stamp.util.text

Class Format

```
java.lang.Object
  |
  +--stamp.util.text.Format
```

public class **Format**
extends Object

This library provides formatted string output and scan methods based on the standard C sprintf and sscanf functions. With this class you can convert byte, int, short and char types into binary, octal or hexadecimal formatted strings. You can specify the minimum and maximum number of columns to display your values, and these values can be left or right justified, with or without padded zeros. Methods are included for the following C library functions: itoa, atoi, printf, sprintf, and sscanf.

Most java implementations of the sprintf and sscanf functions use a linked list of objects to replace the variable list of arguments of the sprintf and sscanf functions while maintaining the original format string (in case of porting from C). This linked list requires many of new objects and these are normally only used once.

This implementation defines the functions bprintf and bscanf that take one, and only one, variable parameter. The original format string must be split into pieces that all have one format specifier. This normally is not a problem since most format strings consists of fixed text with format specifiers for values. As there are no linked lists it requires few memory resources.

Constructor Summary

`Format()`

Method Summary

`static int`	`atoi(char[] s)` Convert signed decimal string to signed integer.
`static int`	`bprintf(char[] str, int si, char[] format)` Print formatted string to character buffer, maintaining buffer index.
`static int`	`bprintf(char[] str, int si, char[] format, char[] data)`

static int	**bprintf**(char[] str, int si, char[] format, int data)
static int	**bprintf**(char[] str, int si, char[] format, String data)
static int	**bprintf**(char[] str, int si, String format)
static int	**bprintf**(char[] str, int si, String format, char[] data)
static int	**bprintf**(char[] str, int si, String format, int data)
static int	**bprintf**(char[] str, int si, String format, String data)
static int	**bscanf**(char[] str, int si, char[] format, char[] data) Scan formatted string into variable, maintaining formatted string index (special version of `sscanf`, to ease porting `scanf` and `sscanf` with multiple arguments) for format specifiers see `sscanf`
static int	**bscanf**(char[] str, int si, char[] format, int[] data)
static int	**bscanf**(char[] str, int si, String format, char[] data)
static int	**bscanf**(char[] str, int si, String format, int[] data)
static int	**bscanf**(String str, int si, char[] format, char[] data)
static int	**bscanf**(String str, int si, char[] format, int[] data)
static int	**bscanf**(String str, int si, String format, char[] data)
static int	**bscanf**(String str, int si, String format, int[] data)
static boolean	**isDigit**(int ch) Test if character is a digit.
static boolean	**isSpace**(int ch) Test if character is a space (0x20), tab (\t) or newline (\n).
static void	**itoa**(int n, char[] s) Convert signed integer to signed decimal string.
static int	**printf**(char[] format) Print formatted string to message window (stdout).
static int	**printf**(char[] format, char[] data)
static int	**printf**(char[] format, int data)

static int	printf(char[] format, String data)
static int	printf(String format)
static int	printf(String format, char[] data)
static int	printf(String format, int data)
static int	printf(String format, String data)
static void	reverse(char[] s) Reverse string in place.
static int	sprintf(char[] str, char[] format) Print formatted string to character buffer.
static int	sprintf(char[] str, char[] format, char[] data)
static int	sprintf(char[] str, char[] format, int data)
static int	sprintf(char[] str, char[] format, String data)
static int	sprintf(char[] str, String format)
static int	sprintf(char[] str, String format, char[] data)
static int	sprintf(char[] str, String format, int data)
static int	sprintf(char[] str, String format, String data)
static int	sscanf(char[] str, char[] format, char[] data) Scan formatted string into variable.
static int	sscanf(char[] str, char[] format, int[] data)
static int	sscanf(char[] str, String format, char[] data)
static int	sscanf(char[] str, String format, int[] data)
static int	sscanf(String str, char[] format, char[] data)
static int	sscanf(String str, char[] format, int[] data)

`static int`	`sscanf(String str, String format, char[] data)`
`static int`	`sscanf(String str, String format, int[] data)`
`static int`	`strLen(char[] s)` Get length of null terminated string in character array.

Methods inherited from class java.lang.Object
`equals`

Constructor Detail

Format

```
public Format()
```

Method Detail

isDigit

```
public static boolean isDigit(int ch)
```

Test if character is a digit.

Parameters:
`ch` - Character to be tested
Returns:
True if ch is a digit

---

isSpace

```
public static boolean isSpace(int ch)
```

Test if character is a space (0x20), tab (\t) or newline (\n).

Parameters:
`ch` - Character to be tested
Returns:
True if ch is a space, tab or newline

---

reverse

```
public static void reverse(char[] s)
```

Reverse string in place.

Parameters:
`s` - Character array to reverse

---

strLen

```
public static int strLen(char[] s)
```

Get length of null terminated string in character array.

Parameters:
`s` - Character array that holds null terminated string
Returns:
Length of null terminated string (not counting the null)

---

itoa

```
public static void itoa(int n,
                        char[] s)
```

Convert signed integer to signed decimal string. (range is -32768 to +32767)

Parameters:
`n` - Integer value to convert
`s` - Character array to hold output

---

atoi

```
public static int atoi(char[] s)
```

Convert signed decimal string to signed integer. (range is -32768 to +32767)

Parameters:
`s` - Character array that holds decimal string
Returns:
Value of signed decimal string

---

printf

```
public static int printf(char[] format)
```

Print formatted string to message window (stdout). (Javelin version of C `printf`, limited to 1 parameter max.)

format specifier for byte, char, short and int: `%c,%d,%i,%b,%o,%u,%x`
format specifier for string: `%s`
field specification (example)

```
%-05.7d where - for left justify, default right justify
              0 for padding with zeroes, default spaces
              5 minimum field width
              . field separator
              7 maximum field width
```

Use %% to print a percentage sign.

Parameters:
`format` - String defining format
Returns:
the number of characters printed to stdout (message window)

---

printf

```
public static int printf(String format)
```

---

printf

```
public static int printf(String format,
                         int data)
```

---

printf

```
public static int printf(char[] format,
                         int data)
```

---

printf

```
public static int printf(String format,
                         String data)
```

---

printf

```
public static int printf(char[] format,
                         String data)
```

---

printf

```
public static int printf(String format,
                         char[] data)
```

---

printf

```
public static int printf(char[] format,
                         char[] data)
```

---

sprintf

```
public static int sprintf(char[] str,
                          char[] format)
```

Print formatted string to character buffer. (Javelin version of C `sprintf`, limited to 1 argument max.) for format specifiers see `printf`
for other parameters see `printf`

Parameters:
`str` - Character buffer holding formatted output

Returns:
Index of str pointing at trailing null

---

sprintf

```
public static int sprintf(char[] str,
                          String format)
```

---

sprintf

```
public static int sprintf(char[] str,
                          String format,
                          int data)
```

---

sprintf

```
public static int sprintf(char[] str,
                          char[] format,
                          int data)
```

---

sprintf

```
public static int sprintf(char[] str,
                          String format,
                          String data)
```

---

sprintf

```
public static int sprintf(char[] str,
                          char[] format,
                          String data)
```

---

sprintf

```
public static int sprintf(char[] str,
                          String format,
                          char[] data)
```

---

sprintf

```
public static int sprintf(char[] str,
                          char[] format,
                          char[] data)
```

---

bprintf

```
public static int bprintf(char[] str,
                          int si,
                          char[] format)
```

Print formatted string to character buffer, maintaining buffer index. (special version of `sprintf`, to ease porting `printf` and `sprintf` with multiple arguments)
for format specifiers see `printf`

Example:
An original `printf` statement like:
`printf("outside temperature %d %s inside temperature %d %s",12,"fahrenheit",24,"celsius");`
would be written for the Javelin as:

```
printf("outside temperature %d ",12);
printf("%s ","fahrenheit");
printf("inside temperature %d ",24);
printf("%s","celsius");
```

The original `printf` is simply split up in smaller `printf` statements that take only 1 argument. The same should apply for `sprintf.` However, `sprintf` outputs characters starting at the first position of the supplied character array. To get a single output string while having multiple arguments, `bprintf` outputs characters starting at a supplied position.

Example:
An original `sprintf` statement like
`sprintf(buffer,"outside temperature %d %s inside temperature %d %s",12,"fahrenheit",24,"celsius");`
would be written for the Javelin as:

```
int k=0;
k=bprintf(buffer,k,"outside temperature %d ",12);
k=bprintf(buffer,k,"%s ","fahrenheit");
k=bprintf(buffer,k,"inside temperature %d ",24);
k=bprintf(buffer,k,"%s","celsius");
```

The original `sprintf` is simply split up in smaller `bprintf` statements that take only 1 argument. The end result is the same: a single output string in buffer. Obviously, if there is only 1 argument in an original `sprintf`, then use `sprintf`.

Parameters:
`str` - Character buffer holding formatted output
`si` - Start index in str for formatted output for other parameters see printf
Returns:
Index of str pointing at closing 0

---

bprintf

```
public static int bprintf(char[] str,
                          int si,
                          String format)
```

---

bprintf

```
public static int bprintf(char[] str,
                          int si,
                             String format,
                             int data)
```

---

bprintf

```
public static int bprintf(char[] str,
                             int si,
                             char[] format,
                             int data)
```

---

bprintf

```
public static int bprintf(char[] str,
                             int si,
                             String format,
                             String data)
```

---

bprintf

```
public static int bprintf(char[] str,
                             int si,
                             char[] format,
                             String data)
```

---

bprintf

```
public static int bprintf(char[] str,
                             int si,
                             String format,
                             char[] data)
```

---

bprintf

```
public static int bprintf(char[] str,
                             int si,
                             char[] format,
                             char[] data)
```

---

sscanf

```
public static int sscanf(char[] str,
                         char[] format,
                         char[] data)
```

Scan formatted string into variable. (Javelin version of C `sscanf`, single argument)

format specifier for byte, char, short and int: `%c,%d,%i,%b,%o,%u,%x`
format specifier for string: `%s`
field specification (example)

```
   %*4d where * specifies to scan but not assign scanned value
              4 (maximum) field width to scan
```

Parameters:
`str` - Character array holding formatted string
`format` - String defining format
`data` - pointer of variable to hold scanned value

Returns:
Index in str pointing at next position to read

---

sscanf

```
public static int sscanf(String str,
                         String format,
                         int[] data)
```

---

sscanf

```
public static int sscanf(String str,
                         char[] format,
                         int[] data)
```

---

sscanf

```
public static int sscanf(char[] str,
                         String format,
                         int[] data)
```

---

sscanf

```
public static int sscanf(char[] str,
                         char[] format,
                         int[] data)
```

---

sscanf

```
public static int sscanf(String str,
                         String format,
                         char[] data)
```

---

sscanf

```
public static int sscanf(String str,
                         char[] format,
                         char[] data)
```

---

sscanf

```
public static int sscanf(char[] str,
                         String format,
                         char[] data)
```

---

bscanf

```
public static int bscanf(char[] str,
                         int si,
                         char[] format,
                         char[] data)
```

Scan formatted string into variable, maintaining formatted string index (special version of `sscanf`, to ease porting `scanf` and `sscanf` with multiple arguments) for format specifiers see `sscanf`

Example:
An original `sscanf` statement like `sscanf(buffer,"outside temperature %d %s inside temperature %d %s",outtemp, outunit,intemp,inunit);`
would be written for the Javelin as:

```
   int k=0;
   k=bscanf(buffer,k,"outside temperature %d ",outtemp);
   k=bscanf(buffer,k,"%s ",outunit);
   k=bscanf(buffer,k,"inside temperature %d ",intemp);
   k=bscanf(buffer,k,"%s",inunit);
```

The original `sscanf` is simply split up in smaller bscanf statements that take only 1 argument. The variable `k` keeps track of the parsing position.
Note:
The one missing is method from the original C library is `scanf`. It would use the Terminal.getChar() method to read parameters from the Javelin message window. The problem with `scanf` is that wrong user input may cause `scanf` to never return. Better to read in a complete

string (terminated by user input carriage return) and then parse the string using `bscanf`.

Parameters:
`str` - Character array holding formatted string
`si` - Start index in `str` to read from for other parameters see sscanf
Returns:
`Index in str pointing at next position to read`

---

bscanf

```
public static int bscanf(String str,
                         int si,
                         String format,
                         char[] data)
```

---

bscanf

```
public static int bscanf(String str,
                         int si,
                         String format,
                         int[] data)
```

---

bscanf

```
public static int bscanf(String str,
                         int si,
                         char[] format,
                         int[] data)
```

---

bscanf

```
public static int bscanf(char[] str,
                         int si,
                         String format,
                         int[] data)
```

---

bscanf

```
public static int bscanf(char[] str,
                         int si,
                         char[] format,
                         int[] data)
```

bscanf

```
public static int bscanf(String str,
                         int si,
                         char[] format,
                         char[] data)
```

bscanf

```
public static int bscanf(char[] str,
                         int si,
                         String format,
                         char[] data)
```