LwPRINTF

Tilen MAJERLE

Dec 10, 2023

CONTENTS

Welcome to the documentation for version v1.0.5.

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

Download library Getting started Open Github Donate

CHAPTER

ONE

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 `vsnprintf`
- 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

CHAPTER

TWO

REQUIREMENTS

- C compiler
- Few kB of non-volatile memory

CHAPTER

THREE

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

CHAPTER

FOUR

LICENSE

```
MIT License

Copyright (c) 2020 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.
```

CHAPTER

FIVE

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 primarly 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_DIR` before adding library's directory to the *CMake* project. Variable must set the output directory path. CMake will copy the template file there, and name it as required.

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

```
/**
 * \file            lwprintf_opts_template.h
 * \brief           LwPRINTF configuration file
 */

/*
 * Copyright (c) 2023 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.
 *
 * This file is part of LwPRINTF - Lightweight stdio manager library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v1.0.5
```

(continues on next page)

(continued from previous page)

```
 */
#ifndef LWPRINTF_OPTS_HDR_H
#define LWPRINTF_OPTS_HDR_H

/* Rename this file to "lwprintf_opts.h" for your application */

/*
 * Open "include/lwprintf/lwprintf_opt.h" and
 * copy & replace here settings you want to change values
 */

#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

```
#include "lwprintf/lwprintf.h"

/* Called for every character to be printed */
int
lwprintf_out(int ch, lwprintf_t* lwp) {
    /* May use printf to output it for test */
    if (ch != '\0') {
        printf("%c", (char)ch);
    }
    return ch;
}

int
main(void) {
    /* Initialize default lwprintf instance with output function */
    lwprintf_init(lwprintf_out);

    /* Print first text */
    lwprintf_printf("Text: %d", 10);
}
```

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

```
#include "lwprintf/lwprintf.h"

/* Called for every character to be printed */
```

(continues on next page)

(continued from previous page)

```c
int
lwprintf_out(int ch, lwprintf_t* lwp) {
    /* May use printf to output it for test */
    if (ch != '\0') {
        printf("%c", (char)ch);
    }
    return ch;
}

int
main(void) {
    /* Initialize default lwprintf instance with output function */
    lwprintf_init(lwprintf_out);

    /* Print first text */
    lwprintf_printf("Text: %d", 10);
}
```

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 *width* 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("%.*s", 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 `printf` to expect an `int`-sized integer argument which was promoted from a char
`h`	For integer types, causes `printf` to expect an `int`-sized integer argument which was promoted from a short
`l`	For integer types, causes `printf` to expect a `long`-sized integer argument. For floating point types, this has no effect
`ll`	For integer types, causes `printf` to expect a `long long`-sized integer argument
`L`	For floating point types, causes `printf` to expect a `long double` argument
`z`	For integer types, causes `printf` to expect a `size_t`-sized integer argument
`j`	For integer types, causes `printf` to expect a `intmax_t`-sized integer argument
`t`	For integer types, causes `printf` to expect a `ptrdiff_t`-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.

Specifier	Supported	Description
`%`	Yes	Prints literal `%` character
`d i`	Yes	Prints `signed int`. No difference between either of them
`u`	Yes	Prints `unsigned int`
`f F`	Yes	Prints `double` in normal fixed-point notation. `f` and `F` only differs in how the strings for an infinite number or NaN are printed (`inf`, `infinity` and `nan` for `f`; `INF`, `INFINITY` and `NAN` for `F`).
`e E`	Yes	Prints `double` in standard form `[-]d.ddd e[+-]ddd`. `e` uses lower-case and `E` uses upper-case letter for exponent annotation.
`g G`	Yes	Prints `double` in either normal or exponential notation, whichever is more appropriate for its magnitude. `g` uses lower-case letters, `G` 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 `unsigned int` as a hexadecimal number. `x` uses lower-case and `X` uses upper-case letters
`o`	Yes	Prints `unsigned int` in octal format
`s`	Yes	Prints null terminated string
`c`	Yes	Prints `char` type
`p`	Yes	Prints `void *` in an hex-based format. Reads input as `unsigned int` by default.
`a A`	Not yet	Prints `double` 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

Specifier	Description
`B` `b`	Prints `unsigned int` data as binary representation.
`K` `k`	Prints `unsigned char` based data array as sequence of hex numbers. Use *width* field to specify length of input array. Use `K` for upper-case hex letters, `k` for lower-case.

Listing 4: Additional format specifiers

```c
/* List of specifiers added in the library which are not available in standard printf implementation */

#include "lwprintf/lwprintf.h"

/**
 * \brief           List of additional specifiers to print
 */
void
additional_format_specifiers(void) {
    unsigned char my_array[] = { 0x01, 0x02, 0xA4, 0xB5, 0xC6 };

    /* Binary output */

    /* Prints number 8 in binary format, so "1000" */
    lwprintf_printf("%b\r\n", 8U);
    /* Prints number 16 in binary format with 10 places, so "     10000" */
    lwprintf_printf("%10b\r\n", 16U);
    /* Prints number 16 in binary format with 10 places, leading zeros, so "0000010000" */
    lwprintf_printf("%010b\r\n", 16U);

    /* Array outputs */

    /* Fixed length with uppercase hex numbers, outputs "0102A4B5C6" */
    lwprintf_printf("%5K\r\n", my_array);
    /* Fixed length with lowercase hex numbers, outputs "0102a4b5c6" */
    lwprintf_printf("%5k\r\n", my_array);
    /* Variable length with uppercase letters, outputs "0102A4B5C6" */
    lwprintf_printf("%*K\r\n", (int)LWPRINTF_ARRAYSIZE(my_array), my_array);
    /* Variable length with lowercase letters, outputs "0102a4b5c6" */
    lwprintf_printf("%*k\r\n", (int)LWPRINTF_ARRAYSIZE(my_array), my_array);
    /* Variable length with uppercase letters and spaces, outputs "01 02 A4 B5 C6" */
    lwprintf_printf("% *K\r\n", (int)LWPRINTF_ARRAYSIZE(my_array), my_array);
    /* Variable length with uppercase letters and spaces, outputs "01 02 a4 b5 c6" */
    lwprintf_printf("% *k\r\n", (int)LWPRINTF_ARRAYSIZE(my_array), my_array);
}
```

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, suchs 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 direcly work with custom instances. Functions without `_ex` suffix use default LwPRINTF instance

---

Listing 5: Custom LwPRINTF instance for output

```
#include "lwprintf/lwprintf.h"

/* Define application custom instance */
lwprintf_t custom_instance;

/* Define custom output function for print */
int
custom_out(int ch, lwprintf_t* p) {
    /* Do whatever with this character */
    if (ch == '\0') {
        /* This is end of string in current formatting */
        /* Maybe time to start DMA transfer? */
    } else {
        /* Print or send character */
    }

    /* Return character to proceed */
    return ch;
}

/* Define output function for default instance */
int
default_out(int ch, lwprintf_t* p) {
    /* Print function for default instance */

    /* See custom_out function for implementation details */
}

int
main(void) {
    /* Initialize default lwprintf instance with output function */
    lwprintf_init(default_out);
    /* Initialize custom lwprintf instance with output function */
    lwprintf_init_ex(&custom_instance, custom_out);

```

(continues on next page)

(continued from previous page)

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

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

```
#include "lwprintf/lwprintf.h"

/* Define application custom instance */
lwprintf_t custom_instance1;
lwprintf_t custom_instance2;

/* Define custom output function for print */
int
my_out(int ch, lwprintf_t* p) {
    if (p == &custom_instance1) {
        /* This is custom instance 1 */
    } else if (p == &custom_instance2) {
        /* This is custom instance 2 */
    } else {
        /* This is default instance */
    }
    return ch;
}

int
main(void) {
    /* Initialize default lwprintf instance with output function */
    lwprintf_init(my_out);
    lwprintf_init_ex(&custom_instance1, my_out);
    lwprintf_init_ex(&custom_instance2, my_out);

    /* Use print functions ... */
}
```

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

```c
#include "lwprintf/lwprintf.h"

/* Assuming LwPRINTF has been initialized before */

void
task_1(void* arg) {
        lwprintf_printf("Hello world\r\n");
}

void
task_2(void* arg) {
        lwprintf_printf("This is Task 2\r\n");
}

/*
 * If thread safety is not enabled,
 * running above example may print:
 *
 * "Hello This is Task 2\r\nworld\r\n"
 */
```

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

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

(continues on next page)

(continued from previous page)

```

/*
 * Copyright (c) 2023 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.
 *
 * This file is part of LwPRINTF - Lightweight stdio manager library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v1.0.5
 */
#include "system/lwprintf_sys.h"

#if LWPRINTF_CFG_OS && !__DOXYGEN__

#include "cmsis_os.h"

uint8_t
lwprintf_sys_mutex_create(LWPRINTF_CFG_OS_MUTEX_HANDLE* m) {
    const osMutexAttr_t attr = {
        .name = "lwprintf_mutex",
        .attr_bits = osMutexRecursive,
    };
    return (*m = osMutexNew(&attr)) != NULL;
}

uint8_t
lwprintf_sys_mutex_isvalid(LWPRINTF_CFG_OS_MUTEX_HANDLE* m) {
    return *m != NULL;
}

uint8_t
lwprintf_sys_mutex_wait(LWPRINTF_CFG_OS_MUTEX_HANDLE* m) {
    return osMutexAcquire(*m, osWaitForever) == osOK;
```

(continues on next page)

(continued from previous page)

```
}

uint8_t
lwprintf_sys_mutex_release(LWPRINTF_CFG_OS_MUTEX_HANDLE* m) {
    return osMutexRelease(*m) == osOK;
}

#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 <cstdarg>.

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 <cstdarg>.

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 *LWPRINTF_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 <cstdarg>.

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

lwvsnprintf

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 <cstdarg>.

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

lwsnprintf

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 <`cstdarg`>.

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

`vsnprintf`

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_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 <`cstdarg`>.

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 *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
- **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.

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 <cstdarg>.

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 <cstdarg>.

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

```
------------------------
Number of tests run: 149
Number of tests passed: 141
Number of tests failed: 8
Coverage: 94.630875 %
------------------------

Negative tests

----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 432432423.342321321"
Result VSprintf: "Precision:  19, 432432423.3423213363"
Length VSprintf: 36
Result LwPRINTF: "Precision:  19,  432432423.342321336"
Length LwPRINTF: 36
Test result: Fail
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 0.0001234567"
Result VSprintf: "Precision:  19, 0.00012345669999999999"
Length VSprintf: 40
Result LwPRINTF: "Precision:  19,         0.0001234567"
Length LwPRINTF: 36
Test result: Fail
----
Format: "%.45f"
Params: "3.23321321"
Result VSprintf: "3.233213210000000170651901498786173760890960693"
Length VSprintf: 47
Result LwPRINTF: "3.233213210000000160000000000000000000000000000000"
Length LwPRINTF: 47
Test result: Fail
----
Format: "%.45F"
Params: "3.23321321"
Result VSprintf: "3.233213210000000170651901498786173760890960693"
Length VSprintf: 47
Result LwPRINTF: "3.233213210000000160000000000000000000000000000000"
Length LwPRINTF: 47
Test result: Fail
----
Format: "%22.33e"
Params: "123.456"
Result VSprintf: "1.234560000000000030695446184836328e+02"
Length VSprintf: 39
Result LwPRINTF: "1.234560000000000096000000000000000e+02"
Length LwPRINTF: 39
```

(continues on next page)

(continued from previous page)

```
Test result: Fail
----
Format: "%22.33e"
Params: "-123.456"
Result VSprintf: "-1.23456000000000003069544618483632 8e+02"
Length VSprintf: 40
Result LwPRINTF: "-1.23456000000000009600000000000000e+02"
Length LwPRINTF: 40
Test result: Fail
----
Format: "%22.33e"
Params: "0.123456"
Result VSprintf: "1.23455999999999996297184168270177 9e-01"
Length VSprintf: 39
Result LwPRINTF: "1.23455999999999987200000000000000e-01"
Length LwPRINTF: 39
Test result: Fail
----
Format: "%22.33e"
Params: "-0.123456"
Result VSprintf: "-1.23455999999999996297184168270177 9e-01"
Length VSprintf: 40
Result LwPRINTF: "-1.23455999999999987200000000000000e-01"
Length LwPRINTF: 40
Test result: Fail
------------------------

Positive tests

----
Format: "Precision: %3d, %.*g"
Params: "17, 17, 0.0001234567"
Result VSprintf: "Precision:  17, 0.0001234567"
Length VSprintf: 28
Result LwPRINTF: "Precision:  17, 0.0001234567"
Length LwPRINTF: 28
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 432432423.342321321"
Result VSprintf: "Precision:   0,                4e+08"
Length VSprintf: 36
Result LwPRINTF: "Precision:   0,                4e+08"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 432432423.342321321"
Result VSprintf: "Precision:   1,                4e+08"
Length VSprintf: 36
Result LwPRINTF: "Precision:   1,                4e+08"
Length LwPRINTF: 36
```

(continues on next page)

(continued from previous page)

```
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 432432423.342321321"
Result VSprintf: "Precision:   2,              4.3e+08"
Length VSprintf: 36
Result LwPRINTF: "Precision:   2,              4.3e+08"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 432432423.342321321"
Result VSprintf: "Precision:   3,             4.32e+08"
Length VSprintf: 36
Result LwPRINTF: "Precision:   3,             4.32e+08"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 432432423.342321321"
Result VSprintf: "Precision:   4,            4.324e+08"
Length VSprintf: 36
Result LwPRINTF: "Precision:   4,            4.324e+08"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 432432423.342321321"
Result VSprintf: "Precision:   5,           4.3243e+08"
Length VSprintf: 36
Result LwPRINTF: "Precision:   5,           4.3243e+08"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 432432423.342321321"
Result VSprintf: "Precision:   6,          4.32432e+08"
Length VSprintf: 36
Result LwPRINTF: "Precision:   6,          4.32432e+08"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 432432423.342321321"
Result VSprintf: "Precision:   7,         4.324324e+08"
Length VSprintf: 36
Result LwPRINTF: "Precision:   7,         4.324324e+08"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 432432423.342321321"
```

(continues on next page)

(continued from previous page)

```
Result VSprintf: "Precision:   8,        4.3243242e+08"
Length VSprintf: 36
Result LwPRINTF: "Precision:   8,        4.3243242e+08"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 432432423.342321321"
Result VSprintf: "Precision:   9,           432432423"
Length VSprintf: 36
Result LwPRINTF: "Precision:   9,           432432423"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 432432423.342321321"
Result VSprintf: "Precision:  10,         432432423.3"
Length VSprintf: 36
Result LwPRINTF: "Precision:  10,         432432423.3"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 432432423.342321321"
Result VSprintf: "Precision:  11,        432432423.34"
Length VSprintf: 36
Result LwPRINTF: "Precision:  11,        432432423.34"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 432432423.342321321"
Result VSprintf: "Precision:  12,        432432423.342"
Length VSprintf: 36
Result LwPRINTF: "Precision:  12,        432432423.342"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 432432423.342321321"
Result VSprintf: "Precision:  13,       432432423.3423"
Length VSprintf: 36
Result LwPRINTF: "Precision:  13,       432432423.3423"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 432432423.342321321"
Result VSprintf: "Precision:  14,      432432423.34232"
Length VSprintf: 36
Result LwPRINTF: "Precision:  14,      432432423.34232"
Length LwPRINTF: 36
```

(continues on next page)

(continued from previous page)

```
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 432432423.342321321"
Result VSprintf: "Precision:  15,     432432423.342321"
Length VSprintf: 36
Result LwPRINTF: "Precision:  15,     432432423.342321"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 432432423.342321321"
Result VSprintf: "Precision:  16,    432432423.3423213"
Length VSprintf: 36
Result LwPRINTF: "Precision:  16,    432432423.3423213"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 432432423.342321321"
Result VSprintf: "Precision:  17,   432432423.34232134"
Length VSprintf: 36
Result LwPRINTF: "Precision:  17,   432432423.34232134"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 432432423.342321321"
Result VSprintf: "Precision:  18,  432432423.342321336"
Length VSprintf: 36
Result LwPRINTF: "Precision:  18,  432432423.342321336"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 0.0001234567"
Result VSprintf: "Precision:   0,               0.0001"
Length VSprintf: 36
Result LwPRINTF: "Precision:   0,               0.0001"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 0.0001234567"
Result VSprintf: "Precision:   1,               0.0001"
Length VSprintf: 36
Result LwPRINTF: "Precision:   1,               0.0001"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 0.0001234567"
```

(continues on next page)

(continued from previous page)

```
Result VSprintf: "Precision:   2,              0.00012"
Length VSprintf: 36
Result LwPRINTF: "Precision:   2,              0.00012"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 0.0001234567"
Result VSprintf: "Precision:   3,             0.000123"
Length VSprintf: 36
Result LwPRINTF: "Precision:   3,             0.000123"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 0.0001234567"
Result VSprintf: "Precision:   4,            0.0001235"
Length VSprintf: 36
Result LwPRINTF: "Precision:   4,            0.0001235"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 0.0001234567"
Result VSprintf: "Precision:   5,           0.00012346"
Length VSprintf: 36
Result LwPRINTF: "Precision:   5,           0.00012346"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 0.0001234567"
Result VSprintf: "Precision:   6,          0.000123457"
Length VSprintf: 36
Result LwPRINTF: "Precision:   6,          0.000123457"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 0.0001234567"
Result VSprintf: "Precision:   7,         0.0001234567"
Length VSprintf: 36
Result LwPRINTF: "Precision:   7,         0.0001234567"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 0.0001234567"
Result VSprintf: "Precision:   8,         0.0001234567"
Length VSprintf: 36
Result LwPRINTF: "Precision:   8,         0.0001234567"
Length LwPRINTF: 36
```

(continues on next page)

(continued from previous page)

```
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 0.0001234567"
Result VSprintf: "Precision:   9,         0.0001234567"
Length VSprintf: 36
Result LwPRINTF: "Precision:   9,         0.0001234567"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 0.0001234567"
Result VSprintf: "Precision:  10,         0.0001234567"
Length VSprintf: 36
Result LwPRINTF: "Precision:  10,         0.0001234567"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 0.0001234567"
Result VSprintf: "Precision:  11,         0.0001234567"
Length VSprintf: 36
Result LwPRINTF: "Precision:  11,         0.0001234567"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 0.0001234567"
Result VSprintf: "Precision:  12,         0.0001234567"
Length VSprintf: 36
Result LwPRINTF: "Precision:  12,         0.0001234567"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 0.0001234567"
Result VSprintf: "Precision:  13,         0.0001234567"
Length VSprintf: 36
Result LwPRINTF: "Precision:  13,         0.0001234567"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 0.0001234567"
Result VSprintf: "Precision:  14,         0.0001234567"
Length VSprintf: 36
Result LwPRINTF: "Precision:  14,         0.0001234567"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 0.0001234567"
```

(continues on next page)

(continued from previous page)

```
Result VSprintf: "Precision:  15,         0.0001234567"
Length VSprintf: 36
Result LwPRINTF: "Precision:  15,         0.0001234567"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 0.0001234567"
Result VSprintf: "Precision:  16,         0.0001234567"
Length VSprintf: 36
Result LwPRINTF: "Precision:  16,         0.0001234567"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 0.0001234567"
Result VSprintf: "Precision:  17,         0.0001234567"
Length VSprintf: 36
Result LwPRINTF: "Precision:  17,         0.0001234567"
Length LwPRINTF: 36
Test result: Pass
----
Format: "Precision: %3d, %20.*g"
Params: "i, i, 0.0001234567"
Result VSprintf: "Precision:  18,         0.0001234567"
Length VSprintf: 36
Result LwPRINTF: "Precision:  18,         0.0001234567"
Length LwPRINTF: 36
Test result: Pass
----
Format: "%.4f"
Params: "3.23321321"
Result VSprintf: "3.2332"
Length VSprintf: 6
Result LwPRINTF: "3.2332"
Length LwPRINTF: 6
Test result: Pass
----
Format: "%.4F"
Params: "3.23321321"
Result VSprintf: "3.2332"
Length VSprintf: 6
Result LwPRINTF: "3.2332"
Length LwPRINTF: 6
Test result: Pass
----
Format: "%g"
Params: "1.23342"
Result VSprintf: "1.23342"
Length VSprintf: 7
Result LwPRINTF: "1.23342"
Length LwPRINTF: 7
```

(continues on next page)

(continued from previous page)

```
Test result: Pass
----
Format: "%g"
Params: "12334.2"
Result VSprintf: "12334.2"
Length VSprintf: 7
Result LwPRINTF: "12334.2"
Length LwPRINTF: 7
Test result: Pass
----
Format: "%.8g"
Params: "0.000000123342"
Result VSprintf: "1.23342e-07"
Length VSprintf: 11
Result LwPRINTF: "1.23342e-07"
Length LwPRINTF: 11
Test result: Pass
----
Format: "%.8G"
Params: "0.000000123342"
Result VSprintf: "1.23342E-07"
Length VSprintf: 11
Result LwPRINTF: "1.23342E-07"
Length LwPRINTF: 11
Test result: Pass
----
Format: "%.4f"
Params: "323243432432432.432"
Result VSprintf: "323243432432432.4375"
Length VSprintf: 20
Result LwPRINTF: "323243432432432.4375"
Length LwPRINTF: 20
Test result: Pass
----
Format: "%e"
Params: "-123.456"
Result VSprintf: "-1.234560e+02"
Length VSprintf: 13
Result LwPRINTF: "-1.234560e+02"
Length LwPRINTF: 13
Test result: Pass
----
Format: "%e"
Params: "0.000001"
Result VSprintf: "1.000000e-06"
Length VSprintf: 12
Result LwPRINTF: "1.000000e-06"
Length LwPRINTF: 12
Test result: Pass
----
Format: "%e"
Params: "0.123456"
```

(continues on next page)

(continued from previous page)

```
Result VSprintf: "1.234560e-01"
Length VSprintf: 12
Result LwPRINTF: "1.234560e-01"
Length LwPRINTF: 12
Test result: Pass
----
Format: "%e"
Params: "-0.123456"
Result VSprintf: "-1.234560e-01"
Length VSprintf: 13
Result LwPRINTF: "-1.234560e-01"
Length LwPRINTF: 13
Test result: Pass
----
Format: "%.4e"
Params: "123.456"
Result VSprintf: "1.2346e+02"
Length VSprintf: 10
Result LwPRINTF: "1.2346e+02"
Length LwPRINTF: 10
Test result: Pass
----
Format: "%.4e"
Params: "-123.456"
Result VSprintf: "-1.2346e+02"
Length VSprintf: 11
Result LwPRINTF: "-1.2346e+02"
Length LwPRINTF: 11
Test result: Pass
----
Format: "%.4e"
Params: "0.123456"
Result VSprintf: "1.2346e-01"
Length VSprintf: 10
Result LwPRINTF: "1.2346e-01"
Length LwPRINTF: 10
Test result: Pass
----
Format: "%.4e"
Params: "-0.123456"
Result VSprintf: "-1.2346e-01"
Length VSprintf: 11
Result LwPRINTF: "-1.2346e-01"
Length LwPRINTF: 11
Test result: Pass
----
Format: "%.0e"
Params: "123.456"
Result VSprintf: "1e+02"
Length VSprintf: 5
Result LwPRINTF: "1e+02"
Length LwPRINTF: 5
```

(continues on next page)

(continued from previous page)

```
Test result: Pass
----
Format: "%.0e"
Params: "-123.456"
Result VSprintf: "-1e+02"
Length VSprintf: 6
Result LwPRINTF: "-1e+02"
Length LwPRINTF: 6
Test result: Pass
----
Format: "%.0e"
Params: "0.123456"
Result VSprintf: "1e-01"
Length VSprintf: 5
Result LwPRINTF: "1e-01"
Length LwPRINTF: 5
Test result: Pass
----
Format: "%.0e"
Params: "-0.123456"
Result VSprintf: "-1e-01"
Length VSprintf: 6
Result LwPRINTF: "-1e-01"
Length LwPRINTF: 6
Test result: Pass
----
Format: "%22.4e"
Params: "123.456"
Result VSprintf: "            1.2346e+02"
Length VSprintf: 22
Result LwPRINTF: "            1.2346e+02"
Length LwPRINTF: 22
Test result: Pass
----
Format: "%22.4e"
Params: "-123.456"
Result VSprintf: "           -1.2346e+02"
Length VSprintf: 22
Result LwPRINTF: "           -1.2346e+02"
Length LwPRINTF: 22
Test result: Pass
----
Format: "%22.4e"
Params: "0.123456"
Result VSprintf: "            1.2346e-01"
Length VSprintf: 22
Result LwPRINTF: "            1.2346e-01"
Length LwPRINTF: 22
Test result: Pass
----
Format: "%22.4e"
Params: "-0.123456"
```

(continues on next page)

(continued from previous page)

```
Result VSprintf: "           -1.2346e-01"
Length VSprintf: 22
Result LwPRINTF: "           -1.2346e-01"
Length LwPRINTF: 22
Test result: Pass
----
Format: "%022.4e"
Params: "123.456"
Result VSprintf: "0000000000001.2346e+02"
Length VSprintf: 22
Result LwPRINTF: "0000000000001.2346e+02"
Length LwPRINTF: 22
Test result: Pass
----
Format: "%022.4e"
Params: "-123.456"
Result VSprintf: "-000000000001.2346e+02"
Length VSprintf: 22
Result LwPRINTF: "-000000000001.2346e+02"
Length LwPRINTF: 22
Test result: Pass
----
Format: "%022.4e"
Params: "0.123456"
Result VSprintf: "0000000000001.2346e-01"
Length VSprintf: 22
Result LwPRINTF: "0000000000001.2346e-01"
Length LwPRINTF: 22
Test result: Pass
----
Format: "%e"
Params: "0.00000000123456"
Result VSprintf: "1.234560e-09"
Length VSprintf: 12
Result LwPRINTF: "1.234560e-09"
Length LwPRINTF: 12
Test result: Pass
----
Format: "%022.4e"
Params: "-0.123456"
Result VSprintf: "-000000000001.2346e-01"
Length VSprintf: 22
Result LwPRINTF: "-000000000001.2346e-01"
Length LwPRINTF: 22
Test result: Pass
----
Format: "%.4E"
Params: "-123.456"
Result VSprintf: "-1.2346E+02"
Length VSprintf: 11
Result LwPRINTF: "-1.2346E+02"
Length LwPRINTF: 11
```

(continues on next page)

(continued from previous page)

```
Test result: Pass
----
Format: "% 3u"
Params: "(unsigned)28"
Result VSprintf: " 28"
Length VSprintf: 3
Result LwPRINTF: " 28"
Length LwPRINTF: 3
Test result: Pass
----
Format: "% 3u"
Params: "(unsigned)123456"
Result VSprintf: "123456"
Length VSprintf: 6
Result LwPRINTF: "123456"
Length LwPRINTF: 6
Test result: Pass
----
Format: "%03d"
Params: "28"
Result VSprintf: "028"
Length VSprintf: 3
Result LwPRINTF: "028"
Length LwPRINTF: 3
Test result: Pass
----
Format: "%+03d"
Params: "28"
Result VSprintf: "+28"
Length VSprintf: 3
Result LwPRINTF: "+28"
Length LwPRINTF: 3
Test result: Pass
----
Format: "%+3d"
Params: "28"
Result VSprintf: "+28"
Length VSprintf: 3
Result LwPRINTF: "+28"
Length LwPRINTF: 3
Test result: Pass
----
Format: "%03d"
Params: "-28"
Result VSprintf: "-28"
Length VSprintf: 3
Result LwPRINTF: "-28"
Length LwPRINTF: 3
Test result: Pass
----
Format: "%+03d"
Params: "-28"
```

(continues on next page)

(continued from previous page)

```
Result VSprintf: "-28"
Length VSprintf: 3
Result LwPRINTF: "-28"
Length LwPRINTF: 3
Test result: Pass
----
Format: "%+3d"
Params: "-28"
Result VSprintf: "-28"
Length VSprintf: 3
Result LwPRINTF: "-28"
Length LwPRINTF: 3
Test result: Pass
----
Format: "%03u"
Params: "(unsigned)123456"
Result VSprintf: "123456"
Length VSprintf: 6
Result LwPRINTF: "123456"
Length LwPRINTF: 6
Test result: Pass
----
Format: "%-010uabc"
Params: "(unsigned)123456"
Result VSprintf: "123456    abc"
Length VSprintf: 13
Result LwPRINTF: "123456    abc"
Length LwPRINTF: 13
Test result: Pass
----
Format: "%010uabc"
Params: "(unsigned)123456"
Result VSprintf: "0000123456abc"
Length VSprintf: 13
Result LwPRINTF: "0000123456abc"
Length LwPRINTF: 13
Test result: Pass
----
Format: "%-10d"
Params: "-123"
Result VSprintf: "-123      "
Length VSprintf: 10
Result LwPRINTF: "-123      "
Length LwPRINTF: 10
Test result: Pass
----
Format: "%10d"
Params: "-123"
Result VSprintf: "      -123"
Length VSprintf: 10
Result LwPRINTF: "      -123"
Length LwPRINTF: 10
```

(continues on next page)

(continued from previous page)

```
Test result: Pass
----
Format: "%-06d"
Params: "-1234567"
Result VSprintf: "-1234567"
Length VSprintf: 8
Result LwPRINTF: "-1234567"
Length LwPRINTF: 8
Test result: Pass
----
Format: "%06d"
Params: "-1234567"
Result VSprintf: "-1234567"
Length VSprintf: 8
Result LwPRINTF: "-1234567"
Length LwPRINTF: 8
Test result: Pass
----
Format: "%-10d"
Params: "-1234567"
Result VSprintf: "-1234567  "
Length VSprintf: 10
Result LwPRINTF: "-1234567  "
Length LwPRINTF: 10
Test result: Pass
----
Format: "%10d"
Params: "-1234567"
Result VSprintf: "  -1234567"
Length VSprintf: 10
Result LwPRINTF: "  -1234567"
Length LwPRINTF: 10
Test result: Pass
----
Format: "%-010d"
Params: "-1234567"
Result VSprintf: "-1234567  "
Length VSprintf: 10
Result LwPRINTF: "-1234567  "
Length LwPRINTF: 10
Test result: Pass
----
Format: "%010d"
Params: "-1234567"
Result VSprintf: "-001234567"
Length VSprintf: 10
Result LwPRINTF: "-001234567"
Length LwPRINTF: 10
Test result: Pass
----
Format: "%s"
Params: ""This is my string""
```

(continues on next page)

(continued from previous page)

```
Result VSprintf: "This is my string"
Length VSprintf: 17
Result LwPRINTF: "This is my string"
Length LwPRINTF: 17
Test result: Pass
----
Format: "%10s"
Params: ""This is my string""
Result VSprintf: "This is my string"
Length VSprintf: 17
Result LwPRINTF: "This is my string"
Length LwPRINTF: 17
Test result: Pass
----
Format: "%0*d"
Params: "10, -123"
Result VSprintf: "-000000123"
Length VSprintf: 10
Result LwPRINTF: "-000000123"
Length LwPRINTF: 10
Test result: Pass
----
Format: "%zu"
Params: "(size_t)10"
Result VSprintf: "10"
Length VSprintf: 2
Result LwPRINTF: "10"
Length LwPRINTF: 2
Test result: Pass
----
Format: "%ju"
Params: "(uintmax_t)10"
Result VSprintf: "10"
Length VSprintf: 2
Result LwPRINTF: "10"
Length LwPRINTF: 2
Test result: Pass
----
Format: "% d"
Params: "1024"
Result VSprintf: " 1024"
Length VSprintf: 5
Result LwPRINTF: " 1024"
Length LwPRINTF: 5
Test result: Pass
----
Format: "% 4d"
Params: "1024"
Result VSprintf: " 1024"
Length VSprintf: 5
Result LwPRINTF: " 1024"
Length LwPRINTF: 5
```

(continues on next page)

(continued from previous page)

```
Test result: Pass
----
Format: "% 3d"
Params: "1024"
Result VSprintf: " 1024"
Length VSprintf: 5
Result LwPRINTF: " 1024"
Length LwPRINTF: 5
Test result: Pass
----
Format: "% 3f"
Params: "32.687"
Result VSprintf: " 32.687000"
Length VSprintf: 10
Result LwPRINTF: " 32.687000"
Length LwPRINTF: 10
Test result: Pass
----
Format: "%*.*s"
Params: "8, 12, "This is my string""
Result VSprintf: "This is my s"
Length VSprintf: 12
Result LwPRINTF: "This is my s"
Length LwPRINTF: 12
Test result: Pass
----
Format: "%*.*s"
Params: "8, 12, "Stri""
Result VSprintf: "    Stri"
Length VSprintf: 8
Result LwPRINTF: "    Stri"
Length LwPRINTF: 8
Test result: Pass
----
Format: "%-6.10s"
Params: ""This is my string""
Result VSprintf: "This is my"
Length VSprintf: 10
Result LwPRINTF: "This is my"
Length LwPRINTF: 10
Test result: Pass
----
Format: "%6.10s"
Params: ""This is my string""
Result VSprintf: "This is my"
Length VSprintf: 10
Result LwPRINTF: "This is my"
Length LwPRINTF: 10
Test result: Pass
----
Format: "%-6.10s"
Params: ""This is my string""
```

(continues on next page)

(continued from previous page)

```
Result VSprintf: "This is my"
Length VSprintf: 10
Result LwPRINTF: "This is my"
Length LwPRINTF: 10
Test result: Pass
----
Format: "%6.10s"
Params: ""Th""
Result VSprintf: "    Th"
Length VSprintf: 6
Result LwPRINTF: "    Th"
Length LwPRINTF: 6
Test result: Pass
----
Format: "%-6.10s"
Params: ""Th""
Result VSprintf: "Th    "
Length VSprintf: 6
Result LwPRINTF: "Th    "
Length LwPRINTF: 6
Test result: Pass
----
Format: "%*.*s"
Params: "-6, 10, "Th""
Result VSprintf: "Th    "
Length VSprintf: 6
Result LwPRINTF: "Th    "
Length LwPRINTF: 6
Test result: Pass
----
Format: "%*.*s"
Params: "6, 10, "Th""
Result VSprintf: "    Th"
Length VSprintf: 6
Result LwPRINTF: "    Th"
Length LwPRINTF: 6
Test result: Pass
----
Format: "%.4s"
Params: ""This is my string""
Result VSprintf: "This"
Length VSprintf: 4
Result LwPRINTF: "This"
Length LwPRINTF: 4
Test result: Pass
----
Format: "%.6s"
Params: ""1234""
Result VSprintf: "1234"
Length VSprintf: 4
Result LwPRINTF: "1234"
Length LwPRINTF: 4
```

(continues on next page)

(continued from previous page)

```
Test result: Pass
----
Format: "%.4s"
Params: ""stri""
Result VSprintf: "stri"
Length VSprintf: 4
Result LwPRINTF: "stri"
Length LwPRINTF: 4
Test result: Pass
----
Format: "%.4s%.2s"
Params: ""123456", "abcdef""
Result VSprintf: "1234ab"
Length VSprintf: 6
Result LwPRINTF: "1234ab"
Length LwPRINTF: 6
Test result: Pass
----
Format: "%.4.2s"
Params: ""123456""
Result VSprintf: ".2s"
Length VSprintf: 3
Result LwPRINTF: ".2s"
Length LwPRINTF: 3
Test result: Pass
----
Format: "%.*s"
Params: "3, "123456""
Result VSprintf: "123"
Length VSprintf: 3
Result LwPRINTF: "123"
Length LwPRINTF: 3
Test result: Pass
----
Format: "%.3s"
Params: """"
Result VSprintf: ""
Length VSprintf: 0
Result LwPRINTF: ""
Length LwPRINTF: 0
Test result: Pass
----
Format: "%yunknown"
Params: """"
Result VSprintf: "yunknown"
Length VSprintf: 8
Result LwPRINTF: "yunknown"
Length LwPRINTF: 8
Test result: Pass
----
Format: "%#2X"
Params: "123"
```

(continues on next page)

(continued from previous page)

```
Result VSprintf: "0X7B"
Length VSprintf: 4
Result LwPRINTF: "0X7B"
Length LwPRINTF: 4
Test result: Pass
----
Format: "%#2x"
Params: "123"
Result VSprintf: "0x7b"
Length VSprintf: 4
Result LwPRINTF: "0x7b"
Length LwPRINTF: 4
Test result: Pass
----
Format: "%#2o"
Params: "123"
Result VSprintf: "0173"
Length VSprintf: 4
Result LwPRINTF: "0173"
Length LwPRINTF: 4
Test result: Pass
----
Format: "%#2X"
Params: "1"
Result VSprintf: "0X1"
Length VSprintf: 3
Result LwPRINTF: "0X1"
Length LwPRINTF: 3
Test result: Pass
----
Format: "%#2x"
Params: "1"
Result VSprintf: "0x1"
Length VSprintf: 3
Result LwPRINTF: "0x1"
Length LwPRINTF: 3
Test result: Pass
----
Format: "%#2o"
Params: "1"
Result VSprintf: "01"
Length VSprintf: 2
Result LwPRINTF: "01"
Length LwPRINTF: 2
Test result: Pass
----
Format: "%#2X"
Params: "0"
Result VSprintf: " 0"
Length VSprintf: 2
Result LwPRINTF: " 0"
Length LwPRINTF: 2
```

(continues on next page)

(continued from previous page)

```
Test result: Pass
----
Format: "%#2x"
Params: "0"
Result VSprintf: " 0"
Length VSprintf: 2
Result LwPRINTF: " 0"
Length LwPRINTF: 2
Test result: Pass
----
Format: "%#2o"
Params: "0"
Result VSprintf: " 0"
Length VSprintf: 2
Result LwPRINTF: " 0"
Length LwPRINTF: 2
Test result: Pass
----
Format: "%p"
Params: "&tests_passed"
Result VSprintf: "00BE0FD4"
Length VSprintf: 8
Result LwPRINTF: "00BE0FD4"
Length LwPRINTF: 8
Test result: Pass
----
Format: "0X%p"
Params: "&tests_passed"
Result VSprintf: "0X00BE0FD4"
Length VSprintf: 10
Result LwPRINTF: "0X00BE0FD4"
Length LwPRINTF: 10
Test result: Pass
----
Format: "0x%p"
Params: "&tests_passed"
Result VSprintf: "0x00BE0FD4"
Length VSprintf: 10
Result LwPRINTF: "0x00BE0FD4"
Length LwPRINTF: 10
Test result: Pass
----
Format: "%llb abc"
Params: "123"
Result expected: "1111011 abc"
Length expected: 11
Result LwPRINTF: "1111011 abc"
Length LwPRINTF: 11
Test result: Pass
----
Format: "%b"
Params: "4"
```

(continues on next page)

(continued from previous page)

```
Result expected: "100"
Length expected: 3
Result LwPRINTF: "100"
Length LwPRINTF: 3
Test result: Pass
----
Format: "%#2B"
Params: "1"
Result expected: "0B1"
Length expected: 3
Result LwPRINTF: "0B1"
Length LwPRINTF: 3
Test result: Pass
----
Format: "%#2b"
Params: "1"
Result expected: "0b1"
Length expected: 3
Result LwPRINTF: "0b1"
Length LwPRINTF: 3
Test result: Pass
----
Format: "%#2B"
Params: "0"
Result expected: " 0"
Length expected: 2
Result LwPRINTF: " 0"
Length LwPRINTF: 2
Test result: Pass
----
Format: "%#2b"
Params: "0"
Result expected: " 0"
Length expected: 2
Result LwPRINTF: " 0"
Length LwPRINTF: 2
Test result: Pass
----
Format: "%#B"
Params: "0"
Result expected: "0"
Length expected: 1
Result LwPRINTF: "0"
Length LwPRINTF: 1
Test result: Pass
----
Format: "%#b"
Params: "0"
Result expected: "0"
Length expected: 1
Result LwPRINTF: "0"
Length LwPRINTF: 1
```

(continues on next page)

(continued from previous page)

```
Test result: Pass
----
Format: "%#B"
Params: "6"
Result expected: "0B110"
Length expected: 5
Result LwPRINTF: "0B110"
Length LwPRINTF: 5
Test result: Pass
----
Format: "%#b"
Params: "6"
Result expected: "0b110"
Length expected: 5
Result LwPRINTF: "0b110"
Length LwPRINTF: 5
Test result: Pass
----
Format: "%5K"
Params: "my_arr"
Result expected: "0102B5C6D7"
Length expected: 10
Result LwPRINTF: "0102B5C6D7"
Length LwPRINTF: 10
Test result: Pass
----
Format: "%*K"
Params: "3, my_arr"
Result expected: "0102B5"
Length expected: 6
Result LwPRINTF: "0102B5"
Length LwPRINTF: 6
Test result: Pass
----
Format: "% *K"
Params: "3, my_arr"
Result expected: "01 02 B5"
Length expected: 8
Result LwPRINTF: "01 02 B5"
Length LwPRINTF: 8
Test result: Pass
----
Format: "%5k"
Params: "my_arr"
Result expected: "0102b5c6d7"
Length expected: 10
Result LwPRINTF: "0102b5c6d7"
Length LwPRINTF: 10
Test result: Pass
----
Format: "%*k"
Params: "3, my_arr"
```

(continues on next page)

(continued from previous page)

```
Result expected: "0102b5"
Length expected: 6
Result LwPRINTF: "0102b5"
Length LwPRINTF: 6
Test result: Pass
----
Format: "% *k"
Params: "3, my_arr"
Result expected: "01 02 b5"
Length expected: 8
Result LwPRINTF: "01 02 b5"
Length LwPRINTF: 8
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 confiuration 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

## 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
```

(continues on next page)

(continued from previous page)

```
- 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

## 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>
Brian <bayuan@purdue.edu>
Peter Maxwell Warasila <madmaxwell@soundcomesout.com>
```

INDEX