SEGGER
Runtime Library

User Guide & Reference Manual

Document: UM12007
Software Version: 2.10
Revision: 0
Date: May 5, 2020

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: May 5, 2020

Software	Revision	Date	By	Description
2.12	0	191220	PC	Chapter "C library API" • Added expm1f().
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

1 Introduction 16

1.1 What is the SEGGER Runtime Library? 17
1.2 Features 17
1.3 Recommended project structure 18
1.4 Package content 19
1.4.1 Include directories 19

2 Compiling the SEGGER Runtime Library 20

2.1 User-facing source files 21
2.2 Implementation source files 22
2.3 Configuring the library 23
2.3.1 The SEGGER Runtime Library configuration file 25
2.3.2 __SEGGER_RTL_BYTE_ORDER 26
2.3.3 __SEGGER_RTL_SIZEOF_LONG 27
2.3.4 __SEGGER_RTL_SIZEOF_PTR 28
2.3.5 __SEGGER_RTL_SIZEOF_WCHAR_T 29
2.3.6 __SEGGER_RTL_OPTIMIZE 30
2.3.7 __SEGGER_RTL_HEAP_SIZE 31
2.3.8 __SEGGER_RTL_FORMAT_INT_WIDTH 32
2.3.9 __SEGGER_RTL_FORMAT_FLOAT_WIDTH 33
2.3.10 __SEGGER_RTL_FORMAT_WCHAR 34
2.3.11 __SEGGER_RTL_FORMAT_CHAR_CLASS 35
2.3.12 __SEGGER_RTL_FORMAT_WIDTH_PRECISION 36
2.3.13 __SEGGER_RTL_STDOUT_BUFFER_LEN 37
2.3.14 __SEGGER_RTL_THREAD 38

3 Runtime support 39

3.1 Getting to main() and then exit() 40
3.1.1 At-exit function support 40
3.2 Input and output 41
3.2.1 Input 41
3.2.2 Output 41

4 C library API 42

4.1 <assert.h> 43
4.1.1 Assertion functions 43
4.1.1.1 assert 44
4.1.1.2 __SEGGER_RTL_X_assert 45
4.2 <ctype.h> 46

4.2.1 Classification functions 46
4.2.1.1 iscntrl 47
4.2.1.2 iscntrl_l 48
4.2.1.3 isblank 49
4.2.1.4 isblank_l 50
4.2.1.5 isspace 51
4.2.1.6 isspace_l 52
4.2.1.7 ispunct 53
4.2.1.8 ispunct_l 54
4.2.1.9 isdigit 55
4.2.1.10 isdigit_l 56
4.2.1.11 isxdigit 57
4.2.1.12 isxdigit_l 58
4.2.1.13 isalpha 59
4.2.1.14 isalpha_l 60
4.2.1.15 isalnum 61
4.2.1.16 isalnum_l 62
4.2.1.17 isupper 63
4.2.1.18 isupper_l 64
4.2.1.19 islower 65
4.2.1.20 islower_l 66
4.2.1.21 isprint 67
4.2.1.22 isprint_l 68
4.2.1.23 isgraph 69
4.2.1.24 isgraph_l 70
4.2.2 Conversion functions 71
4.2.2.1 toupper 72
4.2.2.2 toupper_l 73
4.2.2.3 tolower 74
4.2.2.4 tolower_l 75
4.3 <errno.h> 76
4.3.1 Errors 76
4.3.1.1 Error names 76
4.3.1.2 errno 77
4.4 <float.h> 78
4.4.1 Floating-point constants 78
4.4.1.1 Common parameters 78
4.4.1.2 Float parameters 79
4.4.1.3 Double parameters 80
4.5 <iso646.h> 81
4.5.1 Macros 81
4.5.1.1 Replacement macros 81
4.6 <limits.h> 82
4.6.1 Minima and maxima 82
4.6.1.1 Character minima and maxima 82
4.6.1.2 Short integer minima and maxima 83
4.6.1.3 Integer minima and maxima 84
4.6.1.4 Long integer minima and maxima (32-bit) 85
4.6.1.5 Long integer minima and maxima (64-bit) 86
4.6.1.6 Long long integer minima and maxima 87
4.6.1.7 Multibyte characters 88
4.7 <locale.h> 89
4.7.1 Data types 89
4.7.1.1 __SEGGER_RTL_lconv 89
4.7.2 Locale management 90
4.7.2.1 setlocale 91
4.7.2.2 localeconv 92
4.8 <math.h> 93
4.8.1 Exponential and logarithm functions 93
4.8.1.1 sqrt 94

4.8.1.2 sqrtf 95
4.8.1.3 cbrt 96
4.8.1.4 cbrtf 97
4.8.1.5 exp 98
4.8.1.6 expf 99
4.8.1.7 expm1f 100
4.8.1.8 exp2 101
4.8.1.9 exp2f 102
4.8.1.10 exp10 103
4.8.1.11 exp10f 104
4.8.1.12 frexp 105
4.8.1.13 frexpf 106
4.8.1.14 hypot 107
4.8.1.15 hypotf 108
4.8.1.16 log 109
4.8.1.17 logf 110
4.8.1.18 log10 111
4.8.1.19 log10f 112
4.8.1.20 ldexp 113
4.8.1.21 ldexpf 114
4.8.1.22 pow 115
4.8.1.23 powf 116
4.8.1.24 scalbn 117
4.8.1.25 scalbnf 118
4.8.2 Trigonometric functions 119
4.8.2.1 sin 120
4.8.2.2 sinf 121
4.8.2.3 cos 122
4.8.2.4 cosf 123
4.8.2.5 tan 124
4.8.2.6 tanf 125
4.8.2.7 sinh 126
4.8.2.8 sinhf 127
4.8.2.9 cosh 128
4.8.2.10 coshf 129
4.8.2.11 tanh 130
4.8.2.12 tanhf 131
4.8.3 Inverse trigonometric functions 132
4.8.3.1 asin 133
4.8.3.2 asinf 134
4.8.3.3 acos 135
4.8.3.4 acosf 136
4.8.3.5 atan 137
4.8.3.6 atanf 138
4.8.3.7 atan2 139
4.8.3.8 atan2f 140
4.8.3.9 asinh 141
4.8.3.10 asinhf 142
4.8.3.11 acosh 143
4.8.3.12 acoshf 144
4.8.3.13 atanh 145
4.8.3.14 atanhf 146
4.8.4 Rounding and remainder functions 147
4.8.4.1 ceil 148
4.8.4.2 ceilf 149
4.8.4.3 floor 150
4.8.4.4 floorf 151
4.8.4.5 fmod 152
4.8.4.6 fmodf 153
4.8.4.7 modf 154

4.8.4.8 modff 155
4.8.5 Absolute value functions 156
4.8.5.1 fabs 157
4.8.5.2 fabsf 158
4.8.6 Fused multiply functions 159
4.8.6.1 fma 160
4.8.6.2 fmaf 161
4.8.7 Maximum, minimum, and positive difference functions 162
4.8.7.1 fmin 163
4.8.7.2 fminf 164
4.8.7.3 fmax 165
4.8.7.4 fmaxf 166
4.9 <setjmp.h> 167
4.9.1 Non-local flow control 167
4.9.1.1 setjmp 167
4.9.1.2 longjmp 168
4.10 <stdbool.h> 169
4.10.1 Macros 169
4.10.1.1 bool 169
4.11 <stddef.h> 170
4.11.1 Macros 170
4.11.1.1 NULL 170
4.11.1.2 offsetof 171
4.11.2 Types 172
4.11.2.1 size_t 172
4.11.2.2 ptrdiff_t 173
4.11.2.3 wchar_t 174
4.12 <stdint.h> 175
4.12.1 Minima and maxima 175
4.12.1.1 Signed integer minima and maxima 175
4.12.1.2 Unsigned integer minima and maxima 176
4.12.1.3 Maximal integer minima and maxima 177
4.12.1.4 Least integer minima and maxima 178
4.12.1.5 Fast integer minima and maxima 179
4.12.1.6 Pointer types minima and maxima 180
4.12.1.7 Wide integer minima and maxima 181
4.12.2 Constant construction macros 182
4.12.2.1 Signed integer construction macros 182
4.12.2.2 Unsigned integer construction macros 183
4.12.2.3 Maximal integer construction macros 184
4.13 <stdio.h> 185
4.13.1 Formatted output control strings 185
4.13.1.1 Composition 185
4.13.1.2 Flag characters 185
4.13.1.3 Length modifiers 186
4.13.1.4 Conversion specifiers 186
4.13.2 Formatted input control strings 188
4.13.2.1 Length modifiers 188
4.13.2.2 Conversion specifiers 189
4.13.3 Character and string I/O functions 191
4.13.3.1 getchar 192
4.13.3.2 gets 193
4.13.3.3 putchar 194
4.13.3.4 puts 195
4.13.4 Formatted input functions 196
4.13.4.1 scanf 197
4.13.4.2 sscanf 198
4.13.4.3 vscanf 199
4.13.4.4 vsscanf 200
4.13.5 Formatted output functions 201

4.13.5.1 printf 202
4.13.5.2 sprintf 203
4.13.5.3 snprintf 204
4.13.5.4 vprintf 205
4.13.5.5 vsprintf 206
4.13.5.6 vsnprintf 207
4.14 <stdlib.h> 208
4.14.1 Integer arithmetic functions 208
4.14.1.1 abs 209
4.14.1.2 labs 210
4.14.1.3 llabs 211
4.14.1.4 div 212
4.14.1.5 ldiv 213
4.14.1.6 lldiv 214
4.14.2 Pseudo-random sequence generation functions 215
4.14.2.1 rand 216
4.14.2.2 srand 217
4.14.3 Memory allocation functions 218
4.14.3.1 malloc 219
4.14.3.2 calloc 220
4.14.3.3 realloc 221
4.14.3.4 free 222
4.14.4 Search and sort functions 223
4.14.4.1 qsort 224
4.14.4.2 bsearch 225
4.14.5 Number to string conversions 226
4.14.5.1 itoa 227
4.14.5.2 ltoa 228
4.14.5.3 lltoa 229
4.14.5.4 utoa 230
4.14.5.5 ultoa 231
4.14.5.6 ulltoa 232
4.14.6 String to number conversions 233
4.14.6.1 atoi 234
4.14.6.2 atol 235
4.14.6.3 atoll 236
4.14.6.4 atof 237
4.14.6.5 strtol 238
4.14.6.6 strtoll 239
4.14.6.7 strtoul 240
4.14.6.8 strtoull 241
4.14.6.9 strtof 242
4.14.6.10 strtod 243
4.14.7 Multi-byte/wide character functions 244
4.14.7.1 btowc 245
4.14.7.2 btowc_l 246
4.14.7.3 mblen 247
4.14.7.4 mblen_l 248
4.14.7.5 mbtowc 249
4.14.7.6 mbtowc_l 250
4.14.7.7 mbstowcs 251
4.14.7.8 mbstowcs_l 252
4.14.7.9 mbsrtowcs 253
4.14.7.10 mbsrtowcs_l 254
4.14.7.11 wctomb 255
4.14.7.12 wctomb_l 256
4.14.7.13 wcstombs 257
4.14.7.14 wcstombs_l 258
4.15 <string.h> 259
4.15.1 Copying functions 260

4.15.1.1 memset 261
4.15.1.2 memcpy 262
4.15.1.3 memccpy 263
4.15.1.4 mempcpy 264
4.15.1.5 memmove 265
4.15.1.6 strcpy 266
4.15.1.7 strncpy 267
4.15.1.8 strlcpy 268
4.15.1.9 strcat 269
4.15.1.10 strncat 270
4.15.1.11 strlcat 271
4.15.1.12 strdup 272
4.15.1.13 strndup 273
4.15.2 Comparison functions 274
4.15.2.1 memcmp 275
4.15.2.2 strcmp 276
4.15.2.3 strncmp 277
4.15.2.4 strcasecmp 278
4.15.2.5 strncasecmp 279
4.15.3 Search functions 280
4.15.3.1 memchr 281
4.15.3.2 memrchr 282
4.15.3.3 memmem 283
4.15.3.4 strchr 284
4.15.3.5 strnchr 285
4.15.3.6 strrchr 286
4.15.3.7 strlen 287
4.15.3.8 strnlen 288
4.15.3.9 strstr 289
4.15.3.10 strnstr 290
4.15.3.11 strcasestr 291
4.15.3.12 strncasestr 292
4.15.3.13 strpbrk 293
4.15.3.14 strspn 294
4.15.3.15 strcspn 295
4.15.3.16 strtok 296
4.15.3.17 strtok_r 297
4.15.3.18 strsep 298
4.15.4 Miscellaneous functions 299
4.15.4.1 strerror 300
4.16 <time.h> 301
4.16.1 Operations 301
4.16.1.1 mktime 302
4.16.1.2 difftime 303
4.16.2 Conversion functions 304
4.16.2.1 ctime 305
4.16.2.2 ctime_r 306
4.16.2.3 asctime 307
4.16.2.4 asctime_r 308
4.16.2.5 gmtime 309
4.16.2.6 gmtime_r 310
4.16.2.7 localtime 311
4.16.2.8 localtime_r 312
4.16.2.9 strftime 313
4.16.2.10 strftime_l 314
4.17 <wchar.h> 315
4.17.1 Copying functions 315
4.17.1.1 wmemset 316
4.17.1.2 wmemcpy 317
4.17.1.3 wmemccpy 318

4.17.1.4 wmempcpy 319
4.17.1.5 wmemmove 320
4.17.1.6 wcscpy 321
4.17.1.7 wcsncpy 322
4.17.1.8 wcslcpy 323
4.17.1.9 wcscat 324
4.17.1.10 wcsncat 325
4.17.1.11 wcslcat 326
4.17.1.12 wcsdup 327
4.17.1.13 wcsndup 328
4.17.2 Comparison functions 329
4.17.2.1 wmemcmp 330
4.17.2.2 wcsncmp 331
4.17.2.3 wcscasecmp 332
4.17.2.4 wcsncasecmp 333
4.17.3 Search functions 334
4.17.3.1 wmemchr 335
4.17.3.2 wcschr 336
4.17.3.3 wcsnchr 337
4.17.3.4 wcsrchr 338
4.17.3.5 wcslen 339
4.17.3.6 wcsnlen 340
4.17.3.7 wcsstr 341
4.17.3.8 wcsnstr 342
4.17.3.9 wcspbrk 343
4.17.3.10 wcsspn 344
4.17.3.11 wcscspn 345
4.17.3.12 wcstok 346
4.17.3.13 wcstok_r 347
4.17.3.14 wcssep 348
4.17.4 Multi-byte/wide string conversion functions 349
4.17.4.1 mbsinit 350
4.17.4.2 mbrlen 351
4.17.4.3 mbrlen_l 352
4.17.4.4 mbrtowc 353
4.17.4.5 mbrtowc_l 354
4.17.4.6 wctob 355
4.17.4.7 wctob_l 356
4.17.4.8 wcrtomb 357
4.17.4.9 wcrtomb_l 358
4.17.4.10 wcsrtombs 359
4.17.4.11 wcsrtombs_l 360
4.18 <wctype.h> 361
4.18.1 Classification functions 361
4.18.1.1 iswcntrl 362
4.18.1.2 iswcntrl_l 363
4.18.1.3 iswblank 364
4.18.1.4 iswblank_l 365
4.18.1.5 iswspace 366
4.18.1.6 iswspace_l 367
4.18.1.7 iswpunct 368
4.18.1.8 iswpunct_l 369
4.18.1.9 iswdigit 370
4.18.1.10 iswdigit_l 371
4.18.1.11 iswxdigit 372
4.18.1.12 iswxdigit_l 373
4.18.1.13 iswalpha 374
4.18.1.14 iswalpha_l 375
4.18.1.15 iswalnum 376
4.18.1.16 iswalnum_l 377

4.18.1.17 iswupper 378
4.18.1.18 iswupper_l 379
4.18.1.19 iswlower 380
4.18.1.20 iswlower_l 381
4.18.1.21 iswprint 382
4.18.1.22 iswprint_l 383
4.18.1.23 iswgraph 384
4.18.1.24 iswgraph_l 385
4.18.1.25 iswctype 386
4.18.1.26 iswctype_l 387
4.18.1.27 wctype 388
4.18.2 Conversion functions 389
4.18.2.1 towupper 390
4.18.2.2 towupper_l 391
4.18.2.3 towlower 392
4.18.2.4 towlower_l 393
4.18.2.5 towctrans 394
4.18.2.6 towctrans_l 395
4.18.2.7 wctrans 396
4.18.2.8 wctrans_l 397
4.19 <xlocale.h> 398
4.19.1 Locale management 398
4.19.1.1 newlocale 399
4.19.1.2 duplocale 400
4.19.1.3 freelocale 401
4.19.1.4 localeconv_l 402

5 Runtime library API 403

5.1 GNU library API 404
5.1.1 Floating arithmetic 404
5.1.1.1 __addsf3 405
5.1.1.2 __adddf3 406
5.1.1.3 __subsf3 407
5.1.1.4 __subdf3 408
5.1.1.5 __mulsf3 409
5.1.1.6 __muldf3 410
5.1.1.7 __divsf3 411
5.1.1.8 __divdf3 412
5.1.2 Floating conversions 413
5.1.2.1 __fixsfsi 414
5.1.2.2 __fixdfsi 415
5.1.2.3 __fixsfdi 416
5.1.2.4 __fixdfdi 417
5.1.2.5 __fixunssfsi 418
5.1.2.6 __fixunsdfsi 419
5.1.2.7 __fixunssfdi 420
5.1.2.8 __fixunsdfdi 421
5.1.2.9 __floatsisf 422
5.1.2.10 __floatsidf 423
5.1.2.11 __floatdisf 424
5.1.2.12 __floatdidf 425
5.1.2.13 __floatunsisf 426
5.1.2.14 __floatunsidf 427
5.1.2.15 __floatundisf 428
5.1.2.16 __floatundidf 429
5.1.2.17 __extendsfdf2 430
5.1.2.18 __truncdfsf2 431
5.1.3 Floating comparisons 432
5.1.3.1 __eqsf2 433

5.1.3.2 __eqsf2 434
5.1.3.3 __nesf2 435
5.1.3.4 __nedf2 436
5.1.3.5 __ltsf2 437
5.1.3.6 __ltdf2 438
5.1.3.7 __lesf2 439
5.1.3.8 __ledf2 440
5.1.3.9 __gtsf2 441
5.1.3.10 __gtdf2 442
5.1.3.11 __gesf2 443
5.1.3.12 __gedf2 444

6 Indexes 445

6.1 Index of types 446
6.2 Index of functions 447

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 and RISC-V processors.

1.2 Features

SEGGER Runtime Library is written in standard ANSI C and Arm assembly language and can run on any Arm or RISC-V 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 standard runtime ABIs for the Arm and RISC-V architectures.
- 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.
<inttypes.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.
<math.h>	Mathematical functions.
<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
__libc.h	General definitions used when compiling the library.
__libc_conf.h	Configuration of the library.
__libc_conf_de-faults.h	Default configuration of the library.

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.
errno.c	Support for errno in a tasking environment.
execops.c	Support for execution control functions e.g. atexit().
floatasmops_arm.s	Support for low-level floating point functions (ARM).
floatasmops_rv.s	Support for low-level floating point functions (RISC-V).
floatops.c	Support for high-level floating point functions.
floatops_rv.c	Support for high-level floating point functions (RISC-V).
heapops.c	Support for dynamic memory functions e.g. malloc().
intops.c	Support for high-level integer functions e.g. ldiv().
intops_rv.c	Support for low-level integer functions (RISC-V).
intasmops_arm.s	Support for low-level integer functions (ARM).
intasmops_rv.s	Support for low-level integer functions (RISC-V).
jumpasmops_arm.s	Support for nonlocal 'goto' functions e.g. longjmp (ARM).
jumpasmops_rv.s	Support for nonlocal 'goto' functions e.g. longjmp (RISC-V).
locales.c	Support for various locales.
mbops.c	Support for multi-byte functions e.g. mbtowc().
prinops.c	Support for formatting functions e.g. sprintf().
prinops_rtt.c	Support for formatted output using RTT.
prinops_semi.c	Support for formatted output using semihosting.
prinops_host.c	Support for formatted output using semihosting with host-side formatting.
scanops.c	Support for formatted input functions e.g. scanf().
sortops.c	Support for searching and sorting functions e.g. qsort().
strasmops_arm.s	Support for fast string and memory functions e.g. strcpy() (ARM).
strasmops_rv.s	Support for fast string and memory functions e.g. strcpy() (RISC-V).
strops.c	Support for string and memory functions e.g. strcat().
strops_rv.c	Support for string and memory functions (RISC-V).
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
__SEGGER_RTL_BYTE_ORDER	Select the target's byte order.
__SEGGER_RTL_SIZEOF_WCHAR_T	Select size of the `wchar_t` type.
__SEGGER_RTL_SIZEOF_LONG	Select size of `long` types.
__SEGGER_RTL_SIZEOF_PTR	Select size of pointer types.
__SEGGER_RTL_OPTIMIZE	Prefer size-optimized or speed-optimized code.
__SEGGER_RTL_HEAP_SIZE	The size of the heap, in bytes.
__SEGGER_RTL_FORMAT_INT_WIDTH	Support for `int`, `long`, and `long long` in `printf()` and `scanf()` functions.
__SEGGER_RTL_FORMAT_FLOAT_WIDTH	Support `float` in `printf()` and `scanf()` functions.
__SEGGER_RTL_FORMAT_WIDTH_PRECISION	Support width and precision in `printf()` and `scanf()` functions.
__SEGGER_RTL_FORMAT_CHAR_CLASS	Support character classes in `scanf()` functions.
__SEGGER_RTL_FORMAT_WCHAR	Support wide character output in `printf()` and `scanf()` functions.
__SEGGER_RTL_ATEXIT_COUNT	The maximum number of registered `atexit()` functions.

In many cases these can be configured automatically. For ARM the default configuration of the library is derived from these preprocessor symbols:

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.
__LITTLE_ENDIAN	Target little-endian processors.
__BIG_ENDIAN	Target big-endian processors.

For RISC-V the default configuration of the library is derived from these preprocessor symbols:

Symbol	Description
__riscv	Target is RISC-V.

Symbol	Description
__riscv_abi_rve	Assemble targeting RV32E.
__riscv_compressed	Target has C extension.
__riscv_float_abi_soft	Target has neither F nor D extension.
__riscv_float_abi_single	Target has F extension.
__riscv_float_abi_double	Target has D and F extensions.
__riscv_mul	Target has M extension.
__riscv_muldiv	Target has M extension with divide support.
__riscv_xlen	Distinguish RV32 and RV64 targets.

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 __SEGGER_RTL_BYTE_ORDER

Set this to -1 for little-endian byte order and +1 for big-endian byte order.

2.3.3 __SEGGER_RTL_SIZEOF_LONG

This preprocessor symbol must be set to the underlying size of `long` and must be 4 or 8.

2.3.4 __SEGGER_RTL_SIZEOF_PTR

This preprocessor symbol must be set to the underlying size of pointer types and must be 4 or 8.

2.3.5 __SEGGER_RTL_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.

2.3.6 __SEGGER_RTL_OPTIMIZE

Define the preprocessor symbol `__SEGGER_RTL_OPTIMIZE` 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 balanced implementations.

Value	Description
-2	Favor size at the expense of speed.
-1	Favor size over speed.
0	Balanced.
+1	Favor speed over size.
+2	Favor speed at the expense of size.
+3	Favor speed at the expense of size and place code and read-only data in the fast section.

2.3.7 __SEGGER_RTL_HEAP_SIZE

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

2.3.8 __SEGGER_RTL_FORMAT_INT_WIDTH

To select the level of `printf()` and `scanf()` support, set this preprocessor symbol as follows:

Value	Description
0	Support only int, do not support long or long long.
1	Support int and long, do not support long long.
2	Support int, long, and long long.

2.3.9 __SEGGER_RTL_FORMAT_FLOAT_WIDTH

Set this preprocessor symbol to include floating-point support in `printf()` and `scanf()` as follows:

Value	Description
0	Eliminate all formatted floating point support.
1	Reserved: support output of float values, no doubles.
2	Support output of float and double values.

2.3.10 __SEGGER_RTL_FORMAT_WCHAR

Set this preprocessor symbol to include wide character support in `printf()` and `scanf()` as follows:

Value	Description
0	Eliminate all wide character support.
1	Support formatted input and output of wide characters.

2.3.11 __SEGGER_RTL_FORMAT_CHAR_CLASS

Set this preprocessor symbol to include character class support in `scanf()` as follows:

Value	Description
0	Eliminate all character class support.
1	Support formatted input with character classes.

2.3.12 __SEGGER_RTL_FORMAT_WIDTH_PRECISION

Set this preprocessor symbol to include width and precision suppoer in `printf()` and `scanf()` as follows:

Value	Description
0	Eliminate all width and precision support.
1	Support formatted input and output with width and precision.

2.3.13 __SEGGER_RTL_STDOUT_BUFFER_LEN

Set this preprocessor symbol to set the internal size of the formatting buffer, in characters, used by RTT and semihosting I/O.

2.3.14 __SEGGER_RTL_THREAD

Set this preprocessor symbol to the keyword that introduces thread-local storage, if using thread local storage. The GNU compiler, for instance, uses `__thread` to indicate thread-local data.

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

After returning from `main()` or by calling `exit()`, any registered `atexit` functions must be called to close down. To do this, call `__SEGGER_RTL_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 `__SEGGER_RTL_stdin_getc()`:

```
int __SEGGER_RTL_stdin_getc(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 `__SEGGER_RTL_stdout_putc()`:

```
int __SEGGER_RTL_stdout_putc(int c);
```

The parameter is the character to write to standard output. The return value should be EOF an error or non-negative to indicate success.

Chapter 4

C library API

4.1 <assert.h>

4.1.1 Assertion functions

Function	Description
assert	
__SEGGER_RTL_X_assert	

4.1.1.1 assert

4.1.1.2 __SEGGER_RTL_X_assert

4.2 <ctype.h>

4.2.1 Classification functions

Function	Description
iscntrl	
iscntrl_l	
isblank	
isblank_l	
isspace	
isspace_l	
ispunct	
ispunct_l	
isdigit	
isdigit_l	
isxdigit	
isxdigit_l	
isalpha	
isalpha_l	
isalnum	
isalnum_l	
isupper	
isupper_l	
islower	
islower_l	
isprint	
isprint_l	
isgraph	
isgraph_l	

4.2.1.1 iscntrl

4.2.1.2 iscntrl_l

4.2.1.3 isblank

4.2.1.4 isblank_l

4.2.1.5 isspace

4.2.1.6 isspace_l

4.2.1.7 ispunct

4.2.1.8 ispunct_l

4.2.1.9 isdigit

4.2.1.10 isdigit_l

4.2.1.11 isxdigit

4.2.1.12 isxdigit_l

4.2.1.13 isalpha

4.2.1.14 isalpha_l

4.2.1.15 isalnum

4.2.1.16 isalnum_l

4.2.1.17 isupper

4.2.1.18 isupper_l

4.2.1.19 islower

4.2.1.20 islower_l

4.2.1.21 isprint

4.2.1.22 isprint_l

4.2.1.23 isgraph

4.2.1.24 isgraph_l

4.2.2 Conversion functions

Function	Description
toupper	
toupper_l	
tolower	
tolower_l	

4.2.2.1 toupper

4.2.2.2 toupper_l

4.2.2.3 tolower

4.2.2.4 tolower_l

4.3 <errno.h>

4.3.1 Errors

4.3.1.1 Error names

4.3.1.2 errno

4.4 <float.h>

4.4.1 Floating-point constants

4.4.1.1 Common parameters

4.4.1.2 Float parameters

4.4.1.3 Double parameters

4.5 <iso646.h>

The header `<iso646.h>` defines macros that expand to the corresponding tokens to ease writing C programs with keyboards that do not have keys for frequently-used operators.

4.5.1 Macros

4.5.1.1 Replacement macros

4.6 <limits.h>

4.6.1 Minima and maxima

4.6.1.1 Character minima and maxima

4.6.1.2 Short integer minima and maxima

4.6.1.3 Integer minima and maxima

4.6.1.4 Long integer minima and maxima (32-bit)

4.6.1.5 Long integer minima and maxima (64-bit)

4.6.1.6 Long long integer minima and maxima

4.6.1.7 Multibyte characters

4.7 <locale.h>

4.7.1 Data types

4.7.1.1 __SEGGER_RTL_lconv

4.7.2 Locale management

Function	Description
setlocale	
localeconv	

4.7.2.1 setlocale

4.7.2.2 localeconv

4.8 <math.h>

4.8.1 Exponential and logarithm functions

Function	Description
sqrt	
sqrtf	
cbrt	
cbrtf	
exp	
expf	
expm1f	
exp2	
exp2f	
exp10	
exp10f	
frexp	
frexpf	
hypot	
hypotf	
log	
logf	
log10	
log10f	
ldexp	
ldexpf	
pow	
powf	
scalbn	
scalbnf	

4.8.1.1 sqrt

4.8.1.2 sqrtf

4.8.1.3 cbrt

4.8.1.4 cbrtf

4.8.1.5 exp

4.8.1.6 expf

4.8.1.7 expm1f

4.8.1.8 exp2

4.8.1.9 exp2f

4.8.1.10 exp10

4.8.1.11 exp10f

4.8.1.12 frexp

4.8.1.13 frexpf

4.8.1.14 hypot

4.8.1.15 hypotf

4.8.1.16 log

4.8.1.17 logf

4.8.1.18 log10

4.8.1.19 log10f

4.8.1.20 ldexp

4.8.1.21 ldexpf

4.8.1.22 pow

4.8.1.23 powf

4.8.1.24 scalbn

4.8.1.25 scalbnf

4.8.2 Trigonometric functions

Function	Description
sin	
sinf	
cos	
cosf	
tan	
tanf	
sinh	
sinhf	
cosh	
coshf	
tanh	
tanhf	

4.8.2.1 sin

4.8.2.2 sinf

4.8.2.3 cos

4.8.2.4 cosf

4.8.2.5 tan

4.8.2.6 tanf

4.8.2.7 sinh

4.8.2.8 sinhf

4.8.2.9 cosh

4.8.2.10 coshf

4.8.2.11 tanh

4.8.2.12 tanhf

4.8.3 Inverse trigonometric functions

Function	Description
asin	
asinf	
acos	
acosf	
atan	
atanf	
atan2	
atan2f	
asinh	
asinhf	
acosh	
acoshf	
atanh	
atanhf	

4.8.3.1 asin

4.8.3.2 asinf

4.8.3.3 acos

4.8.3.4 acosf

4.8.3.5 atan

4.8.3.6 atanf

4.8.3.7 atan2

4.8.3.8 atan2f

4.8.3.9 asinh

4.8.3.10 asinhf

4.8.3.11 acosh

4.8.3.12 acoshf

4.8.3.13 atanh

4.8.3.14 atanhf

4.8.4 Rounding and remainder functions

Function	Description
ceil	
ceilf	
floor	
floorf	
fmod	
fmodf	
modf	
modff	

4.8.4.1 ceil

4.8.4.2 ceilf

4.8.4.3 floor

4.8.4.4 floorf

4.8.4.5 fmod

4.8.4.6 fmodf

4.8.4.7 modf

4.8.4.8 modff

4.8.5 Absolute value functions

Function	Description
fabs	
fabsf	

4.8.5.1 fabs

4.8.5.2 fabsf

4.8.6 Fused multiply functions

Function	Description
fma	
fmaf	

4.8.6.1 fma

4.8.6.2 fmaf

4.8.7 Maximum, minimum, and positive difference functions

Function	Description
fmin	
fminf	
fmax	
fmaxf	

4.8.7.1 fmin

4.8.7.2 fminf

4.8.7.3 fmax

4.8.7.4 fmaxf

4.9 <setjmp.h>

4.9.1 Non-local flow control

4.9.1.1 setjmp

4.9.1.2 longjmp

4.10 <stdbool.h>

4.10.1 Macros

4.10.1.1 bool

4.11 <stddef.h>

4.11.1 Macros

4.11.1.1 NULL

4.11.1.2 offsetof

4.11.2 Types

4.11.2.1 size_t

4.11.2.2 ptrdiff_t

4.11.2.3 wchar_t

4.12 <stdint.h>

4.12.1 Minima and maxima

4.12.1.1 Signed integer minima and maxima

4.12.1.2 Unsigned integer minima and maxima

4.12.1.3 Maximal integer minima and maxima

4.12.1.4 Least integer minima and maxima

4.12.1.5 Fast integer minima and maxima

4.12.1.6 Pointer types minima and maxima

4.12.1.7 Wide integer minima and maxima

4.12.2 Constant construction macros

4.12.2.1 Signed integer construction macros

4.12.2.2 Unsigned integer construction macros

4.12.2.3 Maximal integer construction macros

4.13 <stdio.h>

4.13.1 Formatted output control strings

The functions in this section that accept a formatted output control string do so according to the specification that follows.

4.13.1.1 Composition

The format is composed of zero or more directives: ordinary characters (not `%`, which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments, converting them, if applicable, according to the corresponding conversion specifier, and then writing the result to the output stream.

Each conversion specification is introduced by the character `%`. After the `%` the following appear in sequence:

- Zero or more *flags* (in any order) that modify the meaning of the conversion specification.
- An optional *minimum field width*. If the converted value has fewer characters than the field width, it is padded with spaces (by default) on the left (or right, if the left adjustment flag has been given) to the field width. The field width takes the form of an asterisk `*` or a decimal integer.
- An optional precision that gives the minimum number of digits to appear for the `d`, `i`, `o`, `u`, `x`, and `X` conversions, the number of digits to appear after the decimal-point character for `e`, `E`, `f`, and `F` conversions, the maximum number of significant digits for the `g` and `G` conversions, or the maximum number of bytes to be written for `s` conversions. The precision takes the form of a period `.` followed either by an asterisk `*` or by an optional decimal integer; if only the period is specified, the precision is taken as zero. If a precision appears with any other conversion specifier, the behavior is undefined.
- An optional length modifier that specifies the size of the argument.
- A conversion specifier character that specifies the type of conversion to be applied.

As noted above, a field width, or precision, or both, may be indicated by an asterisk. In this case, an int argument supplies the field width or precision. The arguments specifying field width, or precision, or both, must appear (in that order) before the argument (if any) to be converted. A negative field width argument is taken as a `-` flag followed by a positive field width. A negative precision argument is taken as if the precision were omitted.

4.13.1.2 Flag characters

The flag characters and their meanings are:

Flag	Description
`-`	The result of the conversion is left-justified within the field. The default, if this flag is not specified, is that the result of the conversion is left-justified within the field.
`+`	The result of a signed conversion *always* begins with a plus or minus sign. The default, if this flag is not specified, is that it begins with a sign only when a negative value is converted.
space	If the first character of a signed conversion is not a sign, or if a signed conversion results in no characters, a space is prefixed to the result. If the space and `+` flags both appear, the space flag is ignored.
`#`	The result is converted to an *alternative form*. For `o` conversion, it increases the precision, if and only if necessary, to force the first digit of the result to be a zero (if the value and precision are both zero, a single `0` is printed). For `x` or `X` conversion, a nonzero result has `0x` or `0X` prefixed to it. For `e`, `E`, `f`, `F`, `g`, and `G` conversions, the result of converting a floating-point number always contains a decimal-point character, even if no digits follow it. (Normally, a decimal-point char-

Flag	Description
	acter appears in the result of these conversions only if a digit follows it.) For g and F conversions, trailing zeros are not removed from the result. As an extension, when used in p conversion, the results has # prefixed to it. For other conversions, the behavior is undefined.
0	For d, i, o, u, x, X, e, E, f, F, g, and G conversions, leading zeros (following any indication of sign or base) are used to pad to the field width rather than performing space padding, except when converting an infinity or NaN. If the 0 and - flags both appear, the 0 flag is ignored. For d, i, o, u, x, and X conversions, if a precision is specified, the 0 flag is ignored. For other conversions, the behavior is undefined.

4.13.1.3 Length modifiers

The length modifiers and their meanings are:

Flag	Description
hh	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a signed char or unsigned char argument (the argument will have been promoted according to the integer promotions, but its value will be converted to signed char or unsigned char before printing); or that a following n conversion specifier applies to a pointer to a signed char argument.
h	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a short int or unsigned short int argument (the argument will have been promoted according to the integer promotions, but its value is converted to short int or unsigned short int before printing); or that a following n conversion specifier applies to a pointer to a short int argument.
l	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a long int or unsigned long int argument; that a following n conversion specifier applies to a pointer to a long int argument; or has no effect on a following e, E, f, F, g, or G conversion specifier.
ll	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a long long int or unsigned long long int argument; that a following n conversion specifier applies to a pointer to a long long int argument.

If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.

4.13.1.4 Conversion specifiers

The conversion specifiers and their meanings are:

Flag	Description
d, i	The argument is converted to signed decimal in the style [-]*dddd*. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it is expanded with leading spaces. The default precision is one. The result of converting a zero value with a precision of zero is no characters.
o, u, x, X	The unsigned argument is converted to unsigned octal for o, unsigned decimal for u, or unsigned hexadecimal notation for x or X in the style *dddd* the letters abcdef are used for x conversion and the letters ABCDEF for X conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in

Flag	Description
	fewer digits, it is expanded with leading spaces. The default precision is one. The result of converting a zero value with a precision of zero is no characters.
f, F	A double argument representing a floating-point number is converted to decimal notation in the style [-]*ddd.ddd*, where the number of digits after the decimal-point character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is zero and the # flag is not specified, no decimal-point character appears. If a decimal-point character appears, at least one digit appears before it. The value is rounded to the appropriate number of digits. A double argument representing an infinity is converted to inf. A double argument representing a NaN is converted to nan. The F conversion specifier produces INF or NAN instead of inf or nan, respectively.
e, E	A double argument representing a floating-point number is converted in the style [-]*d.ddd*e±*dd*, where there is one digit (which is nonzero if the argument is nonzero) before the decimal-point character and the number of digits after it is equal to the precision; if the precision is missing, it is taken as 6; if the precision is zero and the # flag is not specified, no decimal-point character appears. The value is rounded to the appropriate number of digits. The E conversion specifier produces a number with E instead of e introducing the exponent. The exponent always contains at least two digits, and only as many more digits as necessary to represent the exponent. If the value is zero, the exponent is zero. A double argument representing an infinity is converted to inf. A double argument representing a NaN is converted to nan. The E conversion specifier produces INF or NAN instead of inf or nan, respectively.
g, G	A double argument representing a floating-point number is converted in style f or e (or in style F or e in the case of a G conversion specifier), with the precision specifying the number of significant digits. If the precision is zero, it is taken as one. The style used depends on the value converted; style e (or E) is used only if the exponent resulting from such a conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional portion of the result unless the # flag is specified; a decimal-point character appears only if it is followed by a digit. A double argument representing an infinity is converted to inf. A double argument representing a NaN is converted to nan. The G conversion specifier produces INF or NAN instead of inf or nan, respectively.
c	The argument is converted to an unsigned char, and the resulting character is written.
s	The argument is be a pointer to the initial element of an array of character type. Characters from the array are written up to (but not including) the terminating null character. If the precision is specified, no more than that many characters are written. If the precision is not specified or is greater than the size of the array, the array must contain a null character.
p	The argument is a pointer to void. The value of the pointer is converted in the same format as the x conversion specifier with a fixed precision of 2*sizeof(void *).
n	The argument is a pointer to a signed integer into which is *written* the number of characters written to the output stream so far by the call to the formatting function. No argument is converted, but one is consumed. If the conversion specification includes any flags, a field width, or a precision, the behavior is undefined.
%	A % character is written. No argument is converted.

Note that the C99 width modifier l used in conjunction with the c and s conversion specifiers is not supported and nor are the conversion specifiers a and A.

4.13.2 Formatted input control strings

The format is composed of zero or more directives: one or more white-space characters, an ordinary character (neither % nor a white-space character), or a conversion specification.

Each conversion specification is introduced by the character %. After the %, the following appear in sequence:

- An optional assignment-suppressing character *.
- An optional nonzero decimal integer that specifies the maximum field width (in characters).
- An optional length modifier that specifies the size of the receiving object.
- A conversion specifier character that specifies the type of conversion to be applied.

The formatted input function executes each directive of the format in turn. If a directive fails, the function returns. Failures are described as input failures (because of the occurrence of an encoding error or the unavailability of input characters), or matching failures (because of inappropriate input).

A directive composed of white-space character(s) is executed by reading input up to the first non-white-space character (which remains unread), or until no more characters can be read.

A directive that is an ordinary character is executed by reading the next characters of the stream. If any of those characters differ from the ones composing the directive, the directive fails and the differing and subsequent characters remain unread. Similarly, if end-of-file, an encoding error, or a read error prevents a character from being read, the directive fails.

A directive that is a conversion specification defines a set of matching input sequences, as described below for each specifier. A conversion specification is executed in the following steps:

- Input white-space characters (as specified by the isspace() function) are skipped, unless the specification includes a [, c, or n specifier.
- An input item is read from the stream, unless the specification includes an n specifier. An input item is defined as the longest sequence of input characters which does not exceed any specified field width and which is, or is a prefix of, a matching input sequence. The first character, if any, after the input item remains unread. If the length of the input item is zero, the execution of the directive fails; this condition is a matching failure unless end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is an input failure.
- Except in the case of a % specifier, the input item (or, in the case of a %n directive, the count of input characters) is converted to a type appropriate to the conversion specifier. If the input item is not a matching sequence, the execution of the directive fails: this condition is a matching failure. Unless assignment suppression was indicated by a *, the result of the conversion is placed in the object pointed to by the first argument following the format argument that has not already received a conversion result. If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the object, the behavior is undefined.

4.13.2.1 Length modifiers

The length modifiers and their meanings are:

Flag	Description
hh	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to signed char or pointer to unsigned char.
h	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to short int or unsigned short int.

Flag	Description
l	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to long int or unsigned long int; that a following e, E, f, F, g, or G conversion specifier applies to an argument with type pointer to double.
ll	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to long long int or unsigned long long int.

If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined. Note that the C99 length modifiers j, z, and t are not supported.

4.13.2.2 Conversion specifiers

Flag	Description
d	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the strtol() function with the value 10 for the base argument. The corresponding argument must be a pointer to signed integer.
i	Matches an optionally signed integer, whose format is the same as expected for the subject sequence of the strtol() function with the value zero for the base argument. The corresponding argument must be a pointer to signed integer.
o	Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of the strtol() function with the value 18 for the base argument. The corresponding argument must be a pointer to signed integer.
u	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the strtoul() function with the value 10 for the base argument. The corresponding argument must be a pointer to unsigned integer.
x	Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of the strtoul() function with the value 16 for the base argument. The corresponding argument must be a pointer to unsigned integer.
e, f, g	Matches an optionally signed floating-point number whose format is the same as expected for the subject sequence of the strtod() function. The corresponding argument shall be a pointer to floating.
c	Matches a sequence of characters of exactly the number specified by the field width (one if no field width is present in the directive). The corresponding argument must be a pointer to the initial element of a character array large enough to accept the sequence. No null character is added.
s	Matches a sequence of non-white-space characters The corresponding argument must be a pointer to the initial element of a character array large enough to accept the sequence and a terminating null character, which will be added automatically.
[	Matches a nonempty sequence of characters from a set of expected characters (the *scanset*). The corresponding argument must be a pointer to the initial element of a character array large enough to accept the sequence and a terminating null character, which will be added automatically. The conversion specifier includes all subsequent characters in the format string, up to and including the matching right bracket]. The characters between the brackets (the *scanlist*) compose the scanset, unless the character after the left bracket is a circumflex ^, in which case the scanset contains all characters that do

Flag	Description
	not appear in the scanlist between the circumflex and the right bracket. If the conversion specifier begins with `[]` or`[^]`, the right bracket character is in the scanlist and the next following right bracket character is the matching right bracket that ends the specification; otherwise the first following right bracket character is the one that ends the specification. If a `-` character is in the scanlist and is not the first, nor the second where the first character is a `^`, nor the last character, it is treated as a member of the scanset.
`p`	Reads a sequence output by the corresponding `%p` formatted output conversion. The corresponding argument must be a pointer to a pointer to `void`.
`n`	No input is consumed. The corresponding argument shall be a pointer to signed integer into which is to be written the number of characters read from the input stream so far by this call to the formatted input function. Execution of a `%n` directive does not increment the assignment count returned at the completion of execution of the fscanf function. No argument is converted, but one is consumed. If the conversion specification includes an assignment-suppressing character or a field width, the behavior is undefined.
`%`	Matches a single `%` character; no conversion or assignment occurs.

Note that the C99 width modifier `l` used in conjunction with the `c`, `s`, and `[` conversion specifiers is not supported and nor are the conversion specifiers `a` and `A`.

4.13.3 Character and string I/O functions

Function	Description
getchar	
gets	
putchar	
puts	

4.13.3.1 getchar

4.13.3.2 gets

4.13.3.3 putchar

4.13.3.4 puts

4.13.4 Formatted input functions

Function	Description
scanf	
sscanf	
vscanf	
vsscanf	

4.13.4.1 scanf

4.13.4.2 sscanf

4.13.4.3 vscanf

4.13.4.4 vsscanf

4.13.5 Formatted output functions

Function	Description
printf	
sprintf	
snprintf	
vprintf	
vsprintf	
vsnprintf	

4.13.5.1 printf

4.13.5.2 sprintf

4.13.5.3 snprintf

4.13.5.4 vprintf

4.13.5.5 vsprintf

4.13.5.6 vsnprintf

4.14 <stdlib.h>

4.14.1 Integer arithmetic functions

Function	Description
abs	
labs	
llabs	
div	
ldiv	
lldiv	

4.14.1.1 abs

4.14.1.2 labs

4.14.1.3 llabs

4.14.1.4 div

4.14.1.5 ldiv

4.14.1.6 lldiv

4.14.2 Pseudo-random sequence generation functions

Function	Description
rand	
srand	

4.14.2.1 rand

4.14.2.2 srand

4.14.3 Memory allocation functions

Function	Description
malloc	
calloc	
realloc	
free	

4.14.3.1 malloc

4.14.3.2 calloc

4.14.3.3 realloc

4.14.3.4 free

4.14.4 Search and sort functions

Function	Description
qsort	
bsearch	

4.14.4.1 qsort

4.14.4.2 bsearch

4.14.5 Number to string conversions

Function	Description
itoa	
ltoa	
lltoa	
utoa	
ultoa	
ulltoa	

4.14.5.1 itoa

4.14.5.2 ltoa

4.14.5.3 lltoa

4.14.5.4 utoa

4.14.5.5 ultoa

4.14.5.6 ulltoa

4.14.6 String to number conversions

Function	Description
atoi	
atol	
atoll	
atof	
strtol	
strtoll	
strtoul	
strtoull	
strtof	
strtod	

4.14.6.1 atoi

4.14.6.2 atol

4.14.6.3 atoll

4.14.6.4 atof

4.14.6.5 strtol

4.14.6.6 strtoll

4.14.6.7 strtoul

4.14.6.8 strtoull

4.14.6.9 strtof

4.14.6.10 strtod

4.14.7 Multi-byte/wide character functions

Function	Description
btowc	
btowc_l	
mblen	
mblen_l	
mbtowc	
mbtowc_l	
mbstowcs	
mbstowcs_l	
mbsrtowcs	
mbsrtowcs_l	
wctomb	
wctomb_l	
wcstombs	
wcstombs_l	

4.14.7.1 btowc

4.14.7.2 btowc_l

4.14.7.3 mblen

4.14.7.4 mblen_l

4.14.7.5 mbtowc

4.14.7.6 mbtowc_l

4.14.7.7 mbstowcs

4.14.7.8 mbstowcs_l

4.14.7.9 mbsrtowcs

4.14.7.10 mbsrtowcs_l

4.14.7.11 wctomb

4.14.7.12 wctomb_l

4.14.7.13 wcstombs

4.14.7.14 wcstombs_l

4.15 <string.h>

The header file `<string.h>` defines functions that operate on arrays that are interpreted as null-terminated strings.

Various methods are used for determining the lengths of the arrays, but in all cases a `char *` or `void *` argument points to the initial (lowest addressed) character of the array. If an array is accessed beyond the end of an object, the behavior is undefined.

Where an argument declared as `size_t` *n* specifies the length of an array for a function, *n* can have the value zero on a call to that function. Unless explicitly stated otherwise in the description of a particular function, pointer arguments must have valid values on a call with a zero size. On such a call, a function that locates a character finds no occurrence, a function that compares two character sequences returns zero, and a function that copies characters copies zero characters.

4.15.1 Copying functions

Function	Description
memset	
memcpy	
memccpy	
mempcpy	
memmove	
strcpy	
strncpy	
strlcpy	
strcat	
strncat	
strlcat	
strdup	
strndup	

4.15.1.1 memset

4.15.1.2 memcpy

4.15.1.3 memccpy

4.15.1.4 mempcpy

4.15.1.5 memmove

4.15.1.6 strcpy

4.15.1.7 strncpy

4.15.1.8 strlcpy

4.15.1.9 strcat

4.15.1.10 strncat

4.15.1.11 strlcat

4.15.1.12 strdup

4.15.1.13 strndup

4.15.2 Comparison functions

Function	Description
memcmp	
strcmp	
strncmp	
strcasecmp	
strncasecmp	

4.15.2.1 memcmp

4.15.2.2 strcmp

4.15.2.3 strncmp

4.15.2.4 strcasecmp

4.15.2.5 strncasecmp

4.15.3 Search functions

Function	Description
memchr	
memrchr	
memmem	
strchr	
strnchr	
strrchr	
strlen	
strnlen	
strstr	
strnstr	
strcasestr	
strncasestr	
strpbrk	
strspn	
strcspn	
strtok	
strtok_r	
strsep	

4.15.3.1 memchr

4.15.3.2 memrchr

4.15.3.3 memmem

4.15.3.4 strchr

4.15.3.5 strnchr

4.15.3.6 strrchr

4.15.3.7 strlen

4.15.3.8 strnlen

4.15.3.9 strstr

4.15.3.10 strnstr

4.15.3.11 strcasestr

4.15.3.12 strncasestr

4.15.3.13 strpbrk

4.15.3.14 strspn

4.15.3.15 strcspn

4.15.3.16 strtok

4.15.3.17 strtok_r

4.15.3.18 strsep

4.15.4 Miscellaneous functions

Function	Description
strerror	

4.15.4.1 strerror

4.16 <time.h>

4.16.1 Operations

Function	Description
mktime	
difftime	

4.16.1.1 mktime

4.16.1.2 difftime

4.16.2 Conversion functions

Function	Description
ctime	
ctime_r	
asctime	
asctime_r	
gmtime	
gmtime_r	
localtime	
localtime_r	
strftime	
strftime_l	

4.16.2.1 ctime

4.16.2.2 ctime_r

4.16.2.3 asctime

4.16.2.4 asctime_r

4.16.2.5 gmtime

4.16.2.6 gmtime_r

4.16.2.7 localtime

4.16.2.8 localtime_r

4.16.2.9 strftime

4.16.2.10 strftime_l

4.17 <wchar.h>

4.17.1 Copying functions

Function	Description
wmemset	
wmemcpy	
wmemccpy	
wmempcpy	
wmemmove	
wcscpy	
wcsncpy	
wcslcpy	
wcscat	
wcsncat	
wcslcat	
wcsdup	
wcsndup	

4.17.1.1 wmemset

4.17.1.2 wmemcpy

4.17.1.3 wmemccpy

4.17.1.4 wmempcpy

4.17.1.5 wmemmove

4.17.1.6 wcscpy

4.17.1.7 wcsncpy

4.17.1.8 wcslcpy

4.17.1.9 wcscat

4.17.1.10 wcsncat

4.17.1.11 wcslcat

4.17.1.12 wcsdup

4.17.1.13 wcsndup

4.17.2 Comparison functions

Function	Description
wmemcmp	
wcsncmp	
wcscasecmp	
wcsncasecmp	

4.17.2.1 wmemcmp

4.17.2.2 wcsncmp

4.17.2.3 wcscasecmp

4.17.2.4 wcsncasecmp

4.17.3 Search functions

Function	Description
wmemchr	
wcschr	
wcsnchr	
wcsrchr	
wcslen	
wcsnlen	
wcsstr	
wcsnstr	
wcspbrk	
wcsspn	
wcscspn	
wcstok	
wcstok_r	
wcssep	

4.17.3.1 wmemchr

4.17.3.2 wcschr

4.17.3.3 wcsnchr

4.17.3.4 wcsrchr

4.17.3.5 wcslen

4.17.3.6 wcsnlen

4.17.3.7 wcsstr

4.17.3.8 wcsnstr

4.17.3.9 wcspbrk

4.17.3.10 wcsspn

4.17.3.11 wcscspn

4.17.3.12 wcstok

4.17.3.13 wcstok_r

4.17.3.14 wcssep

4.17.4 Multi-byte/wide string conversion functions

Function	Description
mbsinit	
mbrlen	
mbrlen_l	
mbrtowc	
mbrtowc_l	
wctob	
wctob_l	
wcrtomb	
wcrtomb_l	
wcsrtombs	
wcsrtombs_l	

4.17.4.1 mbsinit

4.17.4.2 mbrlen

4.17.4.3 mbrlen_l

4.17.4.4 mbrtowc

4.17.4.5 mbrtowc_l

4.17.4.6 wctob

4.17.4.7 wctob_l

4.17.4.8 wcrtomb

4.17.4.9 wcrtomb_l

4.17.4.10 wcsrtombs

4.17.4.11 wcsrtombs_l

4.18 <wctype.h>

4.18.1 Classification functions

Function	Description
iswcntrl	
iswcntrl_l	
iswblank	
iswblank_l	
iswspace	
iswspace_l	
iswpunct	
iswpunct_l	
iswdigit	
iswdigit_l	
iswxdigit	
iswxdigit_l	
iswalpha	
iswalpha_l	
iswalnum	
iswalnum_l	
iswupper	
iswupper_l	
iswlower	
iswlower_l	
iswprint	
iswprint_l	
iswgraph	
iswgraph_l	
iswctype	
iswctype_l	
wctype	

4.18.1.1 iswcntrl

4.18.1.2 iswcntrl_l

4.18.1.3 iswblank

4.18.1.4 iswblank_l

4.18.1.5 iswspace

4.18.1.6 iswspace_l

4.18.1.7 iswpunct

4.18.1.8 iswpunct_l

4.18.1.9 iswdigit

4.18.1.10 iswdigit_l

4.18.1.11 iswxdigit

4.18.1.12 iswxdigit_l

4.18.1.13 iswalpha

4.18.1.14 iswalpha_l

4.18.1.15 iswalnum

4.18.1.16 iswalnum_l

4.18.1.17 iswupper

4.18.1.18 iswupper_l

4.18.1.19 iswlower

4.18.1.20 iswlower_l

4.18.1.21 iswprint

4.18.1.22 iswprint_l

4.18.1.23 iswgraph

4.18.1.24 iswgraph_l

4.18.1.25 iswctype

4.18.1.26 iswctype_l

4.18.1.27 wctype

4.18.2 Conversion functions

Function	Description
towupper	
towupper_l	
towlower	
towlower_l	
towctrans	
towctrans_l	
wctrans	
wctrans_l	

4.18.2.1 towupper

4.18.2.2 towupper_l

4.18.2.3 towlower

4.18.2.4 towlower_l

4.18.2.5 towctrans

4.18.2.6 towctrans_l

4.18.2.7 wctrans

4.18.2.8 wctrans_l

4.19 <xlocale.h>

4.19.1 Locale management

Function	Description
newlocale	
duplocale	
freelocale	
localeconv_l	

4.19.1.1 newlocale

4.19.1.2 duplocale

4.19.1.3 freelocale

4.19.1.4 localeconv_l

Chapter 5

Runtime library API

5.1 GNU library API

5.1.1 Floating arithmetic

Function	Description
__addsf3	
__adddf3	
__subsf3	
__subdf3	
__mulsf3	
__muldf3	
__divsf3	
__divdf3	

5.1.1.1 __addsf3

5.1.1.2 __adddf3

5.1.1.3 __subsf3

5.1.1.4 __subdf3

5.1.1.5 __mulsf3

5.1.1.6 __muldf3

5.1.1.7 __divsf3

5.1.1.8 __divdf3

5.1.2 Floating conversions

Function	Description
__fixsfsi	
__fixdfsi	
__fixsfdi	
__fixdfdi	
__fixunssfsi	
__fixunsdfsi	
__fixunssfdi	
__fixunsdfdi	
__floatsisf	
__floatsidf	
__floatdisf	
__floatdidf	
__floatunsisf	
__floatunsidf	
__floatundisf	
__floatundidf	
__extendsfdf2	
__truncdfsf2	

5.1.2.1 __fixsfsi

5.1.2.2 __fixdfsi

5.1.2.3 __fixsfdi

5.1.2.4 __fixdfdi

5.1.2.5 __fixunssfsi

5.1.2.6 __fixunsdfsi

5.1.2.7 __fixunssfdi

5.1.2.8 __fixunsdfdi

5.1.2.9 __floatsisf

5.1.2.10 __floatsidf

5.1.2.11 __floatdisf

5.1.2.12 __floatdidf

5.1.2.13 __floatunsisf

5.1.2.14 __floatunsidf

5.1.2.15 __floatundisf

5.1.2.16 __floatundidf

5.1.2.17 __extendsfdf2

5.1.2.18 __truncdfsf2

5.1.3 Floating comparisons

Function	Description
__eqsf2	
__eqsf2	
__nesf2	
__nedf2	
__ltsf2	
__ltdf2	
__lesf2	
__ledf2	
__gtsf2	
__gtdf2	
__gesf2	
__gedf2	

5.1.3.1 __eqsf2

5.1.3.2 __eqsf2

5.1.3.3 __nesf2

5.1.3.4 __nedf2

5.1.3.5 __ltsf2

5.1.3.6 __ltdf2

5.1.3.7 __lesf2

5.1.3.8 __ledf2

5.1.3.9 __gtsf2

5.1.3.10 __gtdf2

5.1.3.11 __gesf2

5.1.3.12 __gedf2

Chapter 6

Indexes

6.1 Index of types

6.2 Index of functions

