`Csprintf` User's Manual

by

Software Industry & General Hardware
(sales@s-i-g-h.com)
(document reference: Csprintf20090313168273OPDT)

Table of Contents

Legalese

License

Software License Agreement

By using this software you are agreeing to be bound by this license agreement.

If you do not agree to be bound by this license, do not use this software.

Either delete software package completely or return the physical package,

including the distribution media in the original packaging, unopened, to the manufacturer at:

Software Industry & General Hardware
ATTN: PKG RETURN
23125 Crooked Arrow Drive
Wildomar, CA USA 92595

This package may not be used or redistributed as part of a commercial release under this license. Contact the owner of the software (S.I.G.H.) for fee information and commercial use licenses. (Contact information: email to **`sales@s-i-g-h.com`**)

Non-commercial use license:

You may use and/or redistribute the java package included in this distribution as is or as part of another software package owned by you as long as the following conditions are met:

1. The new package cannot alter the functionality or use of the existing package.
2. The new package must allow the user access to the "version()" method of the existing package.
3. The new package must allow the user access to the "license()" method of the existing package.
4. This Software License Agreement, in its entirety, must be included in your documentation.
5. The following notice must be included in your documentation.

"This software is COPYRIGHT © 2009 by Software Industry & General Hardware.
ALL RIGHTS reserved.

This software package includes and uses the **`Csprintf.jar`** package developed by:
Software Industry & General Hardware (S.I.G.H.).
For more information on this or any other software package from S.I.G.H.,
email info@s-i-g-h.com

This software may not be included in a commercial package."

This package may be distributed via a network.

THIS SOFTWARE IS PROVIDED BY SOFTWARE INDUSTRY & GENERAL HARDWARE “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION,

ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

USE OF THE SOFTWARE CONSTITUTES AGREEMENT TO CONFORM WITH THIS LICENSE AND TO BE BOUND BY THE LEGAL CONSEQUENCES OF VIOLATING THIS LICENSE.

Commercial Use Information

For information on commercial use of this software contact Software Industry & General Hardware at: sales@s-i-g-h.com, or write for information to:

Software Industry & General Hardware
23125 Crooked Arrow Drive
Wildomar, CA USA 92595

Non-commercial Use Information

Non-commercial use is governed by the License above.

What does all this mean?

If you are using the software for non-commercial use then have fun learning programming, writing that program to help others or anything you are not intending to sell for profit. Have fun and enjoy the software. Use it to your hearts content.

If you are planning on using this software as part of a package you are going to sell; you may not do so without contacting the owner of the software and obtaining a license from the Software Owner.

If you are planning on distributing this software as part of a non-profit or not-for-profit venture you may do so with impunity.

If you are going to distribute this software as part of a software collection for which you are not charging (except for a nominal fee for the media and/or shipping [no more than $10(US)]) you may do so.

If you are going to distribute this software as part of a software collection you are going to sell for profit you may not do so.

Williams' Notation

This notation was invented by the author to help solve some of the drudgery of note taking in Computer Science classes. The problem this notation solves is one familiar to most computer science students and practitioners. Often one must describe something that is a place-holder. Once very good example is describing something in BNF (Backus-Normal-Form if you have something against Mr. Naur; or Backus-Naur Form if you don't have anything against Mr. Naur [for a more complete discussion please see this Wikipedia entry). Or another is where the author wants to indicate that a file-name would be inserted into a particular position in a command line. For example:

“The student should type: **`cat <insert_a_file_name_here>`**”.

The problem isn't the **`insert_a_file_name_here`** clause but rather telling the student NOT to type the left and right angle-brackets. Williams notation is to simply note the non-entered entities at the end of the sentence, enclosed by some meta-character of its own. So the above sentence would become:

“The student should type: **cat <insert_a_file_name_here>**. **`/< >/`**”

The notation **`/< >/`** indicates the angle brackets are not typed and do not belong to the clause **insert_a_file_name_here**. You can also indicate that a punctuation or phrase is simply intended for emphasis and not to be considered part of the sentence itself. For example:

“The student should type **`cat <insert_a_file_name_here>`**”. **`/< > “ ”/`**

The space between the characters is intentional so that the characters are more easily spotted. This is called “visual acuity” and is something that the author wishes was more deeply considered by computer program language authors. Sometimes a bracket, brace or a parenthesis are indistinguishable in print or on a fading screen with glare: {([

Lastly in the notation the meta-character does not have to be a slash[1]. It may be any character you wish to use. Just use it in pairs.

Pseudo BNF

As mentioned in the “Williams' Notation” section BNF (Backus-Naur Form) gives us an easy grammatical ordering descriptive notation. Arcane to the un-initiated pseudo-BNF is simply a relaxation of the rules of BNF. <identifier> declares a descriptive name. A literal symbol is shown by itself and things enclosed in square brackets are optional. There are several common ways to indicate a space character. A ␣ (known as a shelf) character, a subscripted logical-and $_\wedge$ and for those of us who are familiar with keypunches there is the old IBM suggestion a small letter-b with a slash through it **b̸**.

The pseudo-BNF of [<name_of_day_of_week>$_\wedge$]<month>$_\wedge$<day>,$_\wedge$<year> should be familiar as a description of March 25, 2009 or optionally Wednesday March 25, 2009.

Pseudo-BNF is used throughout this document.

---

[1] Slash is the character “/”. /“ ”/ More properly known as a virgule. It is NOT the reverse-virgule or “\”. /“ ”/ This is another lunacy perpetrated by MS bigots. A / is a slash and a \ is a back-slash.

Notes And Abbreviations

KNR --- “The c Programming Language” reference, by Brian W. Kernighan and Dennis M. Ritchie, published by Prentice Hall.

Wikipedia --- http://www.wikipedia.com encyclopedia

ISO --- The Internal Standards Organization

IEEE --- The Institute of Electrical and Electronics Engineers

Overview

`Csprintf` is a java equivalent of most of the c programming language functionality found in sprintf. There are additionally some useful proprietary additions that are explained later in this manual. In general the rules are governed first by `sprintf` in KNR and secondly as defined in ISO C99 standard. Basic functionality can be discerned from the C99 reference to printf found on Wikipedia at: http://en.wikipedia.org/wiki/Printf. See the KNR and C99 inclusions sections of this manual for more specific information. This manual assumes the reader is familiar and comfortable with the c programming language statements `printf`, `sprintf` and their use.

KNR

Inclusions

In the c programming language the statement “**printf**” consists of the following general syntax: `printf (“<format_place_holder>...”, <variable_identifier>...);` /“ ” **...**/ Where **...** means zero or more. In general a <**format_place_holder**> has the following syntax: **"*%[parameter][flags][width][.precision][length]type*"** `Csprintf` accepts and processes the types: c, d, e, f, i, g, o, p, s and x (and the upper-case equivalents) as well as the following proprietary types: `0t`, `1t`, `2t`. `Csprintf` also includes a proprietary flag “,”. /“ ”/ The comma flag and t-types are discussed in the Proprietary Inclusions section.

Exclusions

The option “n” is not included in gcc. /“ ”/ This option returns the number of characters output so far to an argument although the argument count is not incremented. What does all of this mean?

Consider the format string:
`“%s%n means we have written %d characters.\n”`

Now consider the following code fragment instruction:

```
int* myStrLen = 0;
char myStr[] = “ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890”;
printf ( “%s%n means we have written %d characters.\n”, myString, myStrLen);
```

The output should be:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890 means we have written 36 characters.
```

Try this in gcc and you will get a Bus error.

If you investigate this option you will find that gcc does not implement `%n`. However...`Csprintf` does!

Try this code in Java (make certain you have included com.sigh.jar in the CLASSPATH).

```
public void testPercentN() {
  String   alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  String   digits   = "1234567890";
  int      aValue   = 0;
  Csprintf csf = new Csprintf();
  Object[] values = new Object [ 3 ];

  values [ 0 ] = alphabet;
  values [ 1 ] = digits;
  values [ 2 ] = (Object) aValue;
  try {
      System.out.println ( csf.doSprintf ( "%s%s is %d characters long",   values ) );
      System.out.prtinln ( csf.doSprintf ( "%s%s%n is %d characters long", values ) );
  } catch ( CsprintfExceptions csfE ) {
      csfE.printStackTrace();
  }
}
```

Note two things. The first is that new Object has only 3 elements. The reason is that `%n` is defined as not incrementing the option count. That is, the value returned by `%n` apples to the next option processed. This means that a `%n%n` must be followed by another **%-option**. The value returned is defined in KNR as an int* so a `%n` must be followed by a **%-option** that is not a **%s** nor a **%c** (actually **%c** might make sense, but only for values less than 256 so the author has simply disallowed it; use a **%d** if you are desperate). Also a `%n` may not be followed by a `%n`. Nor may a `%n` be the final **%-option** in a format string. Here are the three rules:

1. `%n` may not exist as the last **%-option** in a format string.
2. `%n` may not be followed by `%n` (with or without intervening characters).
3. `%n` may not be followed by **%c** nor **%s** (with or without intervening characters).
4. `%n` must be followed by a numeric %-options of some kind (usually a **%d**, **%i** or `%u`).

Percent-n is tricky. The gcc compiler does not implement it. The author found very little supporting documentation, no examples and several reasons NOT to implement percent-n.

Still, **%n** is included in **Csprintf**.

C99

Inclusions

`Csprintf` follows the KNR description of `printf`, `sprintf` and avoids the C99 extensions. Especially avoided are the manufacturer specific extensions `DWORD`, etc.

Proprietary Inclusions

Flags

The proprietary addition of the comma allows the programmer to include a separator (defaulted to the local locale). In North America this means a number such as 1234567890 would be returned as 1,234,567,890. /./ In some places in the world the number would be returned as 1.234.567.890 (see also: "Methods" chapter for varying the separator count and the separator character).

Types

`Csprintf()` adds the following additional type **`%t`**.

In *KNR* time functions are handled in the standard c-library **`<time.h>`** and include a specific function **`strftime()`** and a completely new set of format strings to format the date/time strings. The most probable reason for the new set of format strings in **`strftime()`** is the fact that types are made up of a percent-sign and a single letter. Even with 26-letters in the alphabet you run out quickly. As a result **`%c`** in **`sprintf()`** and **`%c`** in **`strftime()`** mean two different things. The question is how to combine the two? **`Csprintf()`** and **`CTime()`** combine these two format string characters via a class named **`TimeFormatString`**. **`TimeFormatString`** consists of a constructor and two getter-methods. These encapsulate two properties named **`formatString`** (a Java String type) and **`longTimeInMilliseconds`** (a Java long type). Instantiation is via the constructor which requires two parameters, String and long. The getters are available to allow the user to test the current values being used. The two getter-methods are **`getFormatString()`** and **`getTimeInMilliseconds()`**(these two methods should be self-explanatory).

`TimeFormatString ( String, long )` takes a **`strftime()`** format string as the String parameter and a long (64-bit) value as the second parameter. The String parameter values for the **`strftime()`** format string can be found in the ***CTime*** *User's Manual*. The long parameter represents a time value in milliseconds the user wants to apply to the format string of the first parameter.

A user has then two ways to include a date/time string in the desired output. One is to write a separate **`strftime()`** call and save that string value. The other is by using the **`TimeFormatString`** object and including it as the object to a **`%t`** type in a call to **`sprintf()`**.

As an example supposed you want todays date and time in your output string. In **`Csprintf()`** you could do the following code:

```
{
    Object[] theValues = new Object [ 1 ];
    Calendar aCalendar = Calendar.getInstance();
    TimeFormatString tfs = new TimeFormatStirng ( "%c", aCalendar.getTimeInMills() );
    theValues [ 0 ] = (Object) tfs;
    System.out.println ( Csprintf.sprintf ( "Todays date is ", theValues ) );
}
```

Using **`Ctime()`** you could do the following:

```
{
    System.out.println ( CTime.strftime ( "Today's date is %c" ) );
}
```

As you can see using `CTime()` can be easier if all you are doing is displaying date/time. But if you are displaying other values along with the date/time (as most people do) then you may do so from `Csprintf()`.

Methods

`Csprintf()` --- Use to instantiate an instance of `Csprintf()`. This method establishes the local locale grouping characters and grouping size.

`doSprintf(String, Object[])` --- Non static call used with an instantiated object. String is the c programming language equivalent format string. `Object[]` is an vector of object values to apply to the format string.

`set3DigitSeparatorString(String)` --- This method allows the programmer to set a new separator string. It returns the old separator string value so you can save and restore the previous separator string.

`set3DigitSeparatorString(char)` --- Same as above except allowing the programmer the convenience of using a single character instead of a string.

`setFractionalSeparatorChar(String)` --- Allows the programmer to specify the fractional separator string in floating point numbers. North America typically uses a period for this character and Europe typically uses a comma.

`setFractionalSeparatorChar(char)` --- Same as above except allowing the programmer the convenience of using a single character instead of a string.

`sprintf(String, Object)` --- Static call where the first String is the c programming language equivalent format string. Object is a single object value to apply to the format string.

`sprintf(String, Object, String)` --- Same as above except that the second String is the separator string to apply to this static call only.

`sprintf(String, Object, String, int)` --- Same as above except that the int is the number of characters to group together instead of using the local locale default for this static call only.

`sprintf(String, Object, int)` --- Static call where the first String is the c programming language equivalent format string. Object is a single object value to apply to the format string and int is the number of characters to group together instead of using the local locale default for this static call only.

`sprintf(String, Object[])` --- Static call used with an instantiated object. String is the c programming language equivalent format string. `Object[]` is a vector of object values to apply to the format string.

`sprintf(String, Object[], String, int)` --- Same as above except the second String is the separator string and the int is the number of characters to group together instead of using the local locale default for this static call only.

`sprintf(String, Object[], String)` --- Same as above except the second String is the separator string.

`sprintf(String, Object[], int)` --- Same as above except the int is the number of characters to group together instead of using the local locale default for this static call only.

`setNumberOfDigitsToSeparate(int)` --- For non-static calls sets the number of characters to group together before a separator string is applied. Returns the previous value.

`getNumberOfDigitsToSeparate()` --- Returns the current number of digits that are grouped together before a separator string is inserted into the numeric string.

`get3DigitSeparatorString()` --- Returns the current grouping separator string.

`getFractionalSeparatorString()` --- Returns the current separator string for the fractional separator.

`set3DigitSeparatorChar(String)` --- Sets the grouping separator string and returns the current grouping separator string.

`setFractionalSeparatorChar(String)` --- Sets the fractional separator string and returns the current separator string.

Code Examples

Csprintf()

```
Object[] testValues = new Object [ 3 ];
Csprintf csf = new Csprintf();
testValues [ 0 ] = 512;
testValues [ 1 ] = 511;
testValues [ 2 ] = 256;

System.out.println ( csf.doSprintf ( "0%o 0%o 0%o", testValues ) );
System.out.println ( csf.doSprintf ( “%#o %#o %#o”, testValues ) );

Object[] anotherSpecialTestValue = new Object [ 1 ];
aTestValue [ 0 ] = new Short ( (short) -1 );
System.out.println ( Csprintf.sprintf ( "%u", aTestValue ) );

specialTestValues [ 0 ] = new Float  ( 0.0 );
specialTestValues [ 1 ] = new Double ( 0.0 );
System.out.println ( csf.doSprintf ( "zero float %f double %f", testValues ) );
```

```
Will output:
01000 0777 0400
01000 0777 0400
65535
zero float 0.000000 double 0.000000
```

doSprintf(String, Object[])

```
Object[] testValues = new Object [ 3 ];
Csprintf csf = new Csprintf();
testValues [ 0 ] = 512;
testValues [ 1 ] = 511;
testValues [ 2 ] = 256;

System.out.println ( csf.doSprintf ( "0%o 0%o 0%o", testValues ) );
System.out.println ( csf.doSprintf ( “%#o %#o %#o”, testValues ) );

Object[] aTestValue = new Object [ 1 ];
aTestValue [ 0 ] = new Short ( (short) -1 );
System.out.println ( Csprintf.sprintf ( "%u", aTestValue ) );

testValues [ 0 ] = new Float  ( 0.0 );
testValues [ 1 ] = new Double ( 0.0 );
System.out.println ( csf.doSprintf ( "zero float %f double %f", testValues ) );
```

Will output:

```
01000 0777 0400
01000 0777 0400
65535
zero float 0.000000 double 0.000000
```

set3DigitSeparatorString(String)

```
Csprintf csf = new Csprintf();
String oldcsf.set3DigitSeparatorString ( “<*>”  );
System.out.println ( csf.sprintf ( “%,Lu”, 1234567890 ) );
```

Will output (for Babylon 5 fans):

```
1<*>234<*>567<*>890
```

set3DigitSeparatorString(char)

setFractionalSeparatorChar(String)

setFractionalSeparatorChar(char)

sprintf(String, Object)

```
System.out.println ( Csprintf.sprintf ( "%,Lu", 1234567890L  ) );
System.out.println ( Csprintf.sprintf ( "%,Li", -1234567890L ) );
System.out.println ( Csprintf.sprintf ( "%10g", new Double ( -0.156400000007 ) ) );
System.out.println ( Csprintf.sprintf ( "%10g", new Double (  0.156400000007 ) ) );
System.out.println ( Csprintf.sprintf ( "%+10g", new Double ( 0.156400000007 ) ) );

System.out.println ( Csprintf.sprintf ( "%12.2e", new Double   ( -0.15640000000007 ) ) );
System.out.println ( Csprintf.sprintf ( "%12.2E", new Double   ( -0.15640000000007 ) ) );
System.out.println ( Csprintf.sprintf ( "%12.2e", new Double   (  0.15640000000007 ) ) );
System.out.println ( Csprintf.sprintf ( "%12.2E", new Double   (  0.15640000000007 ) ) );
System.out.println ( Csprintf.sprintf ( "%12.3e", new Double   ( -0.15640000000007 ) ) );
System.out.println ( Csprintf.sprintf ( "%12.3E", new Double   ( -0.15640000000007 ) ) );
System.out.println ( Csprintf.sprintf ( "%12.3e", new Double   (  0.15640000000007 ) ) );
System.out.println ( Csprintf.sprintf ( "%12.3E", new Double   (  0.15640000000007 ) ) );
System.out.println ( Csprintf.sprintf ( "%12.3e", new Integer  (  1                 ) ) );
```

```
    System.out.println ( Csprintf.sprintf ( "%12.2e", new Double  (  0.0              ) ) );
    System.out.println ( Csprintf.sprintf ( "%12.3E", new Double  (  0.0              ) ) );
            // Now force a plus sign
    System.out.println ( Csprintf.sprintf ( "%+12.2e", new Double (  0.15640000000007 ) ) );
    System.out.println ( Csprintf.sprintf ( "%+12.2E", new Double (  0.15640000000007 ) ) );
    System.out.println ( Csprintf.sprintf ( "%+12.3e", new Double (  0.15640000000007 ) ) );
    System.out.println ( Csprintf.sprintf ( "%+12.3E", new Double (  0.15640000000007 ) ) );
    System.out.println ( Csprintf.sprintf ( "%+12.2e", new Double (  0.0              ) ) );
    System.out.println ( Csprintf.sprintf ( "%+12.3E", new Double (  0.0              ) ) );
    System.out.println ( "% can stand alone Yes they can!"
                     , sprintf ( "%% can stand alone %s", new String ( "Yes they can!") ) );
```

Will output:

```
1,234,567,890
-1,234,567,890
   -0.1564
    0.1564
   +0.1564
   -1.56e-01
   -1.56E-01
    1.56e-01
    1.56E-01
  -1.564e-01
  -1.564E-01
   1.564e-01
   1.564E-01
NSF
    0.00e+00
   0.000E+00
   +1.56E-01
   +1.56e-01
  +1.564e-01
  +1.564E-01
   +0.00e+00
  +0.000E+00
% can stand alone Yes they can!%% can stand alone Yes they can!
```

`Note: The NSF is generated by the line containing: "%12.3e", new Integer ( 1 ). Integer is not of base type double.`

sprintf(String, Object, String)

```
System.out.println ( Csprintf.sprintf ( “%#,0o”, -1L, “ ” ) );
System.out.println ( Csprintf.sprintf ( “%012,o”, 4269801473L, " " ) );
System.out.println ( Csprintf.sprintf ( “%,0o”, 2814749767106561L, “.” ) );
```

Will output:

```
0o1 777 777 777 777 777 777 777
037 640 000 001
120.000.000.000.000.001
```

sprintf(String, Object, String, int)

```
System.out.println ( Csprintf.sprintf ( "0x%0,x", -1L, " ", 4 ) );
System.out.println ( Csprintf.sprintf ( "%0,x", 4269801473L, " ", 4 ) );
System.out.println ( Csprintf.sprintf ( "%,0x", 2814749767106561L, ":", 4 ) );
```

Will output:

```
0xffff ffff ffff ffff
fe80 0001
0010:0000:0000:0001
```

sprintf(String, Object, int)

```
System.out.println ( Csprintf.sprintf ( "%,0x", 2814749767106561L, 4 ) );
```

Will output:

```
0010,0000,0000,0001
```

sprintf(String, Object[])

```
Object[] anObject = new Object [ 1 ];
anObject [ 0 ] = 1L;
System.out.println ( sprintf ( "Example of SIGH's sprintf: %06ld", anObject ) );
```

Will output:

```
Example of SIGH's sprintf: 000001
```

sprintf(String, Object[], String, int)

sprintf(String, Object[], String)

sprintf(String, Object[], int)

setNumberOfDigitsToSeparate(int)

```
Csprintf csf = new Csprintf();
int oldValue = csf.getNumberOfDigitsToSeparate();
csf.setNumberOfDigitsToSeparate ( 4 );
csf.set3DigitSeparatorChar ( ' ' );
System.out.println ( "Previous number of digits to separate: " + oldValue );
tempObj [ 0 ] = new Long ( -1L );
System.out.println ( csf.doSprintf ( "%#,x", tempObj ) );
tempObj [ 0 ] = new Short ( (short) 23130 );
System.out.println ( csf.doSprintf ( "%0,x", tempObj ) );
System.out.println ( "Previous number of digits to separate: " +
                    csf.setNumberOfDigitsToSeparate ( oldValue ) );
csf.setLocalLocaleForBothSeparators();
```

Will output:

```
Previous number of digits to separate: 3
0xffff ffff ffff ffff
5a5a
Previous number of digits to separate: 4
```

getNumberOfDigitsToSeparate()

```
int widget = Csprintf.getNumberOfDigitsToSeparate();
```

get3DigitSeparatorString()

getFractionalSeparatorString()

set3DigitSeparatorChar(String)

setFractionalSeparatorChar(String)

C vs Java And Types

Java has the following integral types and values:

`byte`	-128 to 127, inclusive
	8-bits
`short`	-32768 to 32767, inclusive
	16-bits
`int`	-2147483648 to 2147483647, inclusive
	32-bits
`long`	-9223372036854775808 to 9223372036854775807, inclusive
	64-bits
`char`	`'\u0000'` to `'\uffff'` inclusive, that is, from 0 to 65535
	16-bits

Java has the following floating-point types and values for float and double:

Parameter	float	float-extended-exponent	double	double-extended-exponent
N	24	24	53	53
K	8	≥11	11	≥15
E_{max}	127	≥+1023	1023	≥+16383
E_{min}	-126	≤-1022	-1022	≤-16382

If you are a c programmer you may be struck by the fact that there is no unsigned (except for char). This is correct. So **`Csprintf`** must “fake” an unsigned value when requested. C99 has long long types and more. This is true. Java does not. How you get your c data into a Java value via JNI is up to you. Once you do you may apply the attributes of unsigned, long and short as you wish. **`Csprintf`** will do its best to evaluate this according to Java types and return a string representing what was requested. There are of course caveats. The best answer is to try a short example of what you are trying to do and see what the output looks like.

Caveats

%g

The formatting string "`%g`" is unusual to say the least. /" "/ KNR indicates that `%g` differs slightly from fixed-point notation in that insignificant zeroes to the right of the decimal point are not included and that the decimal point is not included on whole numbers. All of this changes if you use the alternate form flag of "`#`". /" "/

KNR also indicates that whether `%f` equivalent or the `%e` equivalent type is used should be determined by the esthetics of the display. **`Csprintf`** does a fairly reasonable job of determining the changeover point. There is no "fuzzy-logic" control of the changeover point.

Common Misconceptions

The "`l`" (letter lowercase el) attribute modifies the format string so that the value is displayed as a long value. It does not change the value itself. /" "/ This doesn't matter in the c programming language since a long and an int are the same length (32-bits). So the following code in c would produce the following output:

```
#include <stdio.h>
#include <stdlib.h>

int main ( int argc, char** argv )
{
  long widget = -1L;
  int  wodget = -1;

  printf ( "%u %u\n", sizeof(widget), sizeof(wodget) );
  printf ( "%u %u\n", widget, wodget );
  return 0;
}
```

```
4 4
4294967295 4294967295
```

Both a long and an int are 4-bytes long (32-bits) and each has the same unsigned maximum value of 4,294,967,295.

Java is different. An int is 32-bits and a long is 64-bits long. **`Csprintf`** follows the Java values and the c programming language's philosophy of the "`l`" modifier when applied to values. /" "/

The following code:

```
public class example0 {
    public example0() {
    }
    public static void main ( String[] argv ) {
        try {
           System.out.println ( Csprintf.sprintf ( "%lu", new Byte    ( (byte)  -1 ) ) );
           System.out.println ( Csprintf.sprintf ( "%lu", new Short   ( (short) -1 ) ) );
           System.out.println ( Csprintf.sprintf ( "%lu", new Integer ( (int)   -1 ) ) );
           System.out.println ( Csprintf.sprintf ( "%lu", new Long    ( (long)  -1 ) ) );
        } catch ( CsprintfExceptions csfE ) {
            csfE.printStackTrace();
        }
    }
}
```

Will output:

```
255
65535
4294967295
18446744073709551615
```

The long modifier should prefix the u flag not suffix the u flag. For example “**`%ul`**” as a format string will produce <value>l where as “**`%lu`**” will produce <value_as_unsigned_long>. /“ ” < >/

The best thing to do is to TRY the format string and see what you get as output. If you disagree with the results then see what the gcc compiler or g++ compiler output is. You may be surprised. If you are convinced your display is correct then see the Support chapter and send it in for an evaluation.

Exceptions

`Csprintf` throws the following exceptions:

- **`CsprintfExceptions`**
    - Consisting of the following possible exception characteristics:
        - UNKNOWN_EXCEPTION
        - INCORRECT_FORMAT_STRING
        - INCORRECT_VALUE_FOR_FORMAT_STRING
        - INCORRECT_RETURN_VALUE_TYPE
        - INCORRECT_VALUES_FOR_ASTERISK_FORMAT
        - INVALID_PERCENT_N_COUNT_OR_LOCATION
- ArrayOutOfBoundsException
- NullPointerException

If you encounter one of these please send the stack trace to: info@s-i-g-h.com along with the items listed in the Support chapter.

All exceptions and any output from them are written to the System.err stream.

Csprintf is designed mostly for developers converting c code to Java. You don't need a production log nor a file. Just redirect the System.err output to a file if you need to do so.

If you catch the exception you will need to do your own call to printStackTrace(). CsprintfExceptions does not do so in the class.

Errors

If `Csprintf` cannot parse the format string or if the value is not represented in the base type of the format string `Csprintf` returns `NSF`. `NSF` stands for "No Such Formatting". Since numbers are converted to strings and the strings are then manipulated there is a possibility that `Csprintf` could throw a Java ArrayOutOfBoundsException; please see the Support chapter if you experience this.

Support

If you believe that `Csprintf` is displaying an incorrect value send the following to info@s-i-g-h.com:

- The actual numeric value(s) you used with `Csprintf`
- The format string value you used with `Csprintf`
- The actual code line of the call to `Csprintf` (just one or two lines please we don't need or want the entire project).
- What you believe `Csprintf` should have displayed as the result of the formatting string.

You should get a response within 48 hours.

Recourse

Software Industry & General Hardware attempts to provide the most bug-free software it can, no project is completely bug free or feature complete. We will listen to and collect suggestions but implementation of the those suggestions is solely at the discretion of Software Industry & General Hardware.

This software is supplied "AS IS". You use this software at your own risk.

Software Industry & General Hardware is not responsible for perceived or actual loss of data, time nor anything else whether to you or your customers/users.

There is no recourse. You're using this for free.

The commercial version is governed by a different set of rules.