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

Returns a pointer to the leftmost occurence of *aChar* in *buffer*. The search is case sensitive. Returns NULL if *aChar* is not found.

See also: **-index:caseSensitive:**, **-index:occurenceNum:**, **-index:caseSensitive:occurenceNum:**, **-rindex:**, **-rindex:caseSensitive:**, **-rindex:occurenceNum:**, and **-rindex:caseSensitive:occurenceNum:**

index:caseSensitive:

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

Returns a pointer to the leftmost occurence of *aChar* in *buffer*. If sense is NO, then the search ignores case. Returns NULL if *aChar* is not found.

See also: **-index**, **-index:occurenceNum:**, **-index:caseSensitive:occurenceNum:**, **-rindex:**, **-rindex:caseSensitive:**, **-rindex:occurenceNum:**, and **-rindex:caseSensitive:occurenceNum:**

index:occurenceNum:

- (const char *)**index:**(char)*aChar* **occurenceNum:**(int)*n*

Returns a pointer to the n^{th} occurence of *aChar* in *buffer* going from left to right. The search is case sensitive. Returns NULL if the n^{th} occurence of *aChar* is not found.

See also: **-index**, **-index:caseSensitive:**, **-index:caseSensitive:occurenceNum:**, **-rindex:**,

-rindex:caseSensitive:, **-rindex:occurenceNum:**, and **-rindex:caseSensitive:occurenceNum:**

index:occurenceNum:caseSensitive:

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

Returns a pointer to the n^{th} occurence of *aChar* in *buffer* going from left to right. If sense is NO, then the search ignores case. Returns NULL if the n^{th} occurence of *aChar* is not found.

See also: **-index**, **-index:caseSensitive:**, **-index:occurenceNum:**, **-rindex:**, **-rindex:caseSensitive:**, **-rindex:occurenceNum:**, and **-rindex:caseSensitive:occurenceNum:**

init

- **init**

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

See also: ±**initString:**

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:at:

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

Inserts *aString* into *buffer* at position *index.* Returns **self**.

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

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**.

See also: **-insert:at:** and **insertString:at:**

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

Inserts the **±stringValue** of *sender* into *buffer* at position *index.* Returns **self**.

See also: **-insert:at:** and **insertChar:at:**

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

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

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

Returns a new DAYString 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 DAYString 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 DAYString 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 DAYString 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 DAYString 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 DAYString 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:**

newWithString:

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

Returns a new DAYString object containing *aString*.

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).

pathName

- **pathName**

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

See also: -**extractPart:useAsDelimiter:**, -**extractPart:useAsDelimiter:caseSensitive:**, ±**extractPart:useAsDelimiter:caseSensitive:fromZone:**, -**extractPart:useAsDelimiter:fromZone:**, -**fileName**, -**fileNameFromZone:**, -**pathNameFromZone:**, and -**wordNum:**

pathNameFromZone:

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

Assuming that the receiving DAYString contains a UNIX path name of some sort, this method returns a new DAYString 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:**, **-extractPart:useAsDelimiter:caseSensitive:**, **±extractPart:useAsDelimiter:caseSensitive:fromZone:**, **-extractPart:useAsDelimiter:fromZone:**, **-fileName**, **-fileNameFromZone:**, **-pathName**, and **-wordNum:**

read:

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

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

See also: **- write:**

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:**(const char *)*subString* **with:**(const char *)*newString*

Searches *buffer* for *subString* and, if found, replaces it with *newString*. Returns **self**.

See also: - **replace:withString:**

replace:withString:
- **replace:**(const char *)*subString* **withString:**(id)*sender*

Searches *buffer* for *subString* and, if found, replaces it with the **±stringValue** of *sender*. Returns **self**.

See also: - **replace:with:**

replaceCharAt:with:
- **replaceCharAt:**(int)*index* **with:**(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**.

replaceFrom:length:with:
- **replaceFrom:**(int)*start* **length:**(int)*len* **with:**(const char *)*aString*

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

See also: - **replaceFrom:length:withString:**, - **replaceFrom:to:with:**, and - **replaceFrom:to:withString:**

replaceFrom:length:withChar:
- **replaceFrom:**(int)*start* **length:**(int)*len* **withChar:**(char)*aChar*

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

See also: - **replaceFrom:length:withString:**, - **replaceFrom:to:with:**, and - **replaceFrom:to:withString:**

replaceFrom:length:withString:
- **removeFrom:**(int)*start* **length:**(int)*end* **withString:**(id)*sender*

Replaces *len* characters from *buffer,* starting with the *start*th character, with 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:length:with:**, - **replaceFrom:to:with:**, and - **replaceFrom:to:withString:**

replaceFrom:to:with:
- **replaceFrom:**(int)*start* **to:**(int)*end* **with:**(const char *)*aString*

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

See also: - **replaceFrom:length:with:**, - **replaceFrom:length:withString:**, and - **replaceFrom:to:withString:**

replaceFrom:to:withChar:
- **replaceFrom:**(int)*start* **to:**(int)*end* **withChar:**(char)*achar*

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

See also: - **replaceFrom:length:with:**, - **replaceFrom:length:withString:**, and - **replaceFrom:to:withString:**

replaceFrom:to:withString:

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

See also: - **replaceFrom:length:with:**, - **replaceFrom:length:withString:**, and - **replaceFrom:to:with:**

reverse

- **reverse**

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

right:

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

Returns a new DAYString 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 DAYString 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 *)**index:**(char)*aChar*

Returns a pointer to the rightmost occurence of *aChar* in *buffer*. The search is case sensitive. Returns NULL if *aChar* is not found.

See also: -**rindex:caseSensitive:**, -**rindex:occurenceNum:**, -**rindex:caseSensitive:occurenceNum:**, -**index:**, -**index:caseSensitive:**, -**index:occurenceNum:**, and -**index:caseSensitive:occurenceNum:**

rindex:caseSensitive:

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

Returns a pointer to the rightmost occurence of *aChar* in *buffer*. If sense is NO, then the search ignores case. Returns NULL if *aChar* is not found.

See also: -**rindex:**, -**rindex:occurenceNum:**, -**rindex:caseSensitive:occurenceNum:**, -**index:**, -**index:caseSensitive:**, -**index:occurenceNum:**, and -**index:caseSensitive:occurenceNum:**

rindex:occurenceNum:

- (const char *)**index:**(char)*aChar* **occurenceNum:**(int)*n*

Returns a pointer to the n^{th} occurence of *aChar* in *buffer* going from right to left. The search is case sensitive. Returns NULL if the n^{th} occurence of *aChar* is not found.

See also: **-rindex:**, **-rindex:caseSensitive:**, **-rindex:caseSensitive:occurenceNum:**, **-index:**, **-index:caseSensitive:**, **-index:occurenceNum:**, and **-index:caseSensitive:occurenceNum:**

rindex:occurenceNum:caseSensitive:

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

Returns a pointer to the n^{th} occurence of *aChar* in *buffer* going from right to left. If sense is NO, then the search ignores case. Returns NULL if the n^{th} occurence of *aChar* is not found.

See also: **-rindex:**, **-rindex:caseSensitive:**, **-rindex:occurenceNum:**, **-index:**, **-index:caseSensitive:**, **-index:occurenceNum:**, and **-index:caseSensitive:occurenceNum:**

rspotOf:

- (int)**rspotOf:**(char)*aChar*

Returns the position number of the rightmost occurrence of *aChar* in *buffer*. The search is case sensitive. Returns **-1** if *aChar* is not found.

See also: **-rspotOf:caseSensitive:**, **-rspotOf:occurenceNum:**, **-rspotOf:occurenceNum:caseSensitive:**, **-spotOf**, **-spotOf:caseSensitive:**, **-spotOf:occurenceNum:**, and **-spotOf:occurenceNum:caseSensitive:**,

rspotOf:caseSensitive

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

Returns the position number of the rightmost occurrence of *aChar* in *buffer*. If *sense* is NO, then the search ignores case. Returns **-1** if the *n*th occurrence is not found.

See also: **-rspotOf**, **-rspotOf:occurenceNum:**, **-rspotOf:occurenceNum:caseSensitive:**, **-spotOf**, **-spotOf:caseSensitive:**, **-spotOf:occurenceNum:**, and **-spotOf:occurenceNum:caseSensitive:**,

rspotOf:occurenceNum:

- (int)**rspotOf:**(char)*aChar* **occurenceNum:**(int)*n*

Returns the position number of the *n*th occurrence of *aChar* in *buffer* going from right to left. The search is case sensitive. Returns **-1** if the *n*th occurrence of *aChar* is not found.

See also: **-rspotOf**, **-rspotOf:caseSensitive:**, **-rspotOf:occurenceNum:caseSensitive:**, **-spotOf**, **-spotOf:caseSensitive:**, **-spotOf:occurenceNum:**, and **-spotOf:occurenceNum:caseSensitive:**,

rspotOf:occurenceNum:caseSensitive:

- (int)**rspotOf:**(char)*aChar* **occurenceNum:**(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.

See also: **-rspotOf**, **-rspotOf:caseSensitive:**, **-rspotOf:occurenceNum:**, **-spotOf**, **-spotOf:caseSensitive:**, **-spotOf:occurenceNum:**, and **-spotOf:occurenceNum:caseSensitive:**,

setStringOrderTable:

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

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

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

setStringValue:

- **setStringValue:**(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: -**setStringValue:fromZone:**, -**takeStringValue:**, and ±**takeStringValue:fromZone:**

setStringValue:fromZone:

- **setStringValue:**(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: -**setStringValue:**, -**takeStringValue:**, and -**takeStringValue:fromZone:**

spotOf:

- (int)**spotOf:**(char)*aChar*

Returns the position number of the leftmost occurrence of *aChar* in *buffer*. The search is case sensitive. Returns **-1** if *aChar* is not found.

See also: -**spotOf:caseSensitive:**, -**spotOf:occurenceNum:**, -**spotOf:occurenceNum:caseSensitive:**, -**rspotOf**, -**rspotOf:caseSensitive:**, -**rspotOf:occurenceNum:**, and -**rspotOf:occurenceNum:caseSensitive:**,

spotOf:caseSensitive:

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

Returns the position number of the leftmost occurrence of *aChar* in *buffer*. If *sense* is NO, then the search ignores case. Returns **-1** if *aChar* is not found.

See also: -**spotOf**, -**spotOf:occurenceNum:**, -**spotOf:occurenceNum:caseSensitive:**, -**rspotOf**, -**rspotOf:caseSensitive:**, -**rspotOf:occurenceNum:**, and -**rspotOf:occurenceNum:caseSensitive:**,

spotOf:occurenceNum:

- (int)**spotOf:**(char)*aChar* **occurenceNum:**(int)*n*

Returns the position number of the n^{th} occurrence of *aChar* in *buffer* going from left to right. The search is case sensitive. Returns **-1** if the n^{th} occurrence of *aChar* is not found.

See also: -**spotOf**, -**spotOf:caseSensitive:**, -**spotOf:occurenceNum:caseSensitive:**, -**rspotOf**, -**rspotOf:caseSensitive:**, -**rspotOf:occurenceNum:**, and -**rspotOf:occurenceNum:caseSensitive:**,

spotOf:occurenceNum:caseSensitive:

- (int)**spotOf:**(char)*aChar* **occurenceNum:**(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.

See also: -**spotOf**, -**spotOf:caseSensitive:**, -**spotOf:occurenceNum:**, -**rspotOf**, -**rspotOf:caseSensitive:**, -**rspotOf:occurenceNum:**, and -**rspotOf:occurenceNum:caseSensitive:**,

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,** -**trimLeadSpaces,** and ±**trimTailSpaces:**

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.

strstr:

- (const char *)**strstr:**(const char *)*subString*

Returns a pointer to the start of the first occurence of *subString* in the string. Note that this pointer points into the internal buffer of the string object and should therefore not be freed. If you wish to manipulate it, you should create a new DAYString object. For example:

```
newString = [[DAYString alloc] initString:[origString strstr:aSubString]];
```

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

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

Returns a new DAYString object which is created from the receiving DAYString 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 DAYString 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 DAYString object with the value ªThis is º) Returns an empty DAYString if *subString* doesn't respond to ±stringValue.

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

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

Returns a new DAYString object which is created from the receiving DAYString 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 DAYString 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 DAYString object with the value ªa string.º) Returns **nil** if *subString* doesn't respond to ±stringValue.

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

takeStringValue:
- **takeStringValue:**(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: **-setStringValue:**, **-setStringValue:fromZone:**, and **-takeStringValue:fromZone:**

takeStringValue: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**.

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

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, ±trimTailSpaces,** and **±squashSpaces**

trimSpaces
- **trimSpaces**

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

See also: **-trimLeadSpaces, ±trimTailSpaces,** 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, -trimLeadSpaces,** and **±squashSpaces**

wordNum:
- **wordNum:**(int)*num*

Returns a new DAYString object wich 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). Returns **nil** if the *num*th word does not exist in *buffer*.

write:

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

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

See also: - **read:**