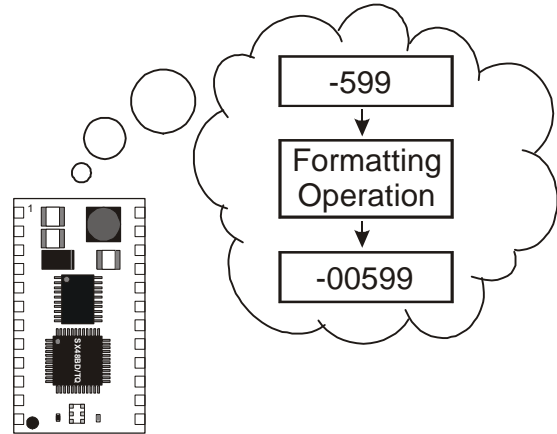


599 Menlo Drive, Suite 100
 Rocklin, California 95765, USA
Office/Tech Support: (916) 624-8333
Fax: (916) 624-8003

Web Site: www.javelinstamp.com
Home Page: www.parallaxinc.com

General: info@parallaxinc.com
Sales: sales@parallaxinc.com
Technical: javelintech@parallaxinc.com



Contents

Introduction to Formatting Text for the Javelin	2
Downloads, Parts, and Equipment for the Format Library	2
The Format Library and How it Works.....	2
Formatted Text	3
Scanning Strings.....	3
Integer/String Conversions	3
Helper methods	4
Testing the Format library.....	4
Program Listing 1.1 – The FormatTest	5
Formatting Example.....	5
Program Sections 1-9 of 12	6
Program Sections 10 and 11 of 12.....	7
Program Section 12	8
Program Listing 1.2 – FormatExample.java.....	9
Format Demo	11
Published Resources – for More Information	11
Javelin Stamp Discussion Forum – Questions and Answers.....	11

Introduction to Formatting Text for the Javelin

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`.

Downloads, Parts, and Equipment for the Format Library

This application note (AppNote012-Format.pdf), the Format library file (Format.java), the library's javadoc file (Format.pdf), the test program (FormatTest.java), the demonstration program (FormatDemo.java) which will demonstrate all the methods available to you in the Format library class, and a formatting example (FormatExample.java) are all available to you for free download from the Application page from www.javelinstamp.com

You can use the AppNote012-Format.exe, to install the files listed below. These files must be located in specific paths within the Javelin Stamp IDE directory. Although the path to this directory can be different, the default root path is: C:\Program Files\Parallax Inc\Javelin Stamp IDE

The file list below is organized by directory, then by filename, please verify that your file list is organized in the same way.

```
<root path>\doc\AppNote012-Format.pdf
<root path>\doc\Format.pdf
<root path>\lib\stamp\util\text\Format.java
<root path>\Projects\examples\util\text\FormatTest.java
<root path>\Projects\examples\util\text\FormatDemo.java
<root path>\Projects\examples\util\text\FormatExample.java
```

The equipment used to test this example includes a Javelin Stamp, Javelin Stamp Demo Board, 7.5 V, 1000 mA DC power supply, serial cable, and PC with the Javelin Stamp IDE v2.01.

The Format Library and How it Works

This Format library is similar in functionality and form to the original C libraries. Additional methods have been included for more functionality for the Javelin. There are four general types of methods available for you to use, formatted text (printing), formatted scanning (of strings), integer/string conversions (`itoa`/`atoi`) and helper methods.

Formatted Text

Prints a formatted string to the message window (stdout), or character buffer using format specifiers. A format specifier begins with a percentage sign, and ends with a particular character, which represents various data types within the Javelin. Here is a list of the available format specifiers:

- 1) **%c** – character formatter for byte, char, short, and int
- 2) **%d** – decimal (same as %i) formatter for byte, char, short, and int
- 3) **%i** – integer formatter for byte, char, short, and int
- 4) **%b** – byte formatter for byte, char, short, and int
- 5) **%o** – octal formatter for byte, char, short, and int
- 6) **%u** – unsigned formatter for byte, char, short, and int
- 7) **%x** – hexadecimal formatter for byte, char, short, and int
- 8) **%s** – string formatter for Strings

For more information on format specifiers and to view working examples, please see the Format Demo section at the end of this application note.

These format specifiers have special field specifications that allow you to justify your data (left/right), add leading or trailing zeros, set a minimum/maximum fixed width and allow for signed integers. Below is a list of field specifiers along with a description of how to use them with the %i format specifier:

%-0min.maxi

- 1) '-' – for left justify (omit for right justification, default).
- 2) '0' – pad with zeros (omit to pad with spaces, default).
- 3) min – You can specify the minimum field width (omitted by default).
- 4) max – You can specify the maximum field width (if omitted, do not include field separator).
- 5) '.' – This is a field separator which separates the min and max field specifiers (omit if max not specified).

For more information on field specifiers and working examples please see Program Listing 1.2.

Scanning Strings

These methods allow you to scan a sprintf buffer and return the value for a specific format specifier. For more information on format specifiers please see previous section titled *Formatted Text*.

Integer/String Conversions

These methods are used to convert integers to strings and strings to integers. They will work on signed values and include the ability to convert integers and strings to binary, octal, decimal, or hexadecimal. These methods

get called from within other internal private library methods. Since you may wish to use them directly without formatting, we have made them available for your use.

Helper methods

These methods are used within the library class, but since they can be so helpful we decided to keep them public.

- 1) `isDigit` – Will test if a character is a digit.
- 2) `isSpace` – Will test if character is a space (0x20), tab (\t) or newline (\n)
- 3) `reverse` – Will reverse the characters within a String.
- 4) `strLen` – Will get the length of null terminated string in a character array.

Testing the Format library

Program Listing 1.1 is a short program that will demonstrate a few **printf** formatting examples.

We begin by initializing the integer **var** to the value -34.

```
int var = -34;
```

Next we simply print a message to the Javelin's terminal window.

```
System.out.println("Testing printf");
```

Now we will print a message to the Javelin's terminal window with an embedded variable. The message that you would like displayed is your 1st argument, your 2nd argument is the variable who contents you would like displayed where the format specifier (**%d**) is located.

```
Format.printf("Print variables like %d, within a message.\n", var);
```

The next command will print a message and an integer, but this time the integer will be printed within a 5 character field (denoted by the '5' within **%05d**), since the integer is 1 character, that leaves you with 4 characters in the field. By default the integer will be placed on the right side of the field with spaces on the left, but in this example we will substitute zeros for the spaces (notice the '0' within **%05d**).

```
Format.printf("Pad fixed output with %05d zeroes.\n",4);
```

The next command will print a message, and substitute the character 'A' for the **%c** format specifier.

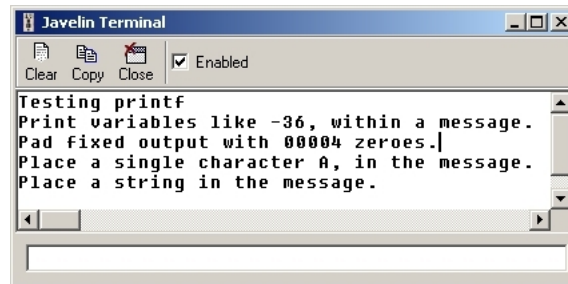
```
Format.printf("Place a single character %c, in the message.\n",'A');
```

The last command will print a message, and substitute the string 'string' for the **%s** format specifier.

```
Format.printf("Place a %s in the message.\n","string");
```

Figure 1.1 is the output for the test program, Program Listing 1.1.

Figure 1.1
Integer segment of
FormatTest.java



Program Listing 1.1 – The FormatTest

```
/* Simple test program to demonstrate a few formatting examples.
 *
 * For more information see AppNote012 from Parallax Inc.
 * Version 1.0 Feb. 24th, 2003 (an012)
 */

import stamp.util.text.*;
public class FormatTest{

    public static void main() {
        int var = -36;

        System.out.println("Testing printf");
        Format.printf("Place variables like %d, within a print message.\n",var);
        Format.printf("Pad fixed output with %05d zeroes.\n",4);
        Format.printf("Place a single character %c, in the message.\n",'A');
        Format.printf("Place a %s in the message.\n","string");
    }
}
```

Formatting Example

Program Listing 1.2 is a good example on formatting various data types using the **printf** method. It runs through all of the format specifiers, and formats them for left/right justification with a fixed width with and without padded zeros. It shows you how you can take an integer and using just the **printf** method not only format but also convert the integer to binary, octal, unsigned, and hexadecimal values.

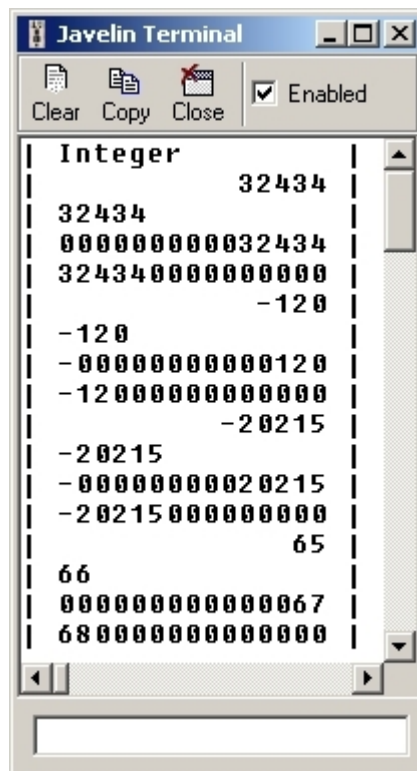
The code of this program is broken up into 12 sections; there is a blank line and a comment that precedes each section. Each section will have a brief explanation.

Program Sections 1-9 of 12

Sections 1 through 9 all have similar characteristics, which are:

- All the lines are fixed to a minimum field width of 15.
- Each line contains a new line (`\n`) field and is bordered with a horizontal bar (`|`).
- The 1st line of code uses the string specifier (`%s`) to print the title.
- The 2nd line of code will display the value to the far right of the field width (default).
- The 3rd line of code will display the value to the far left by using the negative sign (`-`) immediately after the `%` declaration.
- The 4th line of code will display the value to the far right (default) and pad the left with zeros.
- The 5th line of code will display the value to the far left and pad the right with zeros.

Figure 1.2 is the output for the Integer segment; all other segments will be similar.



```
| Integer |
|          |
|          | 32434 |
| 32434    |          |
| 0000000000032434 |
| 3243400000000000 |
|          | -120 |
| -120     |          |
| -000000000000120 |
| -1200000000000000 |
|          | -20215 |
| -20215   |          |
| -0000000000020215 |
| -2021500000000000 |
|          | 65 |
| 66       |          |
| 0000000000000067 |
| 6800000000000000 |
```

Figure 1.2
Integer segment of
FormatExample.java

Section 1: Integer formatting of a positive integer value

This section prints a signed integer with the value of 32434 using the format specifier of an integer (**%i**).

Section 2: Integer formatting of a negative integer value

This section prints a signed integer with the value of -120 using the format specifier of an integer (**%i**).

Section 3: Integer formatting of a short integer value (output will be signed)

This section prints a short value of 45321 using the format specifier of an integer (**%i**). The output will be a negative number (-20215) since the format specifier (**%i**) is a signed integer. *If you would like the output to be unsigned you may use the unsigned format specifier (**%u**).*

Section 4: Integer formatting of a character (will be converted to ASCII)

This section prints a character using the format specifier of an integer (**%i**). The output will be an integer value that represents the particular character being printed. For a list of character values and their corresponding values refer to an ASCII chart. *Notice the value 65 corresponds with the capital A, 66 with B, 67 with C, etc.*

Section 5: Character formatting of a character

This section prints a character using the format specifier for a character (**%c**).

Section 6: String formatting of a string

This section prints a string using the format specifier for a string (**%s**).

Section 7: Hexadecimal formatting of integer

This section converts an integer to a hexadecimal number then prints it using the format specifier for a hexadecimal number (**%x**).

Section 8: Unsigned formatting of short

This section prints a short as an unsigned integer using the format specifier for an unsigned number (**%u**).

Section 9: Octal formatting of short

This section converts an integer to an octal number then prints it using the format specifier for an octal number (**%o**).

Program Sections 10 and 11 of 12

The only difference between sections 10-11 and sections 1-9 is the fixed minimum field width. This width, which was set to 15 in the previous sections, has been extended to 25 to accommodate binary numbers.

Section 10: Binary formatting of an integer

This section converts an integer to a binary number then prints it using the format specifier for a binary number (**%b**).

Section 11: Binary formatting of a short integer

This section converts a short to a binary number then prints it using the format specifier for a binary number (%b).

Program Section 12**Section 12: Integer field formatting**

This last section will display two columns; the 1st column will contain several numbers, and the other column will contain their formatted counterpart. The formatting here will display a sign (+/-), all be lined up and left justified. By default the Format class does not print the positive sign, but we have included code to show you how to do this if it suits your needs.

Here is the output, shown in Figure 1.3, for this section of code.

Figure 1.3
Integer field formatting of
FormatExample.java

```
Signed Formatting
Before      After
-12 --> -12
145 --> +145
1356 --> +1356
-31356 --> -31356
-10112 --> -10112
-12 --> -120000000
145 --> +145000000
1356 --> +135600000
-31356 --> -313560000
-10112 --> -101120000
```

There are two near identical loops, the code in the 2nd loop is exactly the same as the 1st except for the '0' in the '%-09d' format specifier on the 3rd line of code which pads the output with zeros.

To print one line of output for this example will require three lines of code. The first line of code will take the signed integer and display it within a fixed minimum field width. The field width of '7' was selected since the largest integer including the minus sign would be 6 digits; the 7th digit is a space.


```
Format.printf("|%7d",i);
```

The next line will display a space, the arrow, another space, and then the sign (+/-). The sign will be determined by checking if the value to be printed is negative (`i<0`), if it is (?) then it will print a negative sign (`'-'`), if it is not (:) it will print a positive sign (`'+'`). This type of if-then-else statement is called *ternary* and its output, in this case either a positive or negative sign, will be passed as the parameter for the called method.

```
Format.printf(" --> %c",i<0?'-':'+');
```

The last line will display the number within a fixed left justified field (the 2nd loop will also pad this field with zeros); this fixed field will have a minimum width of 9. Again we will use the ternary if-then-else statement, but for a different reason. Since we have already printed the sign, we must now remove the sign from any negative number. If we did not we would have two negative signs. So for this statement we simply test to see if the value `i` is less than zero (`i<0`), and if it is (?), it will negate the value of `i` (`-i`), if it is not it will simply return `i`.

```
Format.printf("%-9d | \n",i<0?-i:i);
```

This program uses the `printf` method from the `Format` library class, there are many overloaded `printf` type methods, please refer to the JavaDocs for which methods are available for you.

Program Listing 1.2 – FormatExample.java

```
/* Format Example to demonstrate justification within a fixed width.
 *
 * For more information see AppNote012 from Parallax Inc.
 * Version 1.0 Feb. 24th, 2003 (an012)
 */

import stamp.util.text.*;

public class FormatExample{

    public static void main() {

        // Integer formatting of positive integer value.
        Format.printf("| %-15s |","Integer");
        Format.printf("\n| %15i |",32434);           // right justified
        Format.printf("\n| %-15i |",32434);           // left justified
        Format.printf("\n| %015i |",32434);           // with leading zeros
        Format.printf("\n| %-015i |",32434);           // with trailing zeros

        // Integer formatting of negative integer value.
        Format.printf("\n| %15i |",-120);
        Format.printf("\n| %-15i |",-120);
        Format.printf("\n| %015i |",-120);
```

```
Format.printf("\n| %-015i |",-120);

// Integer formatting of short integer value (output will be signed).
Format.printf("\n| %15i |",(short)45321);
Format.printf("\n| %-15i |",(short)45321);
Format.printf("\n| %015i |",(short)45321);
Format.printf("\n| %-015i |",(short)45321);

// Integer formatting of character (will be converted to ASCII).
Format.printf("\n| %15i |",'A');
Format.printf("\n| %-15i |",'B');
Format.printf("\n| %015i |",'C');
Format.printf("\n| %-015i |",'D');

// Character formatting of character.
Format.printf("\n\n| %-15s |","Character");
Format.printf("\n| %15c |",'A');
Format.printf("\n| %-15c |",'B');
Format.printf("\n| %015c |",'C');
Format.printf("\n| %-015c |",'D');

// String formatting of string.
Format.printf("\n\n| %-15s |","String");
Format.printf("\n| %15s |","Sammy Sam");
Format.printf("\n| %-15s |","Sammy Sam");
Format.printf("\n| %015s |","Sammy Sam");
Format.printf("\n| %-015s |","Sammy Sam");

// Hexadecimal formatting of integer.
Format.printf("\n\n| %-15s |","Hexadecimal");
Format.printf("\n| %15x |",14990);
Format.printf("\n| %-15x |",14990);
Format.printf("\n| %015x |",14990);
Format.printf("\n| %-015x |",14990);

// Unsigned formatting of short.
Format.printf("\n\n| %-15s |","Unsigned");
Format.printf("\n| %15u |",(short)54937);
Format.printf("\n| %-15u |",(short)54937);
Format.printf("\n| %015u |",(short)54937);
Format.printf("\n| %-015u |",(short)54937);

// Octal formatting of short.
Format.printf("\n\n| %-15s |","Octal");
Format.printf("\n| %15o |",(short)54937);
Format.printf("\n| %-15o |",(short)54937);
Format.printf("\n| %015o |",(short)54937);
Format.printf("\n| %-015o |",(short)54937);

// Binary formatting of integer.
Format.printf("\n\n| %-25s |","Binary");
Format.printf("\n| %25b |",30433);
```

```

Format.printf("\n| %-25b |", 30433);
Format.printf("\n| %025b |", 30433);
Format.printf("\n| %-025b |", 30433);

// Binary formatting of short.
Format.printf("\n\n| %25b |", (short)45321);
Format.printf("\n| %-25b |", (short)45321);
Format.printf("\n| %025b |", (short)45321);
Format.printf("\n| %-025b |", (short)45321);

// Integer field formatting
Format.printf("\n\n| %-21s |", "Signed Formatting");
Format.printf("\n| %-21s | \n", "Before          After");

int [] x = {-12, 145, 1356, -31356, -10112};

for(int j=0;j<5;j++){
    int i=x[j];
    Format.printf("|%7d",i);                               // fixed min width of 7
    Format.printf(" --> %c",i<0?'-':'');                    // determine sign
    Format.printf("%-9d | \n",i<0?-i:i);                    // fixed min width of 9
}

for(int j=0;j<5;j++){
    int i=x[j];
    Format.printf("|%7d",i);
    Format.printf(" --> %c",i<0?'-':'');
    Format.printf("%-09d | \n",i<0?-i:i);                    // pad with zeros
}
}
}

```

Format Demo

FormatDemo.java is a detailed, step-by-step, fully comprehensive demonstration of the widget class is included with this application note.

Published Resources – for More Information

This class was developed to allow the Javelin to have access to C-like text formatting. For more information you may reference documentation for the various C functions with the same name as our methods.

Javelin Stamp Discussion Forum – Questions and Answers

The Parallax, Inc. Javelin Stamp Discussion Forum is a searchable repository of design questions and answers for the Javelin Stamp. To view the Javelin Stamp Forum, go to www.javelinstamp.com and follow the Discussion link. You can also join this forum and post your own questions. The Parallax technical staff, and Javelin Stamp users who monitor the list, will see your questions and reply with helpful tips, part numbers, pointers to useful web pages, etc.

Copyright © 2003 by Parallax, Inc. All rights reserved. Javelin, Stamp, and PBASIC are trademarks of Parallax, Inc., and BASIC Stamp is a registered trademark of Parallax, Inc. Windows is a registered trademark of Microsoft Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. Other brand and product names are trademarks or registered trademarks of their respective holders.

Parallax, Inc. is not responsible for special, incidental, or consequential damages resulting from any breach of warranty, or under any legal theory, including lost profits, downtime, goodwill, damage to or replacement of equipment or property, and any costs of recovering, reprogramming, or reproducing any data stored in or used with Parallax products.