
LwPRINTF

Tilen MAJERLE

Mar 23, 2024

CONTENTS

1	Features	3
2	Requirements	5
3	Contribute	7
4	License	9
5	Table of contents	11
5.1	Getting started	11
5.2	User manual	15
5.3	API reference	24
5.4	Test results	36
5.5	Examples and demos	60
5.6	Changelog	60
5.7	Authors	61
	Index	63

Welcome to the documentation for version latest-develop.

LwPRINTF is lightweight stdio manager optimized for embedded systems. It includes implementation of standard output functions such as `printf`, `vprintf`, `snprintf`, `sprintf` and `vsprintf` in an embedded-systems optimized way.

[Download library](#) [Getting started](#) [Open Github](#) [Donate](#)

FEATURES

- Written in C (C11), compatible with `size_t` and `uintmax_t` types for some specifiers
- Implements output functions compatible with `printf`, `vprintf`, `snprintf`, `sprintf` and `vsprintf`
- Low-memory footprint, suitable for embedded systems
- Reentrant access to all API functions
- Operating-system ready
- Requires single output function to be implemented by user for `printf`-like API calls
- With optional functions for operating systems to protect multiple threads printing to the same output stream
- Allows multiple output stream functions (unlike standard `printf` which supports only one) to separate parts of application
- Added additional specifiers vs original features
- User friendly MIT license

REQUIREMENTS

- C compiler
- Few kB of non-volatile memory

CONTRIBUTE

Fresh contributions are always welcome. Simple instructions to proceed:

1. Fork Github repository
2. Respect `C style & coding rules` used by the library
3. Create a pull request to `develop` branch with new features or bug fixes

Alternatively you may:

1. Report a bug
2. Ask for a feature request

LICENSE

MIT License

Copyright (c) 2024 Tilen MAJERLE

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "[Software](#)"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to [do](#) so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "[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 NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

TABLE OF CONTENTS

5.1 Getting started

Getting started may be the most challenging part of every new library. This guide is describing how to start with the library quickly and effectively

5.1.1 Download library

Library is primarily hosted on [Github](#).

You can get it by:

- Downloading latest release from [releases area](#) on Github
- Cloning `main` branch for latest stable version
- Cloning `develop` branch for latest development

Download from releases

All releases are available on Github [releases area](#).

Clone from Github

First-time clone

This is used when you do not have yet local copy on your machine.

- Make sure `git` is installed.
- Open console and navigate to path in the system to clone repository to. Use command `cd your_path`
- Clone repository with one of available options below
 - Run `git clone --recurse-submodules https://github.com/MaJerle/lwprintf` command to clone entire repository, including submodules
 - Run `git clone --recurse-submodules --branch develop https://github.com/MaJerle/lwprintf` to clone *development* branch, including submodules
 - Run `git clone --recurse-submodules --branch main https://github.com/MaJerle/lwprintf` to clone *latest stable* branch, including submodules
- Navigate to `examples` directory and run favourite example

Update cloned to latest version

- Open console and navigate to path in the system where your repository is located. Use command `cd your_path`
- Run `git pull origin main` command to get latest changes on main branch
- Run `git pull origin develop` command to get latest changes on develop branch
- Run `git submodule update --init --remote` to update submodules to latest version

Note: This is preferred option to use when you want to evaluate library and run prepared examples. Repository consists of multiple submodules which can be automatically downloaded when cloning and pulling changes from root repository.

5.1.2 Add library to project

At this point it is assumed that you have successfully download library, either cloned it or from releases page. Next step is to add the library to the project, by means of source files to compiler inputs and header files in search path.

CMake is the main supported build system. Package comes with the `CMakeLists.txt` and `library.cmake` files, both located in the `lwprintf` directory:

- `CMakeLists.txt`: Is a wrapper and only includes `library.cmake` file. It is used if target application uses `add_subdirectory` and then uses `target_link_libraries` to include the library in the project
- `library.cmake`: It is a fully configured set of variables. User must use `include(path/to/library.cmake)` to include the library and must manually add files/includes to the final target

Tip: Open `library.cmake` file and manually analyze all the possible variables you can set for full functionality.

If you do not use the *CMake*, you can do the following:

- Copy `lwprintf` folder to your project, it contains library files
- Add `lwprintf/src/include` folder to *include path* of your toolchain. This is where *C/C++* compiler can find the files during compilation process. Usually using `-I` flag
- Add source files from `lwprintf/src/` folder to toolchain build. These files are built by *C/C++* compiler. *CMake* configuration comes with the library, allows users to include library in the project as **subdirectory** and **library**.
- Copy `lwprintf/src/include/lwprintf/lwprintf_opts_template.h` to project folder and rename it to `lwprintf_opts.h`
- Build the project

5.1.3 Configuration file

Configuration file is used to overwrite default settings defined for the essential use case. Library comes with template config file, which can be modified according to the application needs. and it should be copied (or simply renamed in-place) and named `lwprintf_opts.h`

Note: Default configuration template file location: `lwprintf/src/include/lwprintf/lwprintf_opts_template.h`. File must be renamed to `lwprintf_opts.h` first and then copied to the project directory where compiler include paths have access to it by using `#include "lwprintf_opts.h"`.

Tip: If you are using *CMake* build system, define the variable `LWPRINTF_OPTS_FILE` before adding library's directory to the *CMake* project. Variable must contain the path to the user options file. If not provided and to avoid build error, one will be generated in the build directory.

Configuration options list is available available in the *Configuration* section. If any option is about to be modified, it should be done in configuration file

Listing 1: Template configuration file

```

1  /**
2   * \file      lwprintf_opts_template.h
3   * \brief    LwPRINTF configuration file
4   */
5
6  /**
7   * Copyright (c) 2024 Tilen MAJERLE
8   *
9   * Permission is hereby granted, free of charge, to any person
10  * obtaining a copy of this software and associated documentation
11  * files (the "Software"), to deal in the Software without restriction,
12  * including without limitation the rights to use, copy, modify, merge,
13  * publish, distribute, sublicense, and/or sell copies of the Software,
14  * and to permit persons to whom the Software is furnished to do so,
15  * subject to the following conditions:
16  *
17  * The above copyright notice and this permission notice shall be
18  * included in all copies or substantial portions of the Software.
19  *
20  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
21  * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
22  * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
23  * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
24  * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
25  * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
26  * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
27  * OTHER DEALINGS IN THE SOFTWARE.
28  *
29  * This file is part of LwPRINTF - Lightweight stdio manager library.
30  *
31  * Author:      Tilen MAJERLE <tilen@majerle.eu>
32  * Version:    v1.0.5

```

(continues on next page)

(continued from previous page)

```

33  */
34  #ifndef LWPRINTF_OPTS_HDR_H
35  #define LWPRINTF_OPTS_HDR_H
36
37  /* Rename this file to "lwprintf_opts.h" for your application */
38
39  /*
40   * Open "include/lwprintf/lwprintf_opt.h" and
41   * copy & replace here settings you want to change values
42   */
43
44  #endif /* LWPRINTF_OPTS_HDR_H */

```

Note: If you prefer to avoid using configuration file, application must define a global symbol `LWPRINTF_IGNORE_USER_OPTS`, visible across entire application. This can be achieved with `-D` compiler option.

5.1.4 Minimal example code

To verify proper library setup, minimal example has been prepared. Run it in your main application file to verify its proper execution

Listing 2: Absolute minimum example

```

1  #include "lwprintf/lwprintf.h"
2
3  /* Called for every character to be printed */
4  int
5  lwprintf_out(int ch, lwprintf_t* lwp) {
6      /* May use printf to output it for test */
7      if (ch != '\0') {
8          printf("%c", (char)ch);
9      }
10     return ch;
11 }
12
13 int
14 main(void) {
15     /* Initialize default lwprintf instance with output function */
16     lwprintf_init(lwprintf_out);
17
18     /* Print first text */
19     lwprintf_printf("Text: %d", 10);
20 }

```

5.2 User manual

5.2.1 How it works

LwPRINTF library supports 2 different formatting output types:

- Write formatted data to user input array
- Directly print formatted characters by calling `output_function` for every formatted character in the input string

Text formatting is based on input format string followed by the data parameters. It is mostly used to prepare numeric data types to human readable format.

Note: LwPRINTF is open-source implementation of regular *stdio.h* library in C language. It implements only output functions, excluding input scanning features

Formatting functions take input *format string* followed by (optional) different data types. Internal algorithm scans character by character to understand type of expected data user would like to have printed.

Every format specifier starts with letter %, followed by optional set of flags, widths and other sets of characters. Last part of every specifier is its type, that being type of format and data to display.

Tip: To print number 1234 in human readable format, use specifier %d. With default configuration, call `lwprintf_printf("%d", 1234);` and it will print "1234".

Check section *Format specifier* for list of all formats and data types

Character output function

API functions printing characters directly to the output stream (ex. `lwprintf_printf`), require output function to be set during initialization procedure.

Output function is called by the API for every character to be printed/transmitted by the application.

Note:

Output function is set during initialization procedure.

If not set (set as NULL), it is not possible to use API function which directly print characters to output stream. Application is then limited only to API functions that write formatted data to input buffer.

Notes to consider:

- Output function must return same character as it was used as an input parameter to consider successful print
- Output function will receive (int) '\0' character to indicate no more characters will follow in this API call
- Single output function may be used for different LwPRINTF instances

Listing 3: Absolute minimum example to support direct output

```

1 #include "lwprintf/lwprintf.h"
2
3 /* Called for every character to be printed */

```

(continues on next page)

(continued from previous page)

```

4  int
5  lwprintf_out(int ch, lwprintf_t* lwp) {
6      /* May use printf to output it for test */
7      if (ch != '\0') {
8          printf("%c", (char)ch);
9      }
10     return ch;
11 }
12
13 int
14 main(void) {
15     /* Initialize default lwprintf instance with output function */
16     lwprintf_init(lwprintf_out);
17
18     /* Print first text */
19     lwprintf_printf("Text: %d", 10);
20 }

```

5.2.2 Format specifier

Syntax

Full syntax for format specifier is %[flags][width][.precision][length]type

Flags

Flags field may have zero or more characters, and in any order. List of supported flags:

Character	Description
minus -	Left-align the output of this placeholder. The default is to right-align the output
plus +	Prepends a plus for positive signed-numeric types. positive = +, negative = -
space space ch	Prepends a space for positive signed-numeric types. positive = space ch, negative = -. This flag is ignored if the + flag exists
zero 0	When the <i>width</i> option is specified, prepends zeros for numeric types. The default prepends spaces, if this flag is not set
apostrophe ,	The integer or exponent of a decimal has the thousands grouping separator applied.
has #	Alternate form: For g and G types, trailing zeros are not removed. For f, F, e, E, g, G types, the output always contains a decimal point. For o, x, X types, the text 0, 0x, 0X, respectively, is prepended to non-zero numbers.

Width

Width field specifies a *minimum* number of characters to output, and is typically used to pad fixed-width fields in tabulated output, where fields would otherwise be smaller. Please keep in mind that this parameter does not truncate output is input is longer than *width* field value.

Concerning *width* field, you may:

- Ignore it completely, output does not rely on *width* field by any means
- Write a fixed value as part of format specifier. Number must be an integer value
- Use asterisk `*` char and pass number as part of parameter. `printf("%3d", 6)` or `printf("%*d", 3, 6)` will generate the same output.

Tip: When fixed value is used to set width field, leading zero is not counted as part of *width* field, but as flag instead, indicating prepend number with leading zeros

Precision

Precision field usually specifies a maximum limit on the output, depending on the particular formatting type. For floating point numeric types, it specifies the number of digits to the right of the decimal point that the output should be rounded. For the string type, it limits the number of characters that should be output, after which the string is truncated.

Precision field may be omitted, or a numeric integer value, or a dynamic value when passed as another argument when indicated by an asterisk `*`. For example, `printf("%.3s", 3, "abcdef")` will result in `abc` being printed.

Length

Length field may be ignored or one of the below:

Character	Description
hh	For integer types, causes <code>printf</code> to expect an <code>int</code> -sized integer argument which was promoted from a <code>char</code>
h	For integer types, causes <code>printf</code> to expect an <code>int</code> -sized integer argument which was promoted from a <code>short</code>
l	For integer types, causes <code>printf</code> to expect a <code>long</code> -sized integer argument. For floating point types, this has no effect
ll	For integer types, causes <code>printf</code> to expect a <code>long long</code> -sized integer argument
L	For floating point types, causes <code>printf</code> to expect a <code>long double</code> argument
z	For integer types, causes <code>printf</code> to expect a <code>size_t</code> -sized integer argument
j	For integer types, causes <code>printf</code> to expect a <code>intmax_t</code> -sized integer argument
t	For integer types, causes <code>printf</code> to expect a <code>ptrdiff_t</code> -sized integer argument

Specifier types

This is a list of standard specifiers for outputting the data to the stream. Column *Supported* gives an overview which specifiers are actually supported by the library.

Spec-ifier	Supported	Description
%	Yes	Prints literal % character
d i	Yes	Prints <code>signed int</code> . No difference between either of them
u	Yes	Prints <code>unsigned int</code>
f F	Yes	Prints <code>double</code> in normal fixed-point notation. <code>f</code> and <code>F</code> only differs in how the strings for an infinite number or NaN are printed (<code>inf</code> , <code>infinity</code> and <code>nan</code> for <code>f</code> ; <code>INF</code> , <code>INFINITY</code> and <code>NAN</code> for <code>F</code>).
e E	Yes	Prints <code>double</code> in standard form <code>[-]d.ddd e[+-]ddd</code> . <code>e</code> uses lower-case and <code>E</code> uses upper-case letter for exponent annotation.
g G	Yes	Prints <code>double</code> in either normal or exponential notation, whichever is more appropriate for its magnitude. <code>g</code> uses lower-case letters, <code>G</code> uses upper-case letters. This type differs slightly from fixed-point notation in that insignificant zeroes to the right of the decimal point are not included. Also, the decimal point is not included on whole numbers.
x X	Yes	Prints <code>unsigned int</code> as a hexadecimal number. <code>x</code> uses lower-case and <code>X</code> uses upper-case letters
o	Yes	Prints <code>unsigned int</code> in octal format
s	Yes	Prints null terminated string
c	Yes	Prints <code>char</code> type
p	Yes	Prints <code>void *</code> in an hex-based format. Reads input as <code>unsigned int</code> by default.
a A	Not yet	Prints <code>double</code> in hexadecimal notation. Currently it will print NaN when used
n	Yes	Prints nothing but writes the number of characters successfully written so far into an integer pointer parameter

Notes about float types

It is important to understand how library works under the hood to understand limitations on floating-point numbers. When it comes to level of precision, maximum number of digits is linked to support `long` or `long long` integer types.

Note: When `long long` type is supported by the compiler (usually part of C99 or later), maximum number of valid digits is 18, or 9 digits if system supports only `long` data types.

If application tries to use more precision digits than maximum, remaining digits are automatically printed as all 0. As a consequence, output using LwPRINTF library may be different in comparison to other `printf` implementations.

Tip: Float data type supports up to 7 and double up to 15.

Additional specifier types

LwPRINTF implementation supports some specifiers that are usually not available in standard implementation. Those are more targeting embedded systems although they may be used in any general-purpose application

Spec-ifier	Description
B b	Prints unsigned int data as binary representation.
K k	Prints unsigned char based data array as sequence of hex numbers. Use <i>width</i> field to specify length of input array. Use K for upper-case hex letters, k for lower-case.

Listing 4: Additional format specifiers

```

1  /* List of specifiers added in the library which are not available in standard printf_
   ↪ implementation */
2
3  #include "lwprintf/lwprintf.h"
4
5  /**
6   * \brief      List of additional specifiers to print
7   */
8  void
9  additional_format_specifiers(void) {
10     unsigned char my_array[] = { 0x01, 0x02, 0xA4, 0xB5, 0xC6 };
11
12     /* Binary output */
13
14     /* Prints number 8 in binary format, so "1000" */
15     lwprintf_printf("%b\r\n", 8U);
16     /* Prints number 16 in binary format with 10 places, so "      10000" */
17     lwprintf_printf("%10b\r\n", 16U);
18     /* Prints number 16 in binary format with 10 places, leading zeros, so "0000010000"
   ↪ */
19     lwprintf_printf("%010b\r\n", 16U);
20
21     /* Array outputs */
22
23     /* Fixed length with uppercase hex numbers, outputs "0102A4B5C6" */
24     lwprintf_printf("%5K\r\n", my_array);
25     /* Fixed length with lowercase hex numbers, outputs "0102a4b5c6" */
26     lwprintf_printf("%5k\r\n", my_array);
27     /* Variable length with uppercase letters, outputs "0102A4B5C6" */
28     lwprintf_printf("%*K\r\n", (int)LWPRINTF_ARRAYSIZE(my_array), my_array);
29     /* Variable length with lowercase letters, outputs "0102a4b5c6" */
30     lwprintf_printf("%*k\r\n", (int)LWPRINTF_ARRAYSIZE(my_array), my_array);
31     /* Variable length with uppercase letters and spaces, outputs "01 02 A4 B5 C6" */
32     lwprintf_printf("% *K\r\n", (int)LWPRINTF_ARRAYSIZE(my_array), my_array);
33     /* Variable length with uppercase letters and spaces, outputs "01 02 a4 b5 c6" */
34     lwprintf_printf("% *k\r\n", (int)LWPRINTF_ARRAYSIZE(my_array), my_array);
35 }

```

5.2.3 LwPRINTF instances

LwPRINTF is very flexible and allows multiple instances for output print functions.

Note: Multiple instances with LwPRINTF are useful only with direct print functions, such as `lwprintf_printf`. If application uses only format functions which write to input buffer, it may always use default LwPRINTF instance which is created by the library itself

Use of different instances is useful if application needs different print configurations. Each instance has its own `print_output` function, allowing application to use multiple debug configurations (as an example)

Tip: Use functions with `_ex` suffix to directly work with custom instances. Functions without `_ex` suffix use default LwPRINTF instance

Listing 5: Custom LwPRINTF instance for output

```

1  #include "lwprintf/lwprintf.h"
2
3  /* Define application custom instance */
4  lwprintf_t custom_instance;
5
6  /* Define custom output function for print */
7  int
8  custom_out(int ch, lwprintf_t* p) {
9      /* Do whatever with this character */
10     if (ch == '\0') {
11         /* This is end of string in current formatting */
12         /* Maybe time to start DMA transfer? */
13     } else {
14         /* Print or send character */
15     }
16
17     /* Return character to proceed */
18     return ch;
19 }
20
21 /* Define output function for default instance */
22 int
23 default_out(int ch, lwprintf_t* p) {
24     /* Print function for default instance */
25
26     /* See custom_out function for implementation details */
27 }
28
29 int
30 main(void) {
31     /* Initialize default lwprintf instance with output function */
32     lwprintf_init(default_out);
33     /* Initialize custom lwprintf instance with output function */
34     lwprintf_init_ex(&custom_instance, custom_out);
35

```

(continues on next page)

(continued from previous page)

```

36  /* Print first text over default output */
37  lwprintf_printf("Text: %d", 10);
38  /* Print text over custom instance */
39  lwprintf_printf_ex(&custom_instance, "Custom: %f", 3.2f);
40  }

```

Note: It is perfectly valid to use single output function for all application instances. Use check against input parameter for `lwprintf_t` if it matches your custom LwPRINTF instance memory address

Listing 6: Single output function for all LwPRINTF instances

```

1  #include "lwprintf/lwprintf.h"
2
3  /* Define application custom instance */
4  lwprintf_t custom_instance1;
5  lwprintf_t custom_instance2;
6
7  /* Define custom output function for print */
8  int
9  my_out(int ch, lwprintf_t* p) {
10     if (p == &custom_instance1) {
11         /* This is custom instance 1 */
12     } else if (p == &custom_instance2) {
13         /* This is custom instance 2 */
14     } else {
15         /* This is default instance */
16     }
17     return ch;
18 }
19
20 int
21 main(void) {
22     /* Initialize default lwprintf instance with output function */
23     lwprintf_init(my_out);
24     lwprintf_init_ex(&custom_instance1, my_out);
25     lwprintf_init_ex(&custom_instance2, my_out);
26
27     /* Use print functions ... */
28 }

```

5.2.4 Thread safety

LwPRINTF uses re-entrant functions, especially the one that format string to user application buffer. It is fully allowed to access to the same LwPRINTF instance from multiple operating-system threads.

However, when it comes to direct print functions, such as `lwprintf_printf_ex()` (or any other similar), calling those functions from multiple threads may introduce mixed output stream of data.

This is due to the fact that direct printing functions use same output function to print single character. When called from multiple threads, one thread may preempt another, causing strange output string.

Listing 7: Multiple threads printing at the same time without thread-safety enabled

```
1  #include "lwprintf/lwprintf.h"
2
3  /* Assuming LwPRINTF has been initialized before */
4
5  void
6  task_1(void* arg) {
7      lwprintf_printf("Hello world\r\n");
8  }
9
10 void
11 task_2(void* arg) {
12     lwprintf_printf("This is Task 2\r\n");
13 }
14
15 /*
16  * If thread safety is not enabled,
17  * running above example may print:
18  *
19  * "Hello This is Task 2\r\nworld\r\n"
20  */
```

LwPRINTF therefore comes with a solution that introduces mutexes to lock print functions when in use from within single thread context.

Note: If application does not have any issues concerning mixed output, it is safe to disable OS support in OS environment. This will not have any negative effect on performance or memory corruption.

Tip: To enable thread-safety support, parameter `LWPRINTF_CFG_OS` must be set to 1. Please check [Configuration](#) for more information about other options.

After thread-safety features has been enabled, it is necessary to implement 4 low-level system functions.

Tip: System function template example is available in `lwprintf/src/system/` folder.

Example code for CMSIS-OS V2

Note: Check [System functions](#) section for function description

Listing 8: System function implementation for CMSIS-OS based operating systems

```
1  /**
2   * \file      lwprintf_sys_cmsis_os.c
3   * \brief     System functions for CMSIS-OS based operating system
4   */
```

(continues on next page)

(continued from previous page)

```

5  /*
6  *
7  * Copyright (c) 2024 Tilen MAJERLE
8  *
9  * Permission is hereby granted, free of charge, to any person
10 * obtaining a copy of this software and associated documentation
11 * files (the "Software"), to deal in the Software without restriction,
12 * including without limitation the rights to use, copy, modify, merge,
13 * publish, distribute, sublicense, and/or sell copies of the Software,
14 * and to permit persons to whom the Software is furnished to do so,
15 * subject to the following conditions:
16 *
17 * The above copyright notice and this permission notice shall be
18 * included in all copies or substantial portions of the Software.
19 *
20 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
21 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
22 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
23 * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
24 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
25 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
26 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
27 * OTHER DEALINGS IN THE SOFTWARE.
28 *
29 * This file is part of LwPRINTF - Lightweight stdio manager library.
30 *
31 * Author:          Tilen MAJERLE <tilen@majerle.eu>
32 * Version:         v1.0.5
33 */
34 #include "system/lwprintf_sys.h"
35
36 #if LWPRINTF_CFG_OS && !__DOXYGEN__
37
38 #include "cmsis_os.h"
39
40 uint8_t
41 lwprintf_sys_mutex_create(LWPRINTF_CFG_OS_MUTEX_HANDLE* m) {
42     const osMutexAttr_t attr = {
43         .name = "lwprintf_mutex",
44         .attr_bits = osMutexRecursive,
45     };
46     return (*m = osMutexNew(&attr)) != NULL;
47 }
48
49 uint8_t
50 lwprintf_sys_mutex_isvalid(LWPRINTF_CFG_OS_MUTEX_HANDLE* m) {
51     return *m != NULL;
52 }
53
54 uint8_t
55 lwprintf_sys_mutex_wait(LWPRINTF_CFG_OS_MUTEX_HANDLE* m) {
56     return osMutexAcquire(*m, osWaitForever) == osOK;

```

(continues on next page)

```

57 }
58
59 uint8_t
60 lwprintf_sys_mutex_release(LWPRINTF_CFG_OS_MUTEX_HANDLE* m) {
61     return osMutexRelease(*m) == osOK;
62 }
63
64 #endif /* LWPRINTF_CFG_OS && !__DOXYGEN__ */

```

5.3 API reference

List of all the modules:

5.3.1 LwPRINTF

group **LWPRINTF**

Lightweight stdio manager.

Defines

LWPRINTF_UNUSED(x)

Unused variable macro.

Parameters

- **x** – [**in**] Unused variable

LWPRINTF_ARRAYSIZE(x)

Calculate size of statically allocated array.

Parameters

- **x** – [**in**] Input array

Returns

Number of array elements

lwprintf_sprintf_ex(lwobj, s, format, ...)

Write formatted data from variable argument list to sized buffer.

Parameters

- **lwobj** – [**inout**] LwPRINTF instance. Set to NULL to use default instance
- **s** – [**in**] Pointer to a buffer where the resulting C-string is stored. The buffer should have a size of at least n characters
- **format** – [**in**] C string that contains a format string that follows the same specifications as format in printf
- ... – [**in**] Optional arguments for format string

Returns

The number of characters that would have been written, not counting the terminating null character.

lwprintf_init(out_fn)

Initialize default LwPRINTF instance.

See also:

[*lwprintf_init_ex*](#)

Parameters

- **out_fn** – [in] Output function used for print operation

Returns

1 on success, 0 otherwise

lwprintf_vprintf(format, arg)

Print formatted data from variable argument list to the output with default LwPRINTF instance.

Parameters

- **format** – [in] C string that contains the text to be written to output
- **arg** – [in] A value identifying a variable arguments list initialized with `va_start`. `va_list` is a special type defined in `<stdarg.h>`.

Returns

The number of characters that would have been written if `n` had been sufficiently large, not counting the terminating null character.

lwprintf_printf(format, ...)

Print formatted data to the output with default LwPRINTF instance.

Parameters

- **format** – [in] C string that contains the text to be written to output
- **...** – [in] Optional arguments for format string

Returns

The number of characters that would have been written if `n` had been sufficiently large, not counting the terminating null character.

lwprintf_vsnprintf(s, n, format, arg)

Write formatted data from variable argument list to sized buffer with default LwPRINTF instance.

Parameters

- **s** – [in] Pointer to a buffer where the resulting C-string is stored. The buffer should have a size of at least `n` characters
- **n** – [in] Maximum number of bytes to be used in the buffer. The generated string has a length of at most `n - 1`, leaving space for the additional terminating null character
- **format** – [in] C string that contains a format string that follows the same specifications as `format` in `printf`
- **arg** – [in] A value identifying a variable arguments list initialized with `va_start`. `va_list` is a special type defined in `<stdarg.h>`.

Returns

The number of characters that would have been written if **n** had been sufficiently large, not counting the terminating null character.

lwprintf_snprintf(s, n, format, ...)

Write formatted data from variable argument list to sized buffer with default LwPRINTF instance.

Parameters

- **s** – [in] Pointer to a buffer where the resulting C-string is stored. The buffer should have a size of at least **n** characters
- **n** – [in] Maximum number of bytes to be used in the buffer. The generated string has a length of at most **n** - 1, leaving space for the additional terminating null character
- **format** – [in] C string that contains a format string that follows the same specifications as format in printf
- ... – [in] Optional arguments for format string

Returns

The number of characters that would have been written if **n** had been sufficiently large, not counting the terminating null character.

lwprintf_sprintf(s, format, ...)

Write formatted data from variable argument list to sized buffer with default LwPRINTF instance.

Parameters

- **s** – [in] Pointer to a buffer where the resulting C-string is stored. The buffer should have a size of at least **n** characters
- **format** – [in] C string that contains a format string that follows the same specifications as format in printf
- ... – [in] Optional arguments for format string

Returns

The number of characters that would have been written, not counting the terminating null character.

lwprintf_protect()

Manually enable mutual exclusion.

Returns

1 if protected, 0 otherwise

lwprintf_unprotect()

Manually disable mutual exclusion.

Returns

1 if protected, 0 otherwise

lwprintf

Print formatted data to the output with default LwPRINTF instance.

Note: This function is equivalent to *lwprintf_printf* and available only if *LW-PRINTF_CFG_ENABLE_SHORTNAMES* is enabled

Parameters

- **format** – [in] C string that contains the text to be written to output
- ... – [in] Optional arguments for format string

Returns

The number of characters that would have been written if **n** had been sufficiently large, not counting the terminating null character.

lwvprintf

Print formatted data from variable argument list to the output with default LwPRINTF instance.

Note: This function is equivalent to *lwprintf_vprintf* and available only if *LWPRINTF_CFG_ENABLE_SHORTNAMES* is enabled

Parameters

- **format** – [in] C string that contains the text to be written to output
- **arg** – [in] A value identifying a variable arguments list initialized with *va_start*. *va_list* is a special type defined in *<stdarg.h>*.

Returns

The number of characters that would have been written if **n** had been sufficiently large, not counting the terminating null character.

lwvsprintf

Write formatted data from variable argument list to sized buffer with default LwPRINTF instance.

Note: This function is equivalent to *lwprintf_vsnprintf* and available only if *LWPRINTF_CFG_ENABLE_SHORTNAMES* is enabled

Parameters

- **s** – [in] Pointer to a buffer where the resulting C-string is stored. The buffer should have a size of at least **n** characters
- **n** – [in] Maximum number of bytes to be used in the buffer. The generated string has a length of at most **n** - 1, leaving space for the additional terminating null character
- **format** – [in] C string that contains a format string that follows the same specifications as format in *printf*
- **arg** – [in] A value identifying a variable arguments list initialized with *va_start*. *va_list* is a special type defined in *<stdarg.h>*.

Returns

The number of characters that would have been written if **n** had been sufficiently large, not counting the terminating null character.

lwsprintf

Write formatted data from variable argument list to sized buffer with default LwPRINTF instance.

Note: This function is equivalent to *lwprintf_snprintf* and available only if *LWPRINTF_CFG_ENABLE_SHORTNAMES* is enabled

Parameters

- **s** – [in] Pointer to a buffer where the resulting C-string is stored. The buffer should have a size of at least **n** characters
- **n** – [in] Maximum number of bytes to be used in the buffer. The generated string has a length of at most **n** - 1, leaving space for the additional terminating null character
- **format** – [in] C string that contains a format string that follows the same specifications as format in printf
- ... – [in] Optional arguments for format string

Returns

The number of characters that would have been written if **n** had been sufficiently large, not counting the terminating null character.

lwsprintf

Write formatted data from variable argument list to sized buffer with default LwPRINTF instance.

Note: This function is equivalent to *lwprintf_sprintf* and available only if *LWPRINTF_CFG_ENABLE_SHORTNAMES* is enabled

Parameters

- **s** – [in] Pointer to a buffer where the resulting C-string is stored. The buffer should have a size of at least **n** characters
- **format** – [in] C string that contains a format string that follows the same specifications as format in printf
- ... – [in] Optional arguments for format string

Returns

The number of characters that would have been written, not counting the terminating null character.

printf

Print formatted data to the output with default LwPRINTF instance.

Note: This function is equivalent to *lwprintf_printf* and available only if *LWPRINTF_CFG_ENABLE_STD_NAMES* is enabled

Parameters

- **format** – [in] C string that contains the text to be written to output
- ... – [in] Optional arguments for format string

Returns

The number of characters that would have been written if `n` had been sufficiently large, not counting the terminating null character.

vprintf

Print formatted data from variable argument list to the output with default LwPRINTF instance.

Note: This function is equivalent to *lwprintf_vprintf* and available only if *LWPRINTF_CFG_ENABLE_STD_NAMES* is enabled

Parameters

- **format** – [in] C string that contains the text to be written to output
- **arg** – [in] A value identifying a variable arguments list initialized with `va_start`. `va_list` is a special type defined in `<stdarg.h>`.

Returns

The number of characters that would have been written if `n` had been sufficiently large, not counting the terminating null character.

vsprintf

Write formatted data from variable argument list to sized buffer with default LwPRINTF instance.

Note: This function is equivalent to *lwprintf_vsprintf* and available only if *LWPRINTF_CFG_ENABLE_STD_NAMES* is enabled

Parameters

- **s** – [in] Pointer to a buffer where the resulting C-string is stored. The buffer should have a size of at least `n` characters
- **n** – [in] Maximum number of bytes to be used in the buffer. The generated string has a length of at most `n - 1`, leaving space for the additional terminating null character
- **format** – [in] C string that contains a format string that follows the same specifications as format in `printf`
- **arg** – [in] A value identifying a variable arguments list initialized with `va_start`. `va_list` is a special type defined in `<stdarg.h>`.

Returns

The number of characters that would have been written if `n` had been sufficiently large, not counting the terminating null character.

snprintf

Write formatted data from variable argument list to sized buffer with default LwPRINTF instance.

Note: This function is equivalent to *lwprintf_snprintf* and available only if *LWPRINTF_CFG_ENABLE_STD_NAMES* is enabled

Parameters

- **s** – [in] Pointer to a buffer where the resulting C-string is stored. The buffer should have a size of at least **n** characters
- **n** – [in] Maximum number of bytes to be used in the buffer. The generated string has a length of at most **n** - 1, leaving space for the additional terminating null character
- **format** – [in] C string that contains a format string that follows the same specifications as format in printf
- ... – [in] Optional arguments for format string

Returns

The number of characters that would have been written if **n** had been sufficiently large, not counting the terminating null character.

sprintf

Write formatted data from variable argument list to sized buffer with default LwPRINTF instance.

Note: This function is equivalent to *lwprintf_sprintf* and available only if *LW-PRINTF_CFG_ENABLE_STD_NAMES* is enabled

Parameters

- **s** – [in] Pointer to a buffer where the resulting C-string is stored. The buffer should have a size of at least **n** characters
- **format** – [in] C string that contains a format string that follows the same specifications as format in printf
- ... – [in] Optional arguments for format string

Returns

The number of characters that would have been written, not counting the terminating null character.

lwprintf_debug(fmt, ...)

Debug output function.

Its purpose is to have a debug printout to the defined output, which will get disabled **for** the release build (when NDEBUG is #defined).

Note: It calls *lwprintf_printf* to execute the print

Note: Defined as empty when NDEBUG is enabled

Parameters

- **fmt** – [in] Format text

- ... – [in] Optional formatting parameters

lwprintf_debug_cond(cond, fmt, ...)

Conditional debug output.

It prints the formatted text only **if** condition is **true**

Its purpose is to have a debug printout to the defined output, which will get disabled **for** the release build (when NDEBUG is **defined**).

Note: It calls *lwprintf_debug* to execute the print

Note: Defined as empty when NDEBUG is enabled

Parameters

- **cond** – [in] Condition to check before outputting the message
- **fmt** – [in] Format text
- ... – [in] Optional formatting parameters

Typedefs

typedef int (***lwprintf_output_fn**)(int ch, struct lwprintf *lwobj)

Callback function for character output.

Param ch

[in] Character to print

Param lwobj

[in] LwPRINTF instance

Return

ch on success, 0 to terminate further string processing

Functions

uint8_t **lwprintf_init_ex**(*lwprintf_t* *lwobj, *lwprintf_output_fn* out_fn)

Initialize LwPRINTF instance.

Parameters

- **lwobj** – [inout] LwPRINTF working instance
- **out_fn** – [in] Output function used for print operation. When set to NULL, direct print to stream functions won't work and will return error if called by the application. Also, system mutex for this specific instance won't be called as system mutex isn't needed. All formatting functions (with print being an exception) are thread safe. Library utilizes stack-based variables

Returns

1 on success, 0 otherwise

int **lwprintf_vprintf_ex**(*lwprintf_t* *const lwobj, const char *format, va_list arg)

Print formatted data from variable argument list to the output.

Parameters

- **lwobj** – [inout] LwPRINTF instance. Set to NULL to use default instance
- **format** – [in] C string that contains the text to be written to output
- **arg** – [in] A value identifying a variable arguments list initialized with *va_start*. *va_list* is a special type defined in <stdarg>.

Returns

The number of characters that would have been written if *n* had been sufficiently large, not counting the terminating null character.

int **lwprintf_printf_ex**(*lwprintf_t* *const lwobj, const char *format, ...)

Print formatted data to the output.

Parameters

- **lwobj** – [inout] LwPRINTF instance. Set to NULL to use default instance
- **format** – [in] C string that contains the text to be written to output
- ... – [in] Optional arguments for format string

Returns

The number of characters that would have been written if *n* had been sufficiently large, not counting the terminating null character.

int **lwprintf_vsnprintf_ex**(*lwprintf_t* *const lwobj, char *s, size_t n, const char *format, va_list arg)

Write formatted data from variable argument list to sized buffer.

Parameters

- **lwobj** – [inout] LwPRINTF instance. Set to NULL to use default instance
- **s_out** – [in] Pointer to a buffer where the resulting C-string is stored. The buffer should have a size of at least *n* characters
- **n_maxlen** – [in] Maximum number of bytes to be used in the buffer. The generated string has a length of at most *n* - 1, leaving space for the additional terminating null character
- **format** – [in] C string that contains a format string that follows the same specifications as format in printf
- **arg** – [in] A value identifying a variable arguments list initialized with *va_start*. *va_list* is a special type defined in <stdarg>.

Returns

The number of characters that would have been written if *n* had been sufficiently large, not counting the terminating null character.

int **lwprintf_snprintf_ex**(*lwprintf_t* *const lwobj, char *s, size_t n, const char *format, ...)

Write formatted data from variable argument list to sized buffer.

Parameters

- **lwobj** – [inout] LwPRINTF instance. Set to NULL to use default instance

- **s_out** – [in] Pointer to a buffer where the resulting C-string is stored. The buffer should have a size of at least **n** characters
- **n_maxlen** – [in] Maximum number of bytes to be used in the buffer. The generated string has a length of at most **n** - 1, leaving space for the additional terminating null character
- **format** – [in] C string that contains a format string that follows the same specifications as format in printf
- ... – [in] Optional arguments for format string

Returns

The number of characters that would have been written if **n** had been sufficiently large, not counting the terminating null character.

uint8_t **lwprintf_protect_ex**(*lwprintf_t* *const lwobj)

Manually enable mutual exclusion.

Parameters

lwobj – [inout] LwPRINTF instance. Set to NULL to use default instance

Returns

1 if protected, 0 otherwise

uint8_t **lwprintf_unprotect_ex**(*lwprintf_t* *const lwobj)

Manually disable mutual exclusion.

Parameters

lwobj – [inout] LwPRINTF instance. Set to NULL to use default instance

Returns

1 if protection disabled, 0 otherwise

struct **lwprintf_t**

#include <lwprintf.h> LwPRINTF instance.

Public Members

lwprintf_output_fn **out_fn**

Output function for direct print operations

LWPRINTF_CFG_OS_MUTEX_HANDLE **mutex**

OS mutex handle

5.3.2 Configuration

This is the default configuration of the middleware. When any of the settings shall be modified, it shall be done in dedicated application config `lwprintf_opts.h` file.

Note: Check *Getting started* to create configuration file.

group **LWPRINTF_OPT**

LwPRINTF options.

Defines

LWPRINTF_CFG_OS

Enables 1 or disables 0 operating system support in the library.

Note: When LWPRINTF_CFG_OS is enabled, user must implement functions in *System functions* group.

LWPRINTF_CFG_OS_MUTEX_HANDLE

Mutex handle type.

Note: This value must be set in case *LWPRINTF_CFG_OS* is set to 1. If data type is not known to compiler, include header file with definition before you define handle type

LWPRINTF_CFG_OS_MANUAL_PROTECT

Enables 1 or disables 0 manual mutex lock.

When this feature is enabled, together with *LWPRINTF_CFG_OS*, behavior is as following:

- System mutex is kept created during init phase
- Calls to direct printing functions are not thread-safe by default anymore
- Calls to sprintf (buffer functions) are kept thread-safe
- User must manually call *lwprintf_protect* or *lwprintf_protect_ex* functions to protect direct printing operation
- User must manually call *lwprintf_unprotect* or *lwprintf_unprotect_ex* functions to exit protected area

Note: If you prefer to completely disable locking mechanism with this library, turn off *LWPRINTF_CFG_OS* and fully manually handle mutual exclusion for non-reentrant functions

LWPRINTF_CFG_SUPPORT_LONG_LONG

Enables 1 or disables 0 support for long long int type, signed or unsigned.

LWPRINTF_CFG_SUPPORT_TYPE_INT

Enables 1 or disables 0 support for any specifier accepting any kind of integer types. This is enabling d, b, u, o, i, x specifiers.

LWPRINTF_CFG_SUPPORT_TYPE_POINTER

Enables 1 or disables 0 support p pointer print type.

When enabled, architecture must support uintptr_t type, normally available with C11 standard

LWPRINTF_CFG_SUPPORT_TYPE_FLOAT

Enables 1 or disables 0 support f float type.

LWPRINTF_CFG_SUPPORT_TYPE_ENGINEERING

Enables 1 or disables 0 support for e engineering output type for float numbers.

Note: *LWPRINTF_CFG_SUPPORT_TYPE_FLOAT* has to be enabled to use this feature

LWPRINTF_CFG_SUPPORT_TYPE_STRING

Enables 1 or disables 0 support for s for string output.

LWPRINTF_CFG_SUPPORT_TYPE_BYTE_ARRAY

Enables 1 or disables 0 support for k for hex byte array output.

LWPRINTF_CFG_FLOAT_DEFAULT_PRECISION

Specifies default number of precision for floating number.

Represents number of digits to be used after comma if no precision is set with specifier itself

LWPRINTF_CFG_ENABLE_SHORTNAMES

Enables 1 or disables 0 optional short names for LwPRINTF API functions.

It adds functions for default instance: `lwprintf`, `lwsnprintf` and others

LWPRINTF_CFG_ENABLE_STD_NAMES

Enables 1 or disables 0 C standard API names.

Disabled by default not to interfere with compiler implementation. Application may need to remove standard C STDIO library from linkage to be able to properly compile LwPRINTF with this option enabled

5.3.3 System functions

System function are used in conjunction with thread safety. Please check *Thread safety* section for more information

group **LWPRINTF_SYS**

System functions when used with operating system.

Functions

`uint8_t lwprintf_sys_mutex_create(LWPRINTF_CFG_OS_MUTEX_HANDLE *m)`

Create a new mutex and assign value to handle.

Parameters

m – [out] Output variable to save mutex handle

Returns

1 on success, 0 otherwise

`uint8_t lwprintf_sys_mutex_isvalid(LWPRINTF_CFG_OS_MUTEX_HANDLE *m)`

Check if mutex handle is valid.

Parameters

m – [in] Mutex handle to check if valid

Returns

1 on success, 0 otherwise

uint8_t **lwprintf_sys_mutex_wait**(LWPRINTF_CFG_OS_MUTEX_HANDLE *m)

Wait for a mutex until ready (unlimited time)

Parameters**m** – [in] Mutex handle to wait for**Returns**

1 on success, 0 otherwise

uint8_t **lwprintf_sys_mutex_release**(LWPRINTF_CFG_OS_MUTEX_HANDLE *m)

Release already locked mutex.

Parameters**m** – [in] Mutex handle to release**Returns**

1 on success, 0 otherwise

5.4 Test results

Library is put under several tests to ensure correct output format. Results are underneath with information about number of passed and failed tests.

Note: Majority of failed tests are linked to precision digits with floating-point based specifiers. This is considered as *OK* since failures are visible at higher number of precision digits, not affecting final results. Keep in mind that effective number of precision digits with `float` type is 7 and for `double` is 15.

With the exception to additional specifiers, supported only by *LwPRINTF* library, all tests are compared against `stdio printf` library included in *Microsoft Visual Studio C/C++ compiler*.

Listing 9: Test results of the library

```
1  -----
2  Number of tests run: 149
3  Number of tests passed: 141
4  Number of tests failed: 8
5  Coverage: 94.630875 %
6  -----
7
8  Negative tests
9
10 -----
11 Format: "Precision: %3d, %20.*g"
12 Params: "i, i, 432432423.342321321"
13 Result VSprintf: "Precision: 19, 432432423.3423213363"
14 Length VSprintf: 36
15 Result LwPRINTF: "Precision: 19, 432432423.342321336"
16 Length LwPRINTF: 36
17 Test result: Fail
18 -----
```

(continues on next page)

(continued from previous page)

```

71 Result LwPRINTF: "-1.2345599999999998720000000000000000e-01"
72 Length LwPRINTF: 40
73 Test result: Fail
74 -----
75
76 Positive tests
77
78 ----
79 Format: "Precision: %3d, %.*g"
80 Params: "17, 17, 0.0001234567"
81 Result VSprintf: "Precision: 17, 0.0001234567"
82 Length VSprintf: 28
83 Result LwPRINTF: "Precision: 17, 0.0001234567"
84 Length LwPRINTF: 28
85 Test result: Pass
86 ----
87 Format: "Precision: %3d, %20.*g"
88 Params: "i, i, 432432423.342321321"
89 Result VSprintf: "Precision: 0, 4e+08"
90 Length VSprintf: 36
91 Result LwPRINTF: "Precision: 0, 4e+08"
92 Length LwPRINTF: 36
93 Test result: Pass
94 ----
95 Format: "Precision: %3d, %20.*g"
96 Params: "i, i, 432432423.342321321"
97 Result VSprintf: "Precision: 1, 4e+08"
98 Length VSprintf: 36
99 Result LwPRINTF: "Precision: 1, 4e+08"
100 Length LwPRINTF: 36
101 Test result: Pass
102 ----
103 Format: "Precision: %3d, %20.*g"
104 Params: "i, i, 432432423.342321321"
105 Result VSprintf: "Precision: 2, 4.3e+08"
106 Length VSprintf: 36
107 Result LwPRINTF: "Precision: 2, 4.3e+08"
108 Length LwPRINTF: 36
109 Test result: Pass
110 ----
111 Format: "Precision: %3d, %20.*g"
112 Params: "i, i, 432432423.342321321"
113 Result VSprintf: "Precision: 3, 4.32e+08"
114 Length VSprintf: 36
115 Result LwPRINTF: "Precision: 3, 4.32e+08"
116 Length LwPRINTF: 36
117 Test result: Pass
118 ----
119 Format: "Precision: %3d, %20.*g"
120 Params: "i, i, 432432423.342321321"
121 Result VSprintf: "Precision: 4, 4.324e+08"
122 Length VSprintf: 36

```

(continues on next page)

(continued from previous page)

```

123 Result LwPRINTF: "Precision: 4, 4.324e+08"
124 Length LwPRINTF: 36
125 Test result: Pass
126 ----
127 Format: "Precision: %3d, %20.*g"
128 Params: "i, i, 432432423.342321321"
129 Result VSprintf: "Precision: 5, 4.3243e+08"
130 Length VSprintf: 36
131 Result LwPRINTF: "Precision: 5, 4.3243e+08"
132 Length LwPRINTF: 36
133 Test result: Pass
134 ----
135 Format: "Precision: %3d, %20.*g"
136 Params: "i, i, 432432423.342321321"
137 Result VSprintf: "Precision: 6, 4.32432e+08"
138 Length VSprintf: 36
139 Result LwPRINTF: "Precision: 6, 4.32432e+08"
140 Length LwPRINTF: 36
141 Test result: Pass
142 ----
143 Format: "Precision: %3d, %20.*g"
144 Params: "i, i, 432432423.342321321"
145 Result VSprintf: "Precision: 7, 4.324324e+08"
146 Length VSprintf: 36
147 Result LwPRINTF: "Precision: 7, 4.324324e+08"
148 Length LwPRINTF: 36
149 Test result: Pass
150 ----
151 Format: "Precision: %3d, %20.*g"
152 Params: "i, i, 432432423.342321321"
153 Result VSprintf: "Precision: 8, 4.3243242e+08"
154 Length VSprintf: 36
155 Result LwPRINTF: "Precision: 8, 4.3243242e+08"
156 Length LwPRINTF: 36
157 Test result: Pass
158 ----
159 Format: "Precision: %3d, %20.*g"
160 Params: "i, i, 432432423.342321321"
161 Result VSprintf: "Precision: 9, 432432423"
162 Length VSprintf: 36
163 Result LwPRINTF: "Precision: 9, 432432423"
164 Length LwPRINTF: 36
165 Test result: Pass
166 ----
167 Format: "Precision: %3d, %20.*g"
168 Params: "i, i, 432432423.342321321"
169 Result VSprintf: "Precision: 10, 432432423.3"
170 Length VSprintf: 36
171 Result LwPRINTF: "Precision: 10, 432432423.3"
172 Length LwPRINTF: 36
173 Test result: Pass
174 ----

```

(continues on next page)

(continued from previous page)

```

175 Format: "Precision: %3d, %20.*g"
176 Params: "i, i, 432432423.342321321"
177 Result VSprintf: "Precision: 11,          432432423.34"
178 Length VSprintf: 36
179 Result LwPRINTF: "Precision: 11,          432432423.34"
180 Length LwPRINTF: 36
181 Test result: Pass
182 ----
183 Format: "Precision: %3d, %20.*g"
184 Params: "i, i, 432432423.342321321"
185 Result VSprintf: "Precision: 12,          432432423.342"
186 Length VSprintf: 36
187 Result LwPRINTF: "Precision: 12,          432432423.342"
188 Length LwPRINTF: 36
189 Test result: Pass
190 ----
191 Format: "Precision: %3d, %20.*g"
192 Params: "i, i, 432432423.342321321"
193 Result VSprintf: "Precision: 13,          432432423.3423"
194 Length VSprintf: 36
195 Result LwPRINTF: "Precision: 13,          432432423.3423"
196 Length LwPRINTF: 36
197 Test result: Pass
198 ----
199 Format: "Precision: %3d, %20.*g"
200 Params: "i, i, 432432423.342321321"
201 Result VSprintf: "Precision: 14,          432432423.34232"
202 Length VSprintf: 36
203 Result LwPRINTF: "Precision: 14,          432432423.34232"
204 Length LwPRINTF: 36
205 Test result: Pass
206 ----
207 Format: "Precision: %3d, %20.*g"
208 Params: "i, i, 432432423.342321321"
209 Result VSprintf: "Precision: 15,          432432423.342321"
210 Length VSprintf: 36
211 Result LwPRINTF: "Precision: 15,          432432423.342321"
212 Length LwPRINTF: 36
213 Test result: Pass
214 ----
215 Format: "Precision: %3d, %20.*g"
216 Params: "i, i, 432432423.342321321"
217 Result VSprintf: "Precision: 16,          432432423.3423213"
218 Length VSprintf: 36
219 Result LwPRINTF: "Precision: 16,          432432423.3423213"
220 Length LwPRINTF: 36
221 Test result: Pass
222 ----
223 Format: "Precision: %3d, %20.*g"
224 Params: "i, i, 432432423.342321321"
225 Result VSprintf: "Precision: 17,          432432423.34232134"
226 Length VSprintf: 36

```

(continues on next page)

(continued from previous page)

```

227 Result LwPRINTF: "Precision: 17, 432432423.34232134"
228 Length LwPRINTF: 36
229 Test result: Pass
230 ----
231 Format: "Precision: %3d, %20.*g"
232 Params: "i, i, 432432423.342321321"
233 Result VSprintf: "Precision: 18, 432432423.342321336"
234 Length VSprintf: 36
235 Result LwPRINTF: "Precision: 18, 432432423.342321336"
236 Length LwPRINTF: 36
237 Test result: Pass
238 ----
239 Format: "Precision: %3d, %20.*g"
240 Params: "i, i, 0.0001234567"
241 Result VSprintf: "Precision: 0, 0.0001"
242 Length VSprintf: 36
243 Result LwPRINTF: "Precision: 0, 0.0001"
244 Length LwPRINTF: 36
245 Test result: Pass
246 ----
247 Format: "Precision: %3d, %20.*g"
248 Params: "i, i, 0.0001234567"
249 Result VSprintf: "Precision: 1, 0.0001"
250 Length VSprintf: 36
251 Result LwPRINTF: "Precision: 1, 0.0001"
252 Length LwPRINTF: 36
253 Test result: Pass
254 ----
255 Format: "Precision: %3d, %20.*g"
256 Params: "i, i, 0.0001234567"
257 Result VSprintf: "Precision: 2, 0.00012"
258 Length VSprintf: 36
259 Result LwPRINTF: "Precision: 2, 0.00012"
260 Length LwPRINTF: 36
261 Test result: Pass
262 ----
263 Format: "Precision: %3d, %20.*g"
264 Params: "i, i, 0.0001234567"
265 Result VSprintf: "Precision: 3, 0.000123"
266 Length VSprintf: 36
267 Result LwPRINTF: "Precision: 3, 0.000123"
268 Length LwPRINTF: 36
269 Test result: Pass
270 ----
271 Format: "Precision: %3d, %20.*g"
272 Params: "i, i, 0.0001234567"
273 Result VSprintf: "Precision: 4, 0.0001235"
274 Length VSprintf: 36
275 Result LwPRINTF: "Precision: 4, 0.0001235"
276 Length LwPRINTF: 36
277 Test result: Pass
278 ----

```

(continues on next page)

(continued from previous page)

```

279 Format: "Precision: %3d, %20.*g"
280 Params: "i, i, 0.0001234567"
281 Result VSprintf: "Precision: 5, 0.00012346"
282 Length VSprintf: 36
283 Result LwPRINTF: "Precision: 5, 0.00012346"
284 Length LwPRINTF: 36
285 Test result: Pass
286 ----
287 Format: "Precision: %3d, %20.*g"
288 Params: "i, i, 0.0001234567"
289 Result VSprintf: "Precision: 6, 0.000123457"
290 Length VSprintf: 36
291 Result LwPRINTF: "Precision: 6, 0.000123457"
292 Length LwPRINTF: 36
293 Test result: Pass
294 ----
295 Format: "Precision: %3d, %20.*g"
296 Params: "i, i, 0.0001234567"
297 Result VSprintf: "Precision: 7, 0.0001234567"
298 Length VSprintf: 36
299 Result LwPRINTF: "Precision: 7, 0.0001234567"
300 Length LwPRINTF: 36
301 Test result: Pass
302 ----
303 Format: "Precision: %3d, %20.*g"
304 Params: "i, i, 0.0001234567"
305 Result VSprintf: "Precision: 8, 0.0001234567"
306 Length VSprintf: 36
307 Result LwPRINTF: "Precision: 8, 0.0001234567"
308 Length LwPRINTF: 36
309 Test result: Pass
310 ----
311 Format: "Precision: %3d, %20.*g"
312 Params: "i, i, 0.0001234567"
313 Result VSprintf: "Precision: 9, 0.0001234567"
314 Length VSprintf: 36
315 Result LwPRINTF: "Precision: 9, 0.0001234567"
316 Length LwPRINTF: 36
317 Test result: Pass
318 ----
319 Format: "Precision: %3d, %20.*g"
320 Params: "i, i, 0.0001234567"
321 Result VSprintf: "Precision: 10, 0.0001234567"
322 Length VSprintf: 36
323 Result LwPRINTF: "Precision: 10, 0.0001234567"
324 Length LwPRINTF: 36
325 Test result: Pass
326 ----
327 Format: "Precision: %3d, %20.*g"
328 Params: "i, i, 0.0001234567"
329 Result VSprintf: "Precision: 11, 0.0001234567"
330 Length VSprintf: 36

```

(continues on next page)

(continued from previous page)

```

331 Result LwPRINTF: "Precision: 11,          0.0001234567"
332 Length LwPRINTF: 36
333 Test result: Pass
334 ----
335 Format: "Precision: %3d, %20.*g"
336 Params: "i, i, 0.0001234567"
337 Result VSprintf: "Precision: 12,          0.0001234567"
338 Length VSprintf: 36
339 Result LwPRINTF: "Precision: 12,          0.0001234567"
340 Length LwPRINTF: 36
341 Test result: Pass
342 ----
343 Format: "Precision: %3d, %20.*g"
344 Params: "i, i, 0.0001234567"
345 Result VSprintf: "Precision: 13,          0.0001234567"
346 Length VSprintf: 36
347 Result LwPRINTF: "Precision: 13,          0.0001234567"
348 Length LwPRINTF: 36
349 Test result: Pass
350 ----
351 Format: "Precision: %3d, %20.*g"
352 Params: "i, i, 0.0001234567"
353 Result VSprintf: "Precision: 14,          0.0001234567"
354 Length VSprintf: 36
355 Result LwPRINTF: "Precision: 14,          0.0001234567"
356 Length LwPRINTF: 36
357 Test result: Pass
358 ----
359 Format: "Precision: %3d, %20.*g"
360 Params: "i, i, 0.0001234567"
361 Result VSprintf: "Precision: 15,          0.0001234567"
362 Length VSprintf: 36
363 Result LwPRINTF: "Precision: 15,          0.0001234567"
364 Length LwPRINTF: 36
365 Test result: Pass
366 ----
367 Format: "Precision: %3d, %20.*g"
368 Params: "i, i, 0.0001234567"
369 Result VSprintf: "Precision: 16,          0.0001234567"
370 Length VSprintf: 36
371 Result LwPRINTF: "Precision: 16,          0.0001234567"
372 Length LwPRINTF: 36
373 Test result: Pass
374 ----
375 Format: "Precision: %3d, %20.*g"
376 Params: "i, i, 0.0001234567"
377 Result VSprintf: "Precision: 17,          0.0001234567"
378 Length VSprintf: 36
379 Result LwPRINTF: "Precision: 17,          0.0001234567"
380 Length LwPRINTF: 36
381 Test result: Pass
382 ----

```

(continues on next page)

(continued from previous page)

```

383 Format: "Precision: %3d, %20.*g"
384 Params: "i, i, 0.0001234567"
385 Result VSprintf: "Precision: 18,          0.0001234567"
386 Length VSprintf: 36
387 Result LwPRINTF: "Precision: 18,          0.0001234567"
388 Length LwPRINTF: 36
389 Test result: Pass
390 ----
391 Format: "%.4f"
392 Params: "3.23321321"
393 Result VSprintf: "3.2332"
394 Length VSprintf: 6
395 Result LwPRINTF: "3.2332"
396 Length LwPRINTF: 6
397 Test result: Pass
398 ----
399 Format: "%.4F"
400 Params: "3.23321321"
401 Result VSprintf: "3.2332"
402 Length VSprintf: 6
403 Result LwPRINTF: "3.2332"
404 Length LwPRINTF: 6
405 Test result: Pass
406 ----
407 Format: "%g"
408 Params: "1.23342"
409 Result VSprintf: "1.23342"
410 Length VSprintf: 7
411 Result LwPRINTF: "1.23342"
412 Length LwPRINTF: 7
413 Test result: Pass
414 ----
415 Format: "%g"
416 Params: "12334.2"
417 Result VSprintf: "12334.2"
418 Length VSprintf: 7
419 Result LwPRINTF: "12334.2"
420 Length LwPRINTF: 7
421 Test result: Pass
422 ----
423 Format: "%.8g"
424 Params: "0.000000123342"
425 Result VSprintf: "1.23342e-07"
426 Length VSprintf: 11
427 Result LwPRINTF: "1.23342e-07"
428 Length LwPRINTF: 11
429 Test result: Pass
430 ----
431 Format: "%.8G"
432 Params: "0.000000123342"
433 Result VSprintf: "1.23342E-07"
434 Length VSprintf: 11

```

(continues on next page)

(continued from previous page)

```

435 Result LwPRINTF: "1.23342E-07"
436 Length LwPRINTF: 11
437 Test result: Pass
438 ----
439 Format: "%.4f"
440 Params: "323243432432432.432"
441 Result VSprintf: "323243432432432.4375"
442 Length VSprintf: 20
443 Result LwPRINTF: "323243432432432.4375"
444 Length LwPRINTF: 20
445 Test result: Pass
446 ----
447 Format: "%e"
448 Params: "-123.456"
449 Result VSprintf: "-1.234560e+02"
450 Length VSprintf: 13
451 Result LwPRINTF: "-1.234560e+02"
452 Length LwPRINTF: 13
453 Test result: Pass
454 ----
455 Format: "%e"
456 Params: "0.000001"
457 Result VSprintf: "1.000000e-06"
458 Length VSprintf: 12
459 Result LwPRINTF: "1.000000e-06"
460 Length LwPRINTF: 12
461 Test result: Pass
462 ----
463 Format: "%e"
464 Params: "0.123456"
465 Result VSprintf: "1.234560e-01"
466 Length VSprintf: 12
467 Result LwPRINTF: "1.234560e-01"
468 Length LwPRINTF: 12
469 Test result: Pass
470 ----
471 Format: "%e"
472 Params: "-0.123456"
473 Result VSprintf: "-1.234560e-01"
474 Length VSprintf: 13
475 Result LwPRINTF: "-1.234560e-01"
476 Length LwPRINTF: 13
477 Test result: Pass
478 ----
479 Format: "%.4e"
480 Params: "123.456"
481 Result VSprintf: "1.2346e+02"
482 Length VSprintf: 10
483 Result LwPRINTF: "1.2346e+02"
484 Length LwPRINTF: 10
485 Test result: Pass
486 ----

```

(continues on next page)

(continued from previous page)

```

487 Format: "%.4e"
488 Params: "-123.456"
489 Result VSprintf: "-1.2346e+02"
490 Length VSprintf: 11
491 Result LwPRINTF: "-1.2346e+02"
492 Length LwPRINTF: 11
493 Test result: Pass
494 ----
495 Format: "%.4e"
496 Params: "0.123456"
497 Result VSprintf: "1.2346e-01"
498 Length VSprintf: 10
499 Result LwPRINTF: "1.2346e-01"
500 Length LwPRINTF: 10
501 Test result: Pass
502 ----
503 Format: "%.4e"
504 Params: "-0.123456"
505 Result VSprintf: "-1.2346e-01"
506 Length VSprintf: 11
507 Result LwPRINTF: "-1.2346e-01"
508 Length LwPRINTF: 11
509 Test result: Pass
510 ----
511 Format: "%.0e"
512 Params: "123.456"
513 Result VSprintf: "1e+02"
514 Length VSprintf: 5
515 Result LwPRINTF: "1e+02"
516 Length LwPRINTF: 5
517 Test result: Pass
518 ----
519 Format: "%.0e"
520 Params: "-123.456"
521 Result VSprintf: "-1e+02"
522 Length VSprintf: 6
523 Result LwPRINTF: "-1e+02"
524 Length LwPRINTF: 6
525 Test result: Pass
526 ----
527 Format: "%.0e"
528 Params: "0.123456"
529 Result VSprintf: "1e-01"
530 Length VSprintf: 5
531 Result LwPRINTF: "1e-01"
532 Length LwPRINTF: 5
533 Test result: Pass
534 ----
535 Format: "%.0e"
536 Params: "-0.123456"
537 Result VSprintf: "-1e-01"
538 Length VSprintf: 6

```

(continues on next page)

(continued from previous page)

```

539 Result LwPRINTF: "-1e-01"
540 Length LwPRINTF: 6
541 Test result: Pass
542 ----
543 Format: "%22.4e"
544 Params: "123.456"
545 Result VSprintf: "          1.2346e+02"
546 Length VSprintf: 22
547 Result LwPRINTF: "          1.2346e+02"
548 Length LwPRINTF: 22
549 Test result: Pass
550 ----
551 Format: "%22.4e"
552 Params: "-123.456"
553 Result VSprintf: "        -1.2346e+02"
554 Length VSprintf: 22
555 Result LwPRINTF: "        -1.2346e+02"
556 Length LwPRINTF: 22
557 Test result: Pass
558 ----
559 Format: "%22.4e"
560 Params: "0.123456"
561 Result VSprintf: "          1.2346e-01"
562 Length VSprintf: 22
563 Result LwPRINTF: "          1.2346e-01"
564 Length LwPRINTF: 22
565 Test result: Pass
566 ----
567 Format: "%22.4e"
568 Params: "-0.123456"
569 Result VSprintf: "        -1.2346e-01"
570 Length VSprintf: 22
571 Result LwPRINTF: "        -1.2346e-01"
572 Length LwPRINTF: 22
573 Test result: Pass
574 ----
575 Format: "%022.4e"
576 Params: "123.456"
577 Result VSprintf: "00000000000001.2346e+02"
578 Length VSprintf: 22
579 Result LwPRINTF: "00000000000001.2346e+02"
580 Length LwPRINTF: 22
581 Test result: Pass
582 ----
583 Format: "%022.4e"
584 Params: "-123.456"
585 Result VSprintf: "-00000000000001.2346e+02"
586 Length VSprintf: 22
587 Result LwPRINTF: "-00000000000001.2346e+02"
588 Length LwPRINTF: 22
589 Test result: Pass
590 ----

```

(continues on next page)

(continued from previous page)

```

591 Format: "%022.4e"
592 Params: "0.123456"
593 Result VSprintf: "00000000000001.2346e-01"
594 Length VSprintf: 22
595 Result LwPRINTF: "00000000000001.2346e-01"
596 Length LwPRINTF: 22
597 Test result: Pass
598 ----
599 Format: "%e"
600 Params: "0.00000000123456"
601 Result VSprintf: "1.234560e-09"
602 Length VSprintf: 12
603 Result LwPRINTF: "1.234560e-09"
604 Length LwPRINTF: 12
605 Test result: Pass
606 ----
607 Format: "%022.4e"
608 Params: "-0.123456"
609 Result VSprintf: "-00000000000001.2346e-01"
610 Length VSprintf: 22
611 Result LwPRINTF: "-00000000000001.2346e-01"
612 Length LwPRINTF: 22
613 Test result: Pass
614 ----
615 Format: "%.4E"
616 Params: "-123.456"
617 Result VSprintf: "-1.2346E+02"
618 Length VSprintf: 11
619 Result LwPRINTF: "-1.2346E+02"
620 Length LwPRINTF: 11
621 Test result: Pass
622 ----
623 Format: "% 3u"
624 Params: "(unsigned)28"
625 Result VSprintf: " 28"
626 Length VSprintf: 3
627 Result LwPRINTF: " 28"
628 Length LwPRINTF: 3
629 Test result: Pass
630 ----
631 Format: "% 3u"
632 Params: "(unsigned)123456"
633 Result VSprintf: "123456"
634 Length VSprintf: 6
635 Result LwPRINTF: "123456"
636 Length LwPRINTF: 6
637 Test result: Pass
638 ----
639 Format: "%03d"
640 Params: "28"
641 Result VSprintf: "028"
642 Length VSprintf: 3

```

(continues on next page)

(continued from previous page)

```

643 Result LwPRINTF: "028"
644 Length LwPRINTF: 3
645 Test result: Pass
646 ----
647 Format: "%+03d"
648 Params: "28"
649 Result VSprintf: "+28"
650 Length VSprintf: 3
651 Result LwPRINTF: "+28"
652 Length LwPRINTF: 3
653 Test result: Pass
654 ----
655 Format: "%+3d"
656 Params: "28"
657 Result VSprintf: "+28"
658 Length VSprintf: 3
659 Result LwPRINTF: "+28"
660 Length LwPRINTF: 3
661 Test result: Pass
662 ----
663 Format: "%03d"
664 Params: "-28"
665 Result VSprintf: "-28"
666 Length VSprintf: 3
667 Result LwPRINTF: "-28"
668 Length LwPRINTF: 3
669 Test result: Pass
670 ----
671 Format: "%+03d"
672 Params: "-28"
673 Result VSprintf: "-28"
674 Length VSprintf: 3
675 Result LwPRINTF: "-28"
676 Length LwPRINTF: 3
677 Test result: Pass
678 ----
679 Format: "%+3d"
680 Params: "-28"
681 Result VSprintf: "-28"
682 Length VSprintf: 3
683 Result LwPRINTF: "-28"
684 Length LwPRINTF: 3
685 Test result: Pass
686 ----
687 Format: "%03u"
688 Params: "(unsigned)123456"
689 Result VSprintf: "123456"
690 Length VSprintf: 6
691 Result LwPRINTF: "123456"
692 Length LwPRINTF: 6
693 Test result: Pass
694 ----

```

(continues on next page)

(continued from previous page)

```

695 Format: "%-010uabc"
696 Params: "(unsigned)123456"
697 Result VSprintf: "123456      abc"
698 Length VSprintf: 13
699 Result LwPRINTF: "123456      abc"
700 Length LwPRINTF: 13
701 Test result: Pass
702 ----
703 Format: "%010uabc"
704 Params: "(unsigned)123456"
705 Result VSprintf: "0000123456abc"
706 Length VSprintf: 13
707 Result LwPRINTF: "0000123456abc"
708 Length LwPRINTF: 13
709 Test result: Pass
710 ----
711 Format: "%-10d"
712 Params: "-123"
713 Result VSprintf: "-123      "
714 Length VSprintf: 10
715 Result LwPRINTF: "-123      "
716 Length LwPRINTF: 10
717 Test result: Pass
718 ----
719 Format: "%10d"
720 Params: "-123"
721 Result VSprintf: "      -123"
722 Length VSprintf: 10
723 Result LwPRINTF: "      -123"
724 Length LwPRINTF: 10
725 Test result: Pass
726 ----
727 Format: "%-06d"
728 Params: "-1234567"
729 Result VSprintf: "-1234567"
730 Length VSprintf: 8
731 Result LwPRINTF: "-1234567"
732 Length LwPRINTF: 8
733 Test result: Pass
734 ----
735 Format: "%06d"
736 Params: "-1234567"
737 Result VSprintf: "-1234567"
738 Length VSprintf: 8
739 Result LwPRINTF: "-1234567"
740 Length LwPRINTF: 8
741 Test result: Pass
742 ----
743 Format: "%-10d"
744 Params: "-1234567"
745 Result VSprintf: "-1234567  "
746 Length VSprintf: 10

```

(continues on next page)

(continued from previous page)

```

747 Result LwPRINTF: "-1234567 "
748 Length LwPRINTF: 10
749 Test result: Pass
750 ----
751 Format: "%10d"
752 Params: "-1234567"
753 Result VSprintf: " -1234567"
754 Length VSprintf: 10
755 Result LwPRINTF: " -1234567"
756 Length LwPRINTF: 10
757 Test result: Pass
758 ----
759 Format: "%-010d"
760 Params: "-1234567"
761 Result VSprintf: "-1234567 "
762 Length VSprintf: 10
763 Result LwPRINTF: "-1234567 "
764 Length LwPRINTF: 10
765 Test result: Pass
766 ----
767 Format: "%010d"
768 Params: "-1234567"
769 Result VSprintf: "-001234567"
770 Length VSprintf: 10
771 Result LwPRINTF: "-001234567"
772 Length LwPRINTF: 10
773 Test result: Pass
774 ----
775 Format: "%s"
776 Params: ""This is my string""
777 Result VSprintf: "This is my string"
778 Length VSprintf: 17
779 Result LwPRINTF: "This is my string"
780 Length LwPRINTF: 17
781 Test result: Pass
782 ----
783 Format: "%10s"
784 Params: ""This is my string""
785 Result VSprintf: "This is my string"
786 Length VSprintf: 17
787 Result LwPRINTF: "This is my string"
788 Length LwPRINTF: 17
789 Test result: Pass
790 ----
791 Format: "%0*d"
792 Params: "10, -123"
793 Result VSprintf: "-000000123"
794 Length VSprintf: 10
795 Result LwPRINTF: "-000000123"
796 Length LwPRINTF: 10
797 Test result: Pass
798 ----

```

(continues on next page)

(continued from previous page)

```
799 Format: "%zu"
800 Params: "(size_t)10"
801 Result VSprintf: "10"
802 Length VSprintf: 2
803 Result LwPRINTF: "10"
804 Length LwPRINTF: 2
805 Test result: Pass
806 ----
807 Format: "%ju"
808 Params: "(uintmax_t)10"
809 Result VSprintf: "10"
810 Length VSprintf: 2
811 Result LwPRINTF: "10"
812 Length LwPRINTF: 2
813 Test result: Pass
814 ----
815 Format: "% d"
816 Params: "1024"
817 Result VSprintf: " 1024"
818 Length VSprintf: 5
819 Result LwPRINTF: " 1024"
820 Length LwPRINTF: 5
821 Test result: Pass
822 ----
823 Format: "% 4d"
824 Params: "1024"
825 Result VSprintf: " 1024"
826 Length VSprintf: 5
827 Result LwPRINTF: " 1024"
828 Length LwPRINTF: 5
829 Test result: Pass
830 ----
831 Format: "% 3d"
832 Params: "1024"
833 Result VSprintf: " 1024"
834 Length VSprintf: 5
835 Result LwPRINTF: " 1024"
836 Length LwPRINTF: 5
837 Test result: Pass
838 ----
839 Format: "% 3f"
840 Params: "32.687"
841 Result VSprintf: " 32.687000"
842 Length VSprintf: 10
843 Result LwPRINTF: " 32.687000"
844 Length LwPRINTF: 10
845 Test result: Pass
846 ----
847 Format: "%*.*s"
848 Params: "8, 12, \"This is my string\""
849 Result VSprintf: "This is my s"
850 Length VSprintf: 12
```

(continues on next page)

(continued from previous page)

```

851 Result LwPRINTF: "This is my s"
852 Length LwPRINTF: 12
853 Test result: Pass
854 ----
855 Format: "%*.*s"
856 Params: "8, 12, "Stri""
857 Result VSprintf: "    Stri"
858 Length VSprintf: 8
859 Result LwPRINTF: "    Stri"
860 Length LwPRINTF: 8
861 Test result: Pass
862 ----
863 Format: "%-6.10s"
864 Params: ""This is my string""
865 Result VSprintf: "This is my"
866 Length VSprintf: 10
867 Result LwPRINTF: "This is my"
868 Length LwPRINTF: 10
869 Test result: Pass
870 ----
871 Format: "%6.10s"
872 Params: ""This is my string""
873 Result VSprintf: "This is my"
874 Length VSprintf: 10
875 Result LwPRINTF: "This is my"
876 Length LwPRINTF: 10
877 Test result: Pass
878 ----
879 Format: "%-6.10s"
880 Params: ""This is my string""
881 Result VSprintf: "This is my"
882 Length VSprintf: 10
883 Result LwPRINTF: "This is my"
884 Length LwPRINTF: 10
885 Test result: Pass
886 ----
887 Format: "%6.10s"
888 Params: ""Th""
889 Result VSprintf: "    Th"
890 Length VSprintf: 6
891 Result LwPRINTF: "    Th"
892 Length LwPRINTF: 6
893 Test result: Pass
894 ----
895 Format: "%-6.10s"
896 Params: ""Th""
897 Result VSprintf: "Th    "
898 Length VSprintf: 6
899 Result LwPRINTF: "Th    "
900 Length LwPRINTF: 6
901 Test result: Pass
902 ----

```

(continues on next page)

(continued from previous page)

```

903 Format: "%*.*s"
904 Params: "-6, 10, "Th""
905 Result VSprintf: "Th  "
906 Length VSprintf: 6
907 Result LwPRINTF: "Th  "
908 Length LwPRINTF: 6
909 Test result: Pass
910 ----
911 Format: "%*.*s"
912 Params: "6, 10, "Th""
913 Result VSprintf: "    Th"
914 Length VSprintf: 6
915 Result LwPRINTF: "    Th"
916 Length LwPRINTF: 6
917 Test result: Pass
918 ----
919 Format: "%.4s"
920 Params: ""This is my string""
921 Result VSprintf: "This"
922 Length VSprintf: 4
923 Result LwPRINTF: "This"
924 Length LwPRINTF: 4
925 Test result: Pass
926 ----
927 Format: "%.6s"
928 Params: ""1234""
929 Result VSprintf: "1234"
930 Length VSprintf: 4
931 Result LwPRINTF: "1234"
932 Length LwPRINTF: 4
933 Test result: Pass
934 ----
935 Format: "%.4s"
936 Params: ""stri""
937 Result VSprintf: "stri"
938 Length VSprintf: 4
939 Result LwPRINTF: "stri"
940 Length LwPRINTF: 4
941 Test result: Pass
942 ----
943 Format: "%.4s%.2s"
944 Params: ""123456", "abcdef""
945 Result VSprintf: "1234ab"
946 Length VSprintf: 6
947 Result LwPRINTF: "1234ab"
948 Length LwPRINTF: 6
949 Test result: Pass
950 ----
951 Format: "%.4.2s"
952 Params: ""123456""
953 Result VSprintf: ".2s"
954 Length VSprintf: 3

```

(continues on next page)

(continued from previous page)

```

955 Result LwPRINTF: ".2s"
956 Length LwPRINTF: 3
957 Test result: Pass
958 ----
959 Format: "%.s"
960 Params: "3, "123456""
961 Result VSprintf: "123"
962 Length VSprintf: 3
963 Result LwPRINTF: "123"
964 Length LwPRINTF: 3
965 Test result: Pass
966 ----
967 Format: "%.3s"
968 Params: "123456"
969 Result VSprintf: ""
970 Length VSprintf: 0
971 Result LwPRINTF: ""
972 Length LwPRINTF: 0
973 Test result: Pass
974 ----
975 Format: "%yunknown"
976 Params: "123456"
977 Result VSprintf: "yunknown"
978 Length VSprintf: 8
979 Result LwPRINTF: "yunknown"
980 Length LwPRINTF: 8
981 Test result: Pass
982 ----
983 Format: "%#2X"
984 Params: "123"
985 Result VSprintf: "0X7B"
986 Length VSprintf: 4
987 Result LwPRINTF: "0X7B"
988 Length LwPRINTF: 4
989 Test result: Pass
990 ----
991 Format: "%#2x"
992 Params: "123"
993 Result VSprintf: "0x7b"
994 Length VSprintf: 4
995 Result LwPRINTF: "0x7b"
996 Length LwPRINTF: 4
997 Test result: Pass
998 ----
999 Format: "%#2o"
1000 Params: "123"
1001 Result VSprintf: "0173"
1002 Length VSprintf: 4
1003 Result LwPRINTF: "0173"
1004 Length LwPRINTF: 4
1005 Test result: Pass
1006 ----

```

(continues on next page)

(continued from previous page)

```

1007 Format: "%#2X"
1008 Params: "1"
1009 Result VPrintf: "0X1"
1010 Length VPrintf: 3
1011 Result LwPRINTF: "0X1"
1012 Length LwPRINTF: 3
1013 Test result: Pass
1014 ----
1015 Format: "%#2x"
1016 Params: "1"
1017 Result VPrintf: "0x1"
1018 Length VPrintf: 3
1019 Result LwPRINTF: "0x1"
1020 Length LwPRINTF: 3
1021 Test result: Pass
1022 ----
1023 Format: "%#2o"
1024 Params: "1"
1025 Result VPrintf: "01"
1026 Length VPrintf: 2
1027 Result LwPRINTF: "01"
1028 Length LwPRINTF: 2
1029 Test result: Pass
1030 ----
1031 Format: "%#2X"
1032 Params: "0"
1033 Result VPrintf: " 0"
1034 Length VPrintf: 2
1035 Result LwPRINTF: " 0"
1036 Length LwPRINTF: 2
1037 Test result: Pass
1038 ----
1039 Format: "%#2x"
1040 Params: "0"
1041 Result VPrintf: " 0"
1042 Length VPrintf: 2
1043 Result LwPRINTF: " 0"
1044 Length LwPRINTF: 2
1045 Test result: Pass
1046 ----
1047 Format: "%#2o"
1048 Params: "0"
1049 Result VPrintf: " 0"
1050 Length VPrintf: 2
1051 Result LwPRINTF: " 0"
1052 Length LwPRINTF: 2
1053 Test result: Pass
1054 ----
1055 Format: "%p"
1056 Params: "&tests_passed"
1057 Result VPrintf: "00BE0FD4"
1058 Length VPrintf: 8

```

(continues on next page)

(continued from previous page)

```

1059 Result LwPRINTF: "00BE0FD4"
1060 Length LwPRINTF: 8
1061 Test result: Pass
1062 ----
1063 Format: "0X%p"
1064 Params: "&tests_passed"
1065 Result VSprintf: "0X00BE0FD4"
1066 Length VSprintf: 10
1067 Result LwPRINTF: "0X00BE0FD4"
1068 Length LwPRINTF: 10
1069 Test result: Pass
1070 ----
1071 Format: "0x%p"
1072 Params: "&tests_passed"
1073 Result VSprintf: "0x00BE0FD4"
1074 Length VSprintf: 10
1075 Result LwPRINTF: "0x00BE0FD4"
1076 Length LwPRINTF: 10
1077 Test result: Pass
1078 ----
1079 Format: "%11b abc"
1080 Params: "123"
1081 Result expected: "1111011 abc"
1082 Length expected: 11
1083 Result LwPRINTF: "1111011 abc"
1084 Length LwPRINTF: 11
1085 Test result: Pass
1086 ----
1087 Format: "%b"
1088 Params: "4"
1089 Result expected: "100"
1090 Length expected: 3
1091 Result LwPRINTF: "100"
1092 Length LwPRINTF: 3
1093 Test result: Pass
1094 ----
1095 Format: "%#2B"
1096 Params: "1"
1097 Result expected: "0B1"
1098 Length expected: 3
1099 Result LwPRINTF: "0B1"
1100 Length LwPRINTF: 3
1101 Test result: Pass
1102 ----
1103 Format: "%#2b"
1104 Params: "1"
1105 Result expected: "0b1"
1106 Length expected: 3
1107 Result LwPRINTF: "0b1"
1108 Length LwPRINTF: 3
1109 Test result: Pass
1110 ----

```

(continues on next page)

(continued from previous page)

```

1111 Format: "%#2B"
1112 Params: "0"
1113 Result expected: " 0"
1114 Length expected: 2
1115 Result LwPRINTF: " 0"
1116 Length LwPRINTF: 2
1117 Test result: Pass
1118 ----
1119 Format: "%#2b"
1120 Params: "0"
1121 Result expected: " 0"
1122 Length expected: 2
1123 Result LwPRINTF: " 0"
1124 Length LwPRINTF: 2
1125 Test result: Pass
1126 ----
1127 Format: "%#B"
1128 Params: "0"
1129 Result expected: "0"
1130 Length expected: 1
1131 Result LwPRINTF: "0"
1132 Length LwPRINTF: 1
1133 Test result: Pass
1134 ----
1135 Format: "%#b"
1136 Params: "0"
1137 Result expected: "0"
1138 Length expected: 1
1139 Result LwPRINTF: "0"
1140 Length LwPRINTF: 1
1141 Test result: Pass
1142 ----
1143 Format: "%#B"
1144 Params: "6"
1145 Result expected: "0B110"
1146 Length expected: 5
1147 Result LwPRINTF: "0B110"
1148 Length LwPRINTF: 5
1149 Test result: Pass
1150 ----
1151 Format: "%#b"
1152 Params: "6"
1153 Result expected: "0b110"
1154 Length expected: 5
1155 Result LwPRINTF: "0b110"
1156 Length LwPRINTF: 5
1157 Test result: Pass
1158 ----
1159 Format: "%5K"
1160 Params: "my_arr"
1161 Result expected: "0102B5C6D7"
1162 Length expected: 10

```

(continues on next page)

(continued from previous page)

```

1163 Result LwPRINTF: "0102B5C6D7"
1164 Length LwPRINTF: 10
1165 Test result: Pass
1166 ----
1167 Format: "%*K"
1168 Params: "3, my_arr"
1169 Result expected: "0102B5"
1170 Length expected: 6
1171 Result LwPRINTF: "0102B5"
1172 Length LwPRINTF: 6
1173 Test result: Pass
1174 ----
1175 Format: "% *K"
1176 Params: "3, my_arr"
1177 Result expected: "01 02 B5"
1178 Length expected: 8
1179 Result LwPRINTF: "01 02 B5"
1180 Length LwPRINTF: 8
1181 Test result: Pass
1182 ----
1183 Format: "%5k"
1184 Params: "my_arr"
1185 Result expected: "0102b5c6d7"
1186 Length expected: 10
1187 Result LwPRINTF: "0102b5c6d7"
1188 Length LwPRINTF: 10
1189 Test result: Pass
1190 ----
1191 Format: "%*k"
1192 Params: "3, my_arr"
1193 Result expected: "0102b5"
1194 Length expected: 6
1195 Result LwPRINTF: "0102b5"
1196 Length LwPRINTF: 6
1197 Test result: Pass
1198 ----
1199 Format: "% *k"
1200 Params: "3, my_arr"
1201 Result expected: "01 02 b5"
1202 Length expected: 8
1203 Result LwPRINTF: "01 02 b5"
1204 Length LwPRINTF: 8
1205 Test result: Pass

```

5.5 Examples and demos

Various examples are provided for fast library evaluation on embedded systems. These are prepared and maintained for 2 platforms, but could be easily extended to more platforms:

- WIN32 examples, prepared as [Visual Studio Community](#) projects
- ARM Cortex-M examples for STM32, prepared as [STM32CubeIDE](#) GCC projects

Warning: Library is platform independent and can be used on any platform.

5.5.1 Debug for STM32L4

Simple example is available, that runs on *STM32L432KC-Nucleo* board and shows basic configuration for library. On-board *Virtual-COM-Port* through embedded ST-Link provides communication to MCU via UART peripheral.

Output function writes data to PC using *USART2* hardware IP.

5.6 Changelog

```
# Changelog

## Develop

- Add `lwprintf_debug` and `lwprintf_debug_cond` functions

## v1.0.5

- Fix building the library with `LWPRINTF_CFG_OS=1` and `LWPRINTF_CFG_OS_MANUAL_
  ↳PROTECT=0` options

## v1.0.4

- Fix calculation for NULL terminated string and precision with 0 as an input
- Split CMakeLists.txt files between library and executable
- Fix missing break in switch statement
- Add support for manual mutual-exclusion setup in OS mode
- Change license year to 2022
- Update code style with astyle
- Add `.clang-format` draft
- Fix protection functions for when print mode is not used

## v1.0.3

- CMSIS-OS improvements for Kernel aware debuggers

## v1.0.2

- Fixed `float` output when engineering mode is disabled
```

(continues on next page)

(continued from previous page)

```
## v1.0.1

- Fixed compiler error when engineering mode disabled but float enabled
- Properly handled `zero` float inputs

## v1.0.0

- First stable release
- Embedded systems optimized library
- Apply all modifiers except `%a`
- Extensive docs available
- Operating system ready with CMSIS-OS template
```

5.7 Authors

List of authors and contributors to the library

```
Tilen Majerle <tilen.majerle@gmail.com>
Tilen Majerle <tilen@majerle.eu>
Okarss <104319900+Okarss@users.noreply.github.com>
Dmitry Karasev <karasevsdmitry@yandex.ru>
Brian <bayuan@purdue.edu>
Peter Maxwell Warasila <madmaxwell@soundcomesout.com>
```


L

lwprintf (*C macro*), 26
 LWPRINTF_ARRAYSIZE (*C macro*), 24
 LWPRINTF_CFG_ENABLE_SHORTNAMES (*C macro*), 35
 LWPRINTF_CFG_ENABLE_STD_NAMES (*C macro*), 35
 LWPRINTF_CFG_FLOAT_DEFAULT_PRECISION (*C macro*), 35
 LWPRINTF_CFG_OS (*C macro*), 34
 LWPRINTF_CFG_OS_MANUAL_PROTECT (*C macro*), 34
 LWPRINTF_CFG_OS_MUTEX_HANDLE (*C macro*), 34
 LWPRINTF_CFG_SUPPORT_LONG_LONG (*C macro*), 34
 LWPRINTF_CFG_SUPPORT_TYPE_BYTE_ARRAY (*C macro*), 35
 LWPRINTF_CFG_SUPPORT_TYPE_ENGINEERING (*C macro*), 34
 LWPRINTF_CFG_SUPPORT_TYPE_FLOAT (*C macro*), 34
 LWPRINTF_CFG_SUPPORT_TYPE_INT (*C macro*), 34
 LWPRINTF_CFG_SUPPORT_TYPE_POINTER (*C macro*), 34
 LWPRINTF_CFG_SUPPORT_TYPE_STRING (*C macro*), 35
 lwprintf_debug (*C macro*), 30
 lwprintf_debug_cond (*C macro*), 31
 lwprintf_init (*C macro*), 25
 lwprintf_init_ex (*C++ function*), 31
 lwprintf_output_fn (*C++ type*), 31
 lwprintf_printf (*C macro*), 25
 lwprintf_printf_ex (*C++ function*), 32
 lwprintf_protect (*C macro*), 26
 lwprintf_protect_ex (*C++ function*), 33
 lwprintf_snprintf (*C macro*), 26
 lwprintf_snprintf_ex (*C++ function*), 32
 lwprintf_sprintf (*C macro*), 26
 lwprintf_sprintf_ex (*C macro*), 24
 lwprintf_sys_mutex_create (*C++ function*), 35
 lwprintf_sys_mutex_isvalid (*C++ function*), 35
 lwprintf_sys_mutex_release (*C++ function*), 36
 lwprintf_sys_mutex_wait (*C++ function*), 36
 lwprintf_t (*C++ struct*), 33
 lwprintf_t::mutex (*C++ member*), 33
 lwprintf_t::out_fn (*C++ member*), 33
 lwprintf_unprotect (*C macro*), 26
 lwprintf_unprotect_ex (*C++ function*), 33
 LWPRINTF_UNUSED (*C macro*), 24

lwprintf_vprintf (*C macro*), 25
 lwprintf_vprintf_ex (*C++ function*), 32
 lwprintf_vsnprintf (*C macro*), 25
 lwprintf_vsnprintf_ex (*C++ function*), 32
 lwsnprintf (*C macro*), 27
 lwsprintf (*C macro*), 28
 lwvprintf (*C macro*), 27
 lwvsprintf (*C macro*), 27

P

printf (*C macro*), 28

S

snprintf (*C macro*), 29
 sprintf (*C macro*), 30

V

vprintf (*C macro*), 29
 vsnprintf (*C macro*), 29