SEGGER
Runtime Library

User Guide & Reference Manual

Document: UM12007
Software Version: 2.10
Revision: 0
Date: March 22, 2019

A product of SEGGER Microcontroller GmbH

www.segger.com

Disclaimer

Specifications written in this document are believed to be accurate, but are not guaranteed to be entirely free of error. The information in this manual is subject to change for functional or performance improvements without notice. Please make sure your manual is the latest edition. While the information herein is assumed to be accurate, SEGGER Microcontroller GmbH (SEGGER) assumes no responsibility for any errors or omissions. SEGGER makes and you receive no warranties or conditions, express, implied, statutory or in any communication with you. SEGGER specifically disclaims any implied warranty of merchantability or fitness for a particular purpose.

Copyright notice

Trademarks

Names mentioned in this manual may be trademarks of their respective companies.

Brand and product names are trademarks or registered trademarks of their respective holders.

Contact address

SEGGER Microcontroller GmbH

Ecolab-Allee 5
D-40789 Monheim am Rhein

Germany

Tel.	+49 2173-99312-0
Fax.	+49 2173-99312-28
E-mail:	support@segger.com*
Internet:	www.segger.com

*By sending us an email your (personal) data will automatically be processed. For further information please refer to our privacy policy which is available at https://www.segger.com/legal/privacy-policy/.

Manual versions

This manual describes the current software version. If you find an error in the manual or a problem in the software, please inform us and we will try to assist you as soon as possible. Contact us for further information on topics or functions that are not yet documented.

Print date: March 22, 2019

Software	Revision	Date	By	Description
2.10	0	190307	PC	Release version.
1.00	0	190204	PC	Internal version.

About this document

Assumptions

This document assumes that you already have a solid knowledge of the following:

- The software tools used for building your application (assembler, linker, C compiler).
- The C programming language.
- The target processor.
- DOS command line.

If you feel that your knowledge of C is not sufficient, we recommend *The C Programming Language* by Kernighan and Richie (ISBN 0--13--1103628), which describes the standard in C programming and, in newer editions, also covers the ANSI C standard.

How to use this manual

This manual explains all the functions and macros that the product offers. It assumes you have a working knowledge of the C language. Knowledge of assembly programming is not required.

Typographic conventions for syntax

This manual uses the following typographic conventions:

Style	Used for
Body	Body text.
`Parameter`	Parameters in API functions.
`Sample`	Sample code in program examples.
`Sample comment`	Comments in program examples.
`User Input`	Text entered at the keyboard by a user in a session transcript.
`Secret Input`	Text entered at the keyboard by a user, but not echoed (e.g. password entry), in a session transcript.
Reference	Reference to chapters, sections, tables and figures.
Emphasis	Very important sections.
SEGGER home page	A hyperlink to an external document or web site.

Table of contents

Chapter 1

Introduction

This section presents an overview of SEGGER Runtime Library, its structure, and its capabilities.

1.1 What is the SEGGER Runtime Library?

SEGGER Runtime Library is an optimized C library for Arm processors.

1.2 Features

SEGGER Runtime Library is written in standard ANSI C and Arm assembly language and can run on any Arm CPU. Here’s a list summarising the main features of SEGGER Runtime Library:

- Clean ISO/ANSI C source code.
- Fast assembly language floating point support.
- Conforms to the standard runtime ABI for the Arm architecture.
- Simple configuration.
- Royalty free.

1.3 Recommended project structure

We recommend keeping SEGGER Runtime Library separate from your application files. It is good practice to keep all the program files (including the header files) together in the `LIB` subdirectory of your project's root directory. This practice has the advantage of being very easy to update to newer versions of SEGGER Runtime Library by simply replacing the `LIB` directory. Your application files can be stored anywhere.

> **Note**
>
> When updating to a newer SEGGER Runtime Library version: as files may have been added, moved or deleted, the project directories may need to be updated accordingly.

1.4 Package content

SEGGER Runtime Library is provided in source code and contains everything needed. The following table shows the content of the SEGGER Runtime Library Package:

Directory	Description
Doc	SEGGER Runtime Library documentation.
LIB	SEGGER Runtime Library source code.

1.4.1 Include directories

You should make sure that the system include path contains the following directory:

- LIB

> **Note**
>
> Always make sure that you have only one version of each file!

It is frequently a major problem when updating to a new version of SEGGER Runtime Library if you have old files included and therefore mix different versions. If you keep SEGGER Runtime Library in the directories as suggested (and only in these), this type of problem cannot occur. When updating to a newer version, you should be able to keep your configuration files and leave them unchanged. For safety reasons, we recommend backing up (or at least renaming) the LIB directories before to updating.

Chapter 2

Compiling the SEGGER Runtime Library

2.1 User-facing source files

The standard C library is exposed to the user by a set of header files that provide an interface to the library. In addition, there must be additional "invisible" functions added to provide C language support, such as software floating point and integer mathematics, that the C compiler calls.

The user-facing interface files are:

File	Description
assert.h	Assertion macros.
ctype.h	Character classification functions.
errno.h	Access to errno.
float.h	Parameterization of floating types.
inttype.h	Parameterization of formatting of integer types.
iso646.h	Alternative spelling of C operators.
limits.h	Minima and maxima of floating and integer types.
locale.h	Functions for internationalizing software.
setjmp.h	Non-local jumps.
stdbool.h	Boolean type and values.
stddef.h	Standard definitions such as NULL.
stdint.h	Specification of fixed-size integer types.
stdio.h	Formatted input and output functions.
stdlib.h	Standardized common library functions.
string.h	String and memory functions.
time.h	Time and date functions.
wchar.h	Wide character functions.
wctype.h	Wide character classification functions.
xlocale.h	Extended POSIX.1 locale functions.

In addition some private header files are required:

File	Description
__argtype.h	Support for non-local formatted I/O.
__codesets.h	Definition of code sets.
__libc.h	General definitions used when compiling SEGGER Runtime Library.
__locales.h	Definition of locales.
__vfprintf.h	Support for formatted I/O.

2.2 Implementation source files

SEGGER Runtime Library is delivered in a small number of files that must be added to your project before building:

File	Description
atomicops.c	Support for atomic operations for the GNU GCC compiler.
basicops.c	Support for simple I/O operations e.g. putc.
codesets.c	Support for code pages used in locales
convops.c	Support for conversion between binary and printable strings.
ctype_no_wchar.c	Support for <ctype.h> without wide characters.
errno.c	Support for errno in a tasking environment.
errno_no_thread.c	Support for errno in a non-tasking environment.
execops.c	Support for execution control functions e.g. atexit().
floatasmops.s	Support for low-level floating point functions.
floatops.c	Support for high-level floating point functions.
heapops.c	Support for dynamic memory functions e.g. malloc().
intasmops.s	Support for low-level integer functions.
intops.c	Support for high-level integer functions e.g. ldiv().
jumpasmops.s	Support for nonlocal 'goto' functions e.g. longjmp.
locales.c	Support for various locales.
mbops.c	Support for multi-byte functions e.g. mbtowc().
prinops.c	Support for formatted output functions e.g. printf().
scanops.c	Support for formatted input functions e.g. scanf().
sortops.c	Support for searching and sorting functions e.g. qsort().
strasmops.s	Support for fast string and memory functions e.g. strcpy().
strops.c	Support for string and memory functions e.g. strcat().
timeops.c	Support for time operations e.g. mktime().
utilops.c	Support for SEGGER Runtime Library framework.
wprinops.c	Support for wide formatted output functions e.g. wprintf().
wscanops.c	Support for wide formatted input functions e.g. wscanf().
wstrops.c	Support for wide string functions e.g. wcscpy().

2.3 Configuring the library

All source files should be added to the project and the following preprocessor symbols set correctly to select the particular variant of the library:

Symbol	Description
__thumb__	Assemble targeting the Thumb instruction set (as opposed to ARM).
__thumb2__	Assemble targeting the Thumb-2 instruction set.
__ARCH_V3	Assemble targeting ARMv3.
__ARCH_V4	Assemble targeting ARMv4.
__ARCH_V4T	Assemble targeting ARMv4T.
__ARCH_V5TE	Assemble targeting for ARMv5TE.
__ARCH_V6M	Assemble targeting for ARMv6-M.
__ARCH_V7M	Assemble targeting for ARMv7-M.
__ARCH_V7EM	Assemble targeting for ARMv7E-M.
__FP_ABI_HARD__	Assemble targeting the hard floating point ABI.
__FPU_VFP__	Assemble targeting the vector FPU.
__FPV4_SP_D16__	Assemble targeting the single-precision v4 FPU.
__FPV5_SP_D16__	Assemble targeting the single-precision v5 FPU.
__OPTIMIZATION_SMALL	Prefer size-optimized code rather than larger, speed-optimized.
__LITTLE_ENDIAN	Target little-endian processors.
__BIG_ENDIAN	Target big-endian processors.
__SIZEOF_WCHAR_T	Select size of wchar_t type.
HEAP_SIZE	The size of the heap, in bytes.
ATEXIT_COUNT	The maximum number of registered atexit() functions.
SUPPORT_CHAR_CLASS	Support character classes in scanf() functions.
SUPPORT_INT	Support int in printf() and scanf() functions.
SUPPORT_LONG	Support long in printf() and scanf() functions.
SUPPORT_LONG_LONG	Support long long in printf() and scanf() functions.
SUPPORT_FLOAT	Support float in printf() and scanf() functions.
SUPPORT_WIDTH_PRECISION	Support width and precision in printf() and scanf() functions.
SUPPORT_WCHAR	Support wide character output in printf() and scanf() functions.

2.3.1 The SEGGER Runtime Library configuration file

The configuration of SEGGER Runtime Library is defined by the content of `__libc_conf.h` which is included by all C and assembly language source files.

2.3.2 __thumb__

This preprocessor symbol is usually defined by the compiler.

This preprocessor symbol must be defined when:

- compiling and assembling the library for targets that implement only the Thumb-2 instruction set
- compiling and assembling the library for the Thumb instruction set on targets that implement both the Arm and thumb instruction sets.

2.3.3 __thumb2__

This preprocessor symbol is usually defined by the compiler; it must be defined in order to compile and assemble for the Thumb-2 instruction set.

2.3.4 __ARCH_V3

This preprocessor symbol must be defined when compiling and assembling the library for the ARMv3 architecture.

2.3.5 __ARCH_V4

This preprocessor symbol must be defined when compiling and assembling the library for the ARMv4 architecture.

2.3.6 __ARCH_V4T

This preprocessor symbol must be defined when compiling and assembling the library for the ARMv4T architecture.

2.3.7 __ARCH_V5TE

This preprocessor symbol must be defined when compiling and assembling the library for the ARMv5TE architecture.

2.3.8 __ARCH_V6M

This preprocessor symbol must be defined when compiling and assembling the library for the ARMv6-M architecture.

2.3.9 __ARCH_V7M

This preprocessor symbol must be defined when compiling and assembling the library for the ARMv7-M architecture.

2.3.10 __ARCH_V7EM

This preprocessor symbol must be defined when compiling and assembling the library for the ARMv7-EM architecture.

2.3.11 __FP_ABI_HARD__

Define this preprocessor symbol when targeting the hard-floating ABI.

2.3.12 __FPV4_SP_D16__

Define this preprocessor symbol when targeting the single-precision v4 FPU.

2.3.13 __FPV5_SP_D16__

Define this preprocessor symbol when targeting the single-precision v5 FPU.

2.3.14 __OPTIMIZATION_SMALL

Define the preprocessor symbol `__OPTIMIZATION_SMALL` to select size-optimized implementations for both C and assembly language code.

If this preprocessor symbol is undefined (the default) the library is configured to select speed-optimized implementations without regard to code size.

2.3.15 __LITTLE_ENDIAN, __BIG_ENDIAN

Only one of these two preprocessor symbols must be defined. Please select the preprocessor symbol corresponding to the byte order of your target execution environment.

2.3.16 __SIZEOF_WCHAR_T

This preprocessor symbol must be set to the underlying size of `wchar_t`. It is possible to select a specific implementation of `wchar_t` using command line switches, ensure that `__SIZEOF_CHAR_T` correctly reflects the size of the type.

Typical implementations of `wchar_t` are 16-bit and 32-bit quantities corresponding to UCS-2 and UCS-4 code points.

2.3.17 HEAP_SIZE

The HEAP_SIZE preprocessor symbol sets the size of the heap, in bytes, available to the application.

2.3.18 SUPPORT_INT, ..., SUPPORT_LONG_LONG

To select the level of `printf()` and `scanf()` support, set one of these preprocessor symbols to 1 and the others to 0 in your project, e.g.

```
#define SUPPORT_INT        0
#define SUPPORT_LONG       0
#define SUPPORT_LONG_LONG  1
```


2.3.19 SUPPORT_FLOAT

Set this preprocessor symbol to 1 to include floating-point support in `printf()` and `scanf()`. Set this preprocessor symbol to 0 to eliminate floating-point input-output support.

Chapter 3

Runtime support

This section describes how to set up the execution environment for the C library.

3.1 Getting to main() and then exit()

Before entering `main()` the execution environment must be set up such that the C standard library will function correctly.

This section does not describe the compiler or linker support for placing code and data into memory, how to configure any RAM, or how to zero memory required for zero-initialized data. For this, please refer to your toolset compiler and linker documentation.

Nor does this section document how to call constructors and destructors in the correct order. Again, refer to your toolset manuals.

3.1.1 At-exit function support

If your application requires support for `atexit()` and for those functions to be executed, you must set the preprocessor symbol `ATEXIT_COUNT` to a nonzero value.

Once `ATEXIT_COUNT` is set nonzero, `atexit()` support is compiled into the library.

After returning from `main()` or by calling `exit()`, any registered `atexit` functions must be called to close down. To do this, call `_execute_at_exit_fns()` from the runtime startup that invoked `main()`.

3.2 Input and output

3.2.1 Input

For formatted input using `scanf()` and for functions such as `gets()` the library requires the user to implement the function `__getchar()`:

```
int __getchar(void);
```

The return value should be EOF an error receiving a character, and in the case of success the code of the character received.

3.2.2 Output

For formatted output using `printf()` and for functions such as `puts()` the library requires the user to implement the function `__putchar()`:

```
int __putchar(int ch, __printf_tag_ptr context);
```

The first parameter is the character to write to standard output. The second parameter is a pointer to an internal context that can safely be ignored.

The return value should be EOF an error or non-negative to indicate success.

