Contents

Introduction ............................................................................................................................................................................................................................................................... 25
What is SEGGER Embedded Studio for ARM? ............................................................................................................................................................................................... 26
What we don't tell you ............................................................................................................................................................................................................................................. 28
Getting Started ............................................................................................................................................................................................................................................. 29
Text conventions ............................................................................................................................................................................................................................................. 30
Release notes ............................................................................................................................................................................................................................................. 32
SEGGER Embedded Studio User Guide .................................................................................................................................................................................. 59
SEGGER Embedded Studio standard layout .................................................................................................................................................................................. 60
  Menu bar ............................................................................................................................................................................................................................................. 61
  Title bar ............................................................................................................................................................................................................................................. 62
  Status bar ............................................................................................................................................................................................................................................. 63
  Editing workspace .................................................................................................................................................................................................................. 65
  Docking windows .................................................................................................................................................................................................................. 66
  Dashboard ............................................................................................................................................................................................................................................. 67
SEGGER Embedded Studio help and assistance .................................................................................................................................................................................. 68
Creating and managing projects .................................................................................................................................................................................................................. 70
  Solutions and projects .................................................................................................................................................................................................................. 71
  Creating a project .................................................................................................................................................................................................................. 74
  Adding existing files to a project .................................................................................................................................................................................................................. 75
  Adding new files to a project .................................................................................................................................................................................................................. 76
  Removing a file, folder, project, or project link .................................................................................................................................................................................................................. 77
Building your application .................................................................................................................................................................................................................. 78
  Creating variants using configurations .................................................................................................................................................................................................................. 80
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project options</td>
<td>82</td>
</tr>
<tr>
<td>Configurations and project options</td>
<td>84</td>
</tr>
<tr>
<td>Project macros</td>
<td>86</td>
</tr>
<tr>
<td>Dependencies and build order</td>
<td>88</td>
</tr>
<tr>
<td>Linking and section placement</td>
<td>89</td>
</tr>
<tr>
<td>Using source control</td>
<td>92</td>
</tr>
<tr>
<td>Source control capabilities</td>
<td>93</td>
</tr>
<tr>
<td>Configuring source-control providers</td>
<td>94</td>
</tr>
<tr>
<td>Connecting to the source-control system</td>
<td>95</td>
</tr>
<tr>
<td>File source-control status</td>
<td>96</td>
</tr>
<tr>
<td>Source-control operations</td>
<td>97</td>
</tr>
<tr>
<td>Adding files to source control</td>
<td>98</td>
</tr>
<tr>
<td>Updating files</td>
<td>99</td>
</tr>
<tr>
<td>Committing files</td>
<td>100</td>
</tr>
<tr>
<td>Reverting files</td>
<td>101</td>
</tr>
<tr>
<td>Locking files</td>
<td>102</td>
</tr>
<tr>
<td>Unlocking files</td>
<td>103</td>
</tr>
<tr>
<td>Removing files from source control</td>
<td>104</td>
</tr>
<tr>
<td>Showing differences between files</td>
<td>105</td>
</tr>
<tr>
<td>Source-control properties</td>
<td>106</td>
</tr>
<tr>
<td>Subversion provider</td>
<td>107</td>
</tr>
<tr>
<td>CVS provider</td>
<td>109</td>
</tr>
<tr>
<td>Package management</td>
<td>111</td>
</tr>
<tr>
<td>Exploring your application</td>
<td>115</td>
</tr>
<tr>
<td>Project explorer</td>
<td>116</td>
</tr>
<tr>
<td>Source navigator window</td>
<td>121</td>
</tr>
<tr>
<td>References window</td>
<td>123</td>
</tr>
<tr>
<td>Symbol browser window</td>
<td>124</td>
</tr>
<tr>
<td>Stack usage window</td>
<td>129</td>
</tr>
<tr>
<td>Memory usage window</td>
<td>130</td>
</tr>
<tr>
<td>Bookmarks window</td>
<td>133</td>
</tr>
<tr>
<td>Code Outline Window</td>
<td>134</td>
</tr>
<tr>
<td>Editing your code</td>
<td>135</td>
</tr>
<tr>
<td>Basic editing</td>
<td>136</td>
</tr>
<tr>
<td>Moving the insertion point</td>
<td>137</td>
</tr>
<tr>
<td>Adding text</td>
<td>139</td>
</tr>
<tr>
<td>Deleting text</td>
<td>140</td>
</tr>
<tr>
<td>Using the clipboard</td>
<td>141</td>
</tr>
<tr>
<td>Undo and redo</td>
<td>142</td>
</tr>
<tr>
<td>Drag and drop</td>
<td>143</td>
</tr>
<tr>
<td>Searching</td>
<td>144</td>
</tr>
</tbody>
</table>
Advanced editing .............................................................................................................................................. 145
  Indenting source code ........................................................................................................................................ 146
  Commenting out sections of code .................................................................................................................. 148
  Adjusting letter case ....................................................................................................................................... 149
Using bookmarks .................................................................................................................................................. 150
Find and Replace window ..................................................................................................................................... 152
Clipboard Ring window ......................................................................................................................................... 154
Mouse-click accelerators ....................................................................................................................................... 156
Regular expressions ............................................................................................................................................... 158
Debugging windows ............................................................................................................................................. 160
  Locals window ................................................................................................................................................ 160
  Globals window ............................................................................................................................................. 162
  Watch window ............................................................................................................................................... 164
  Register window .......................................................................................................................................... 167
  Memory window .......................................................................................................................................... 170
Breakpoints window ........................................................................................................................................... 174
  Call Stack window ..................................................................................................................................... 178
  Threads window .......................................................................................................................................... 181
Execution Profile window .................................................................................................................................... 185
Execution Trace window ....................................................................................................................................... 186
Debug file search editor ...................................................................................................................................... 187
Debug Terminal window ....................................................................................................................................... 189
Breakpoint expressions ....................................................................................................................................... 190
Debug expressions ............................................................................................................................................... 191
Utility windows ................................................................................................................................................... 192
  Terminal emulator window .............................................................................................................................. 192
Command-line options ........................................................................................................................................ 193
  -D (Define macro) ...................................................................................................................................... 194
  -nocolang (Disable Clang support) ................................................................................................................ 195
  -packagesdir (Specify packages directory) .................................................................................................. 196
  -permit-multiple-studio-instances (Permit multiple studio instances) ......................................................... 197
  -rootuserdir (Set the root user data directory) .............................................................................................. 198
  -save-settings-off (Disable saving of environment settings) ................................................................. 199
  -set-setting (Set environment setting) ...................................................................................................... 200
  -templatesfile (Set project templates path) .............................................................................................. 201
Uninstalling SEGGER Embedded Studio for ARM .......................................................................................... 203
ARM target support ............................................................................................................................................. 207
  Target startup code ..................................................................................................................................... 209
  Startup code ............................................................................................................................................... 211
  Section Placement ..................................................................................................................................... 214
Using the SEGGER Assembler .......................................................................................................................... 217
<ctype.h> ................................................................. 292

isetnum ............................................................... 294
isetnum_l ............................................................. 295
isalpha ................................................................. 296
isalpha_l ............................................................... 297
isblank ................................................................. 298
isblank_l .............................................................. 299
iscntrl ................................................................. 300
iscntrl_l .............................................................. 301
isdigit ................................................................. 302
isdigit_l ............................................................... 303
isgraph ............................................................... 304
isgraph_l ............................................................ 305
islower ............................................................... 306
islower_l ............................................................. 307
isprint ............................................................... 308
isprint_l .............................................................. 309
ispunct .............................................................. 310
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ispunct_l</td>
<td>311</td>
</tr>
<tr>
<td>isspace</td>
<td>312</td>
</tr>
<tr>
<td>isspace_l</td>
<td>313</td>
</tr>
<tr>
<td>isupper</td>
<td>314</td>
</tr>
<tr>
<td>isupper_l</td>
<td>315</td>
</tr>
<tr>
<td>isxdigit</td>
<td>316</td>
</tr>
<tr>
<td>isxdigit_l</td>
<td>317</td>
</tr>
<tr>
<td>tolower</td>
<td>318</td>
</tr>
<tr>
<td>tolower_l</td>
<td>319</td>
</tr>
<tr>
<td>toupper</td>
<td>320</td>
</tr>
<tr>
<td>toupper_l</td>
<td>321</td>
</tr>
<tr>
<td>debug_io.h</td>
<td>322</td>
</tr>
<tr>
<td>debug_abort</td>
<td>325</td>
</tr>
<tr>
<td>debug_break</td>
<td>326</td>
</tr>
<tr>
<td>debug_clearerr</td>
<td>327</td>
</tr>
<tr>
<td>debug_enabled</td>
<td>328</td>
</tr>
<tr>
<td>debug_exit</td>
<td>329</td>
</tr>
<tr>
<td>debug_fclose</td>
<td>330</td>
</tr>
<tr>
<td>debug_feof</td>
<td>331</td>
</tr>
<tr>
<td>debug_ferror</td>
<td>332</td>
</tr>
<tr>
<td>debug_fflush</td>
<td>333</td>
</tr>
<tr>
<td>debug_fgetc</td>
<td>334</td>
</tr>
<tr>
<td>debug_fgetpos</td>
<td>335</td>
</tr>
<tr>
<td>debug_fgets</td>
<td>336</td>
</tr>
<tr>
<td>debug_filesize</td>
<td>337</td>
</tr>
<tr>
<td>debug_fopen</td>
<td>338</td>
</tr>
<tr>
<td>debug_fprintf</td>
<td>339</td>
</tr>
<tr>
<td>debug_fprintf_c</td>
<td>340</td>
</tr>
<tr>
<td>debug_fputc</td>
<td>341</td>
</tr>
<tr>
<td>debug_fputs</td>
<td>342</td>
</tr>
<tr>
<td>debug_fread</td>
<td>343</td>
</tr>
<tr>
<td>debug_freopen</td>
<td>344</td>
</tr>
<tr>
<td>debug_fscanf</td>
<td>345</td>
</tr>
<tr>
<td>debug_fscanf_c</td>
<td>346</td>
</tr>
<tr>
<td>debug_fseek</td>
<td>347</td>
</tr>
<tr>
<td>debug_fsetpos</td>
<td>348</td>
</tr>
<tr>
<td>debug_ftell</td>
<td>349</td>
</tr>
<tr>
<td>debug_fwrite</td>
<td>350</td>
</tr>
<tr>
<td>debug_getargs</td>
<td>351</td>
</tr>
<tr>
<td>debug_getch</td>
<td>352</td>
</tr>
<tr>
<td>debug_getchar</td>
<td>353</td>
</tr>
</tbody>
</table>
debug_getd ................................................................. 354
debug_getenv ............................................................. 355
debug_getf ................................................................. 356
debug_geti ................................................................. 357
debug_getl ................................................................. 358
debug_getll ............................................................... 359
debug_gets ................................................................. 360
debug_getu ................................................................. 361
debug_getul ............................................................... 362
debug_getull .............................................................. 363
debug_kbhit ................................................................. 364
debug_loadsymbols .................................................... 365
diagnostic ................................................................. 366
diagnostic ................................................................. 367
diagnostic ................................................................. 368
diagnostic ................................................................. 369
diagnostic ................................................................. 370
diagnostic ................................................................. 371
diagnostic ................................................................. 372
diagnostic ................................................................. 373
diagnostic ................................................................. 374
diagnostic ................................................................. 375
diagnostic ................................................................. 376
diagnostic ................................................................. 377
diagnostic ................................................................. 378
diagnostic ................................................................. 379
diagnostic ................................................................. 380
diagnostic ................................................................. 381
diagnostic ................................................................. 382
diagnostic ................................................................. 383
diagnostic ................................................................. 384
diagnostic ................................................................. 385
diagnostic ................................................................. 386
<errno.h> .................................................................. 387
EDOM .................................................................. 388
EILSEQ .................................................................. 389
EINVAL .................................................................. 390
ENOMEM .................................................................. 391
ERANGE .................................................................. 392
errno .................................................................... 393
<float.h> .................................................................. 394
DBL_DIG ................................................................. 395
DBL_EPSILON ...................................................... 396
DBL_MANT_DIG .................................................... 397
DBL_MAX ............................................................ 398
DBL_MAX_10_EXP ............................................... 399
DBL_MAX_EXP .................................................... 400
DBL_MIN ............................................................ 401
DBL_MIN_10_EXP ............................................... 402
DBL_MIN_EXP .................................................... 403
DECIMAL_DIG .................................................... 404
FLT_DIG ............................................................ 405
FLT_EPSILON ..................................................... 406
FLT_EVAL_METHOD ............................................. 407
FLT_MANT_DIG .................................................. 408
FLT_MAX .......................................................... 409
FLT_MAX_10_EXP ............................................... 410
FLT_MAX_EXP .................................................... 411
FLT_MIN .......................................................... 412
FLT_MIN_10_EXP ............................................... 413
FLT_MIN_EXP .................................................... 414
FLT_RADIX ....................................................... 415
FLT_ROUNDS ..................................................... 416

<iso646.h> ........................................................................ 417
and ................................................................. 418
and_eq .............................................................. 419
bitand ............................................................... 420
bitor ................................................................. 421
compl ............................................................... 422
not ................................................................. 423
not_eq .............................................................. 424
or ................................................................. 425
or_eq .............................................................. 426
xor ................................................................. 427
xor_eq .............................................................. 428

<limits.h> ........................................................................ 429
CHAR_BIT ........................................................... 430
CHAR_MAX ........................................................ 431
CHAR_MIN ........................................................ 432
INT_MAX .......................................................... 433
INT_MIN .......................................................... 434
LONGLONG_MAX .................................................. 435
LLONG_MIN ................................................................................................................................. 436
LONG_MAX ................................................................................................................................. 437
LONG_MIN ................................................................................................................................. 438
MB_LEN_MAX ............................................................................................................................ 439
SCHAR_MAX ............................................................................................................................... 440
SCHAR_MIN ............................................................................................................................... 441
SHRT_MAX ................................................................................................................................. 442
SHRT_MIN ................................................................................................................................. 443
UCHAR_MAX ............................................................................................................................... 444
UINT_MAX ................................................................................................................................... 445
ULLONG_MAX ............................................................................................................................ 446
ULONG_MAX .................................................................................................................................. 447
USHRT_MAX .................................................................................................................................. 448

<locale.h> ................................................................................................................................. 449
iconv ........................................................................................................................................... 450
localeconv .................................................................................................................................. 452
setlocale ...................................................................................................................................... 453

<math.h> ...................................................................................................................................... 454
acos ............................................................................................................................................. 458
acosf .......................................................................................................................................... 459
acosh .......................................................................................................................................... 460
acoshf .......................................................................................................................................... 461
asin .............................................................................................................................................. 462
asinf ............................................................................................................................................ 463
asinh ............................................................................................................................................ 464
asinhf ........................................................................................................................................... 465
atan ............................................................................................................................................. 466
atan2 .......................................................................................................................................... 467
atan2f .......................................................................................................................................... 468
atanf ............................................................................................................................................ 469
atanh ........................................................................................................................................... 470
atanhf .......................................................................................................................................... 471
cbrt ............................................................................................................................................ 472
cbrtf ............................................................................................................................................. 473
ceil ............................................................................................................................................. 474
ceilf ............................................................................................................................................. 475
copysign ...................................................................................................................................... 476
copysignf .................................................................................................................................... 477
cos .............................................................................................................................................. 478
cosf ............................................................................................................................................. 479
cosh ............................................................................................................................................ 480
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>coshf</code></td>
<td>481</td>
</tr>
<tr>
<td><code>erf</code></td>
<td>482</td>
</tr>
<tr>
<td><code>erfc</code></td>
<td>483</td>
</tr>
<tr>
<td><code>erfcf</code></td>
<td>484</td>
</tr>
<tr>
<td><code>erff</code></td>
<td>485</td>
</tr>
<tr>
<td><code>exp</code></td>
<td>486</td>
</tr>
<tr>
<td><code>exp2</code></td>
<td>487</td>
</tr>
<tr>
<td><code>exp2f</code></td>
<td>488</td>
</tr>
<tr>
<td><code>expf</code></td>
<td>489</td>
</tr>
<tr>
<td><code>expm1</code></td>
<td>490</td>
</tr>
<tr>
<td><code>expm1f</code></td>
<td>491</td>
</tr>
<tr>
<td><code>fabs</code></td>
<td>492</td>
</tr>
<tr>
<td><code>fabsf</code></td>
<td>493</td>
</tr>
<tr>
<td><code>fdim</code></td>
<td>494</td>
</tr>
<tr>
<td><code>fdimf</code></td>
<td>495</td>
</tr>
<tr>
<td><code>floor</code></td>
<td>496</td>
</tr>
<tr>
<td><code>floorf</code></td>
<td>497</td>
</tr>
<tr>
<td><code>fma</code></td>
<td>498</td>
</tr>
<tr>
<td><code>fmaf</code></td>
<td>499</td>
</tr>
<tr>
<td><code>fmax</code></td>
<td>500</td>
</tr>
<tr>
<td><code>fmaxf</code></td>
<td>501</td>
</tr>
<tr>
<td><code>fmin</code></td>
<td>502</td>
</tr>
<tr>
<td><code>fminf</code></td>
<td>503</td>
</tr>
<tr>
<td><code>fmod</code></td>
<td>504</td>
</tr>
<tr>
<td><code>fmodf</code></td>
<td>505</td>
</tr>
<tr>
<td><code>fpclassify</code></td>
<td>506</td>
</tr>
<tr>
<td><code>frexp</code></td>
<td>507</td>
</tr>
<tr>
<td><code>frexpf</code></td>
<td>508</td>
</tr>
<tr>
<td><code>hypot</code></td>
<td>509</td>
</tr>
<tr>
<td><code>hypotf</code></td>
<td>510</td>
</tr>
<tr>
<td><code>ilogb</code></td>
<td>511</td>
</tr>
<tr>
<td><code>ilogbf</code></td>
<td>512</td>
</tr>
<tr>
<td><code>isfinite</code></td>
<td>513</td>
</tr>
<tr>
<td><code>isgreater</code></td>
<td>514</td>
</tr>
<tr>
<td><code>isgreaterequal</code></td>
<td>515</td>
</tr>
<tr>
<td><code>isinf</code></td>
<td>516</td>
</tr>
<tr>
<td><code>isless</code></td>
<td>517</td>
</tr>
<tr>
<td><code>islessequal</code></td>
<td>518</td>
</tr>
<tr>
<td><code>islessgreater</code></td>
<td>519</td>
</tr>
<tr>
<td><code>isnan</code></td>
<td>520</td>
</tr>
<tr>
<td><code>isnormal</code></td>
<td>521</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------</td>
<td>------</td>
</tr>
<tr>
<td>isunordered</td>
<td>522</td>
</tr>
<tr>
<td>ldexp</td>
<td>523</td>
</tr>
<tr>
<td>ldexpf</td>
<td>524</td>
</tr>
<tr>
<td>lgamma</td>
<td>525</td>
</tr>
<tr>
<td>lgammaf</td>
<td>526</td>
</tr>
<tr>
<td>llrint</td>
<td>527</td>
</tr>
<tr>
<td>llrintf</td>
<td>528</td>
</tr>
<tr>
<td>llround</td>
<td>529</td>
</tr>
<tr>
<td>llroundf</td>
<td>530</td>
</tr>
<tr>
<td>log</td>
<td>531</td>
</tr>
<tr>
<td>log10</td>
<td>532</td>
</tr>
<tr>
<td>log10f</td>
<td>533</td>
</tr>
<tr>
<td>log1p</td>
<td>534</td>
</tr>
<tr>
<td>log1pf</td>
<td>535</td>
</tr>
<tr>
<td>log2</td>
<td>536</td>
</tr>
<tr>
<td>log2f</td>
<td>537</td>
</tr>
<tr>
<td>logb</td>
<td>538</td>
</tr>
<tr>
<td>logbf</td>
<td>539</td>
</tr>
<tr>
<td>logf</td>
<td>540</td>
</tr>
<tr>
<td>lrint</td>
<td>541</td>
</tr>
<tr>
<td>lrintf</td>
<td>542</td>
</tr>
<tr>
<td>lround</td>
<td>543</td>
</tr>
<tr>
<td>lroundf</td>
<td>544</td>
</tr>
<tr>
<td>modf</td>
<td>545</td>
</tr>
<tr>
<td>modff</td>
<td>546</td>
</tr>
<tr>
<td>nearbyint</td>
<td>547</td>
</tr>
<tr>
<td>nearbyintf</td>
<td>548</td>
</tr>
<tr>
<td>nextafter</td>
<td>549</td>
</tr>
<tr>
<td>nextafterf</td>
<td>550</td>
</tr>
<tr>
<td>pow</td>
<td>551</td>
</tr>
<tr>
<td>powf</td>
<td>552</td>
</tr>
<tr>
<td>remainder</td>
<td>553</td>
</tr>
<tr>
<td>remainderf</td>
<td>554</td>
</tr>
<tr>
<td>remquo</td>
<td>555</td>
</tr>
<tr>
<td>remquof</td>
<td>556</td>
</tr>
<tr>
<td>rint</td>
<td>557</td>
</tr>
<tr>
<td>rintf</td>
<td>558</td>
</tr>
<tr>
<td>round</td>
<td>559</td>
</tr>
<tr>
<td>roundf</td>
<td>560</td>
</tr>
<tr>
<td>scalbln</td>
<td>561</td>
</tr>
<tr>
<td>scalblnf</td>
<td>562</td>
</tr>
</tbody>
</table>
scalbn ........................................... 563
scalbnf ........................................ 564
signbit ........................................ 565
sin ............................................ 566
sinf ........................................... 567
sinh .......................................... 568
sinhf ........................................ 569
sqrt .......................................... 570
sqrtf ........................................ 571
tan ........................................... 572
tanf .......................................... 573
tanh ........................................... 574
tanhf ......................................... 575
tgamma ....................................... 576
tgammaf ..................................... 577
trunc ......................................... 578
truncf ....................................... 579

<stdio.h> ........................................ 580
longjmp ..................................... 581
setjmp ....................................... 582

<stdlib.h> ...................................... 583
va_arg ...................................... 584
va_copy ..................................... 585
va_end ...................................... 586
va_start .................................... 587

<stddef.h> ..................................... 588
NULL ......................................... 589
offsetof .................................... 590
ptrdiff_t .................................... 591
size_t ........................................ 592

<stdio.h> ...................................... 593
getchar ..................................... 594
gets ......................................... 595
printf ....................................... 596
putchar ..................................... 601
puts .......................................... 602
scanf ....................................... 603
snprintf ..................................... 607
sprintf ...................................... 608
vscanf ....................................... 609
vprintf ...................................... 610
### Contents

- `vscanf` ............................................................................................................................................................................. 611
- `vsnprintf` ........................................................................................................................................................................... 612
- `vprintf` .................................................................................................................................................................................. 614
- `vsscanf` .................................................................................................................................................................................. 614
- `<stdlib.h>` ............................................................................................................................................................................ 615
- `EXIT_FAILURE` ....................................................................................................................................................................... 617
- `EXIT_SUCCESS` ....................................................................................................................................................................... 618
- `MB_CUR_MAX` ......................................................................................................................................................................... 619
- `RAND_MAX` ........................................................................................................................................................................... 620
- `abs` ......................................................................................................................................................................................... 621
- `atexit` ..................................................................................................................................................................................... 622
- `atof` ......................................................................................................................................................................................... 623
- `atoi` ......................................................................................................................................................................................... 624
- `atol` ......................................................................................................................................................................................... 625
- `atoll` ....................................................................................................................................................................................... 626
- `bsearch` .................................................................................................................................................................................. 627
- `calloc` .................................................................................................................................................................................... 628
- `div` ......................................................................................................................................................................................... 629
- `div_t` ....................................................................................................................................................................................... 630
- `exit` ......................................................................................................................................................................................... 631
- `free` ......................................................................................................................................................................................... 632
- `itoa` ......................................................................................................................................................................................... 633
- `labs` ......................................................................................................................................................................................... 634
- `ldiv` ......................................................................................................................................................................................... 635
- `ldiv_t` ....................................................................................................................................................................................... 636
- `llabs` ....................................................................................................................................................................................... 637
- `lldiv` ....................................................................................................................................................................................... 638
- `lldiv_t` .................................................................................................................................................................................... 639
- `ltoa` ......................................................................................................................................................................................... 640
- `ltoa` ......................................................................................................................................................................................... 641
- `malloc` .................................................................................................................................................................................... 642
- `mblen` ..................................................................................................................................................................................... 643
- `mblen_l` ................................................................................................................................................................................... 644
- `mbstowcs` ............................................................................................................................................................................. 645
- `mbstowcs_l` ........................................................................................................................................................................... 646
- `mbtowc` ................................................................................................................................................................................ 647
- `mbtowc_l` ............................................................................................................................................................................ 648
- `qsort` ...................................................................................................................................................................................... 649
- `rand` ....................................................................................................................................................................................... 650
- `realloc` .................................................................................................................................................................................. 651
- `srand` ..................................................................................................................................................................................... 652
strtod ................................................................. 653
strtof ................................................................. 654
strtol ................................................................. 655
strtol ................................................................. 657
strtoull ............................................................... 659
strtoall ............................................................... 661
ulltoa ................................................................. 663
ultoa ................................................................. 664
utoa ................................................................. 665
<string.h> .......................................................... 666
memccpy ............................................................ 668
memchr .............................................................. 669
memcmp ............................................................. 670
memcpy ............................................................. 671
memcpy ............................................................... 672
memmove ........................................................... 673
mempcpy ............................................................ 674
memset ............................................................... 675
strcasecmp .......................................................... 676
strcasestr .......................................................... 677
strcat ............................................................... 678
strchr ............................................................... 679
strcspn ............................................................. 680
strdup ............................................................... 681
strlen ............................................................... 682
strerror ............................................................. 683
strerror ............................................................. 684
strcat ............................................................... 685
strcspn ............................................................. 686
strlen ............................................................... 687
strncasecmp ....................................................... 688
strncasestr ......................................................... 689
strncat .............................................................. 690
strnchr ............................................................. 691
strncmp ............................................................. 692
strncpy ............................................................. 693
strndup ............................................................. 694
strnlen ............................................................. 695
strnstr ............................................................. 696
strpbrk ............................................................. 697
strrchr ............................................................. 698
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>strsep</td>
<td>699</td>
</tr>
<tr>
<td>strspn</td>
<td>700</td>
</tr>
<tr>
<td>strstr</td>
<td>701</td>
</tr>
<tr>
<td>strtok</td>
<td>702</td>
</tr>
<tr>
<td>strtok_r</td>
<td>703</td>
</tr>
<tr>
<td>&lt;time.h&gt;</td>
<td>704</td>
</tr>
<tr>
<td>asctime</td>
<td>705</td>
</tr>
<tr>
<td>asctime_r</td>
<td>706</td>
</tr>
<tr>
<td>clock_t</td>
<td>707</td>
</tr>
<tr>
<td>ctime</td>
<td>708</td>
</tr>
<tr>
<td>ctime_r</td>
<td>709</td>
</tr>
<tr>
<td>difftime</td>
<td>710</td>
</tr>
<tr>
<td>gmtime</td>
<td>711</td>
</tr>
<tr>
<td>gmtime_r</td>
<td>712</td>
</tr>
<tr>
<td>localtime</td>
<td>713</td>
</tr>
<tr>
<td>localtime_r</td>
<td>714</td>
</tr>
<tr>
<td>mktime</td>
<td>715</td>
</tr>
<tr>
<td>strftime</td>
<td>716</td>
</tr>
<tr>
<td>time_t</td>
<td>718</td>
</tr>
<tr>
<td>tm</td>
<td>719</td>
</tr>
<tr>
<td>&lt;wchar.h&gt;</td>
<td>720</td>
</tr>
<tr>
<td>WCHAR_MAX</td>
<td>722</td>
</tr>
<tr>
<td>WCHAR_MIN</td>
<td>723</td>
</tr>
<tr>
<td>WEOF</td>
<td>724</td>
</tr>
<tr>
<td>btowc</td>
<td>725</td>
</tr>
<tr>
<td>btowc_l</td>
<td>726</td>
</tr>
<tr>
<td>mbrlen</td>
<td>727</td>
</tr>
<tr>
<td>mbrlen_l</td>
<td>728</td>
</tr>
<tr>
<td>mbrtowc</td>
<td>729</td>
</tr>
<tr>
<td>mbrtowc_l</td>
<td>730</td>
</tr>
<tr>
<td>mbsrtowcs</td>
<td>731</td>
</tr>
<tr>
<td>mbsrtowcs_l</td>
<td>732</td>
</tr>
<tr>
<td>msbinit</td>
<td>733</td>
</tr>
<tr>
<td>wchar_t</td>
<td>734</td>
</tr>
<tr>
<td>wcrtomb</td>
<td>735</td>
</tr>
<tr>
<td>wcrtomb_l</td>
<td>736</td>
</tr>
<tr>
<td>wcscat</td>
<td>737</td>
</tr>
<tr>
<td>wcschr</td>
<td>738</td>
</tr>
<tr>
<td>wcscmp</td>
<td>739</td>
</tr>
<tr>
<td>wcscpy</td>
<td>740</td>
</tr>
<tr>
<td>wcscspn</td>
<td>741</td>
</tr>
</tbody>
</table>


<wctype.h>

wcsdup ................................................................. 742
wcslen ................................................................. 743
wcsncat ................................................................. 744
wcsnchr ................................................................. 745
wcsncmp ................................................................. 746
wcsncpy ................................................................. 747
wcsnlen ................................................................. 748
wcsnstr ................................................................. 749
wcspbrk ................................................................. 750
wcsrchr ................................................................. 751
wcsspn ................................................................. 752
wcsstr ................................................................. 753
wcstok ................................................................. 754
wcstok_r ............................................................... 755
wctob ................................................................. 756
wctob_l ................................................................. 757
wint_t ................................................................. 758
wmemccpy ............................................................. 759
wmemchr .............................................................. 760
wmemcmp ............................................................. 761
wmemcpy .............................................................. 762
wmemmove ........................................................... 763
wmempcpy ............................................................ 764
wmemset .............................................................. 765
wstrsep ............................................................... 766

iswalnum .............................................................. 769
iswalnum_l ............................................................ 770
iswalpha ............................................................... 771
iswalpha_l ............................................................. 772
iswblank .............................................................. 773
iswblank_l ............................................................. 774
iswcntrl ............................................................... 775
iswcntrl_l ............................................................. 776
iswctype .............................................................. 777
iswctype_l ............................................................ 778
iswdigit .............................................................. 779
iswdigit_l ............................................................. 780
iswgraph ............................................................. 781
iswgraph_l ........................................................... 782
iswlower ............................................................. 783
Utilities Reference

C++ Library User Guide

Subset API reference

Standard template library

Command-line options

Compiler driver

File naming conventions

-allow-multiple-definition (Allow multiple symbol definition)............................. 821
-ansi (Warn about potential ANSI problems) .................................................... 822
-ar (Archive output) ......................................................................................... 823
-arch (Set ARM architecture) ......................................................................... 824
-be (Big Endian) ......................................................................................... 825
-builtins (Use Builtins) ................................................................. 826
-c (Compile to object code, do not link) ........................................ 827
-clang (Use clang compiler/assembler) ....................................... 828
-cmselib (Create CMSE import library) ....................................... 829
-codec (Set file codec) ............................................................... 830
-common (Allocate globals in common) ...................................... 831
-cpu (Set ARM cpu core) ............................................................ 832
-d (Define linker symbol) ......................................................... 833
-debugio (Define debugio implementation) .............................. 834
-dep (Generate dependency file) ................................................ 835
-D (Define macro symbol) .......................................................... 836
-emit-relocs (Emit relocations) .................................................. 837
-e (Set entry point symbol) ....................................................... 838
-exceptions (Enable C++ Exception Support) ............................ 839
-E (Preprocess) ........................................................................ 840
-fill (Fill gaps) ......................................................................... 841
-fabi (Floating Point Code Generation) ....................................... 842
-fpu (Set ARM FPU) ................................................................ 843
-framepointer (Enable generation of framepointer) ...................... 844
-F (Set output format) ............................................................... 845
-g (Generate debugging information) ............................................ 846
-hascme (Generate cmse instructions) ......................................... 847
-hascrc (Generate crc instructions) ............................................. 848
-hascrypto (Generate crypto instructions) ..................................... 849
-hasdsp (Generate dsp instructions) ............................................ 850
-hasdiv (Generate integer divide instructions) ............................ 851
-hassmallmultiplier (Do not generate multiply instructions) ...... 852
-help (Display help information) ................................................ 853
-instrument (Instrument functions) ............................................ 854
-I (Define user include directories) ........................................... 855
-I- (Exclude standard include directories) ................................. 856
-J (Define system include directories) ........................................ 857
-kasm (Keep assembly code) ..................................................... 858
-kldscript (Keep linker script) .................................................. 859
-kpp (Keep preprocessor output) ................................................ 860
-K (Keep linker symbol) ............................................................ 861
-L (Do not link standard libraries) .............................................. 862
-longcalls (Generate long calling sequences) ............................ 863
-lto (Enable link time optimization) ......................................... 864
-L (Set library directory path) ................................................... 865
-memorymap (Memory map file) ............................................... 866
-memorymapmacros (Memory map macros) ................................................................. 867
-M (Display linkage map) .......................................................................................... 868
-n (Dry run, no execution) .......................................................................................... 869
-nointerwork (No interwork code for v4t) ................................................................. 870
-nowarn-mismatch (No warning on architecture mismatch) ....................................... 871
-nowarn-enumsize (No warning on enum size mismatch) ........................................... 872
-nowarn-wcharsize (No warning on wide character size mismatch) ......................... 873
-nostderr (No stderr output) ....................................................................................... 874
-O (Optimize output) ................................................................................................... 875
-o (Set output file name) ............................................................................................ 876
-patch (Run patch command) ..................................................................................... 877
-placement (Section placement file) .......................................................................... 878
-placementmacros (Section placement macros) ......................................................... 879
-placementsegments (Section placement segments) .................................................... 880
-printf (Select printf capability) ............................................................................... 881
-rtti (Enable C++ RTTI Support) .............................................................................. 882
-R (Set section name) ................................................................................................. 883
-scanf (Select scanf capability) .................................................................................. 884
-segger (Use SEGGER assembler/compiler/linker) ...................................................... 885
-shortenums (Minimal sized enums) ........................................................................... 886
-shortwchar (16-bit wide chars) .................................................................................. 887
-simd (Generate vector processing code) ....................................................................... 888
-std (Select language standard) .................................................................................. 889
-strip (Strip symbols from executable) ...................................................................... 890
-symbols (Link symbols) ............................................................................................ 891
-thumb (Generate thumb code) .................................................................................... 892
-T (Supply linker script) .............................................................................................. 893
-U (Undefine macro symbol) ....................................................................................... 894
-unwindtables (Generate unwind tables) ...................................................................... 895
-v (Verbose execution) ............................................................................................... 896
-vectorize (Generate vector processing code) ............................................................. 897
-w (Suppress warnings) ............................................................................................... 898
-WE (Treat warnings as errors) ................................................................................... 899
-W (Pass option to tool) ............................................................................................. 900
-x (Specify file types) .................................................................................................. 901

Command-Line Project Builder .................................................................................. 902
  Building with a SEGGER Embedded Studio project file ........................................... 903
  Building without a SEGGER Embedded Studio project file .................................... 905
Command-line options ................................................................................................ 906
  -batch (Batch build) ................................................................................................. 907
  -config (Select build configuration) ......................................................................... 908
-clean (Remove output files) ................................................................. 909
-D (Define macro) .................................................................................. 910
-echo (Show command lines) ............................................................... 911
-file (Build a named file) .................................................................... 912
-packagesdir (Specify packages directory) ........................................... 913
-project (Specify project to build) ....................................................... 914
-property (Set project property) ........................................................ 915
-rebuild (Always rebuild) .................................................................... 916
-show (Dry run, don’t execute) ............................................................ 917
-solution (Specify solution to build) .................................................. 918
-studiordir (Specify SEGGER Embedded Studio directory) ............... 919
-template (Specify project template) .................................................. 920
-time (Time the build) .......................................................................... 921
-threadnum (Specify number of build threads) .................................... 922
-type (Specify project type) ................................................................. 923
-verbose (Show build information) ...................................................... 924

Command-Line Simulator ................................................................... 925

Command-line options ........................................................................ 926
  file (Elf executable file) .................................................................... 927
  -segments (Specify memory segments) ............................................ 928
  args (User arguments) ...................................................................... 929

Command-Line Scripting .................................................................... 930

Command-line options ........................................................................ 931
  -define (Define global variable) ..................................................... 932
  -help (Show usage) ......................................................................... 933
  -load (Load script file) ................................................................... 934
  -define (Verbose output) ............................................................... 935

emScript classes ................................................................................. 936

Example uses ..................................................................................... 937

Embed ................................................................................................. 938

Header file generator .......................................................................... 939

Using the header generator .............................................................. 940
Command line options ........................................................................ 941
  -regbaseoffsets (Use offsets from peripheral base) ....................... 942
  -nobitfields (Inhibit bitfield macros) ............................................. 943

Linker script file generator ................................................................. 944

Command-line options ........................................................................ 945
  -check-segment-overflow ............................................................ 946
  -disable-missing-runin-error ....................................................... 947
  -memory-map-file ....................................................................... 948
  -memory-map-macros .................................................................. 949
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical</td>
<td>556</td>
</tr>
<tr>
<td>File formats</td>
<td>958</td>
</tr>
<tr>
<td>Memory Map file format</td>
<td>959</td>
</tr>
<tr>
<td>Section Placement file format</td>
<td>961</td>
</tr>
<tr>
<td>Project file format</td>
<td>963</td>
</tr>
<tr>
<td>Project Templates file format</td>
<td>964</td>
</tr>
<tr>
<td>Property Groups file format</td>
<td>966</td>
</tr>
<tr>
<td>Package Description file format</td>
<td>968</td>
</tr>
<tr>
<td>External Tools file format</td>
<td>972</td>
</tr>
<tr>
<td>Debugger Type Interpretation file format</td>
<td>975</td>
</tr>
<tr>
<td>Environment Options</td>
<td>977</td>
</tr>
<tr>
<td>Building Environment Options</td>
<td>977</td>
</tr>
<tr>
<td>Debugging Environment Options</td>
<td>979</td>
</tr>
<tr>
<td>IDE Environment Options</td>
<td>982</td>
</tr>
<tr>
<td>Programming Language Environment Options</td>
<td>988</td>
</tr>
<tr>
<td>Source Control Environment Options</td>
<td>991</td>
</tr>
<tr>
<td>Text Editor Environment Options</td>
<td>993</td>
</tr>
<tr>
<td>Windows Environment Options</td>
<td>1004</td>
</tr>
<tr>
<td>Project Options</td>
<td>1014</td>
</tr>
<tr>
<td>Code Options</td>
<td>1014</td>
</tr>
<tr>
<td>Debug Options</td>
<td>1037</td>
</tr>
<tr>
<td>Macros</td>
<td>1046</td>
</tr>
<tr>
<td>System Macros</td>
<td>1046</td>
</tr>
<tr>
<td>Build Macros</td>
<td>1049</td>
</tr>
<tr>
<td>Script classes</td>
<td>1053</td>
</tr>
<tr>
<td>BinaryFile</td>
<td>1053</td>
</tr>
<tr>
<td>CWSys</td>
<td>1054</td>
</tr>
<tr>
<td>Debug</td>
<td>1055</td>
</tr>
<tr>
<td>ElfFile</td>
<td>1056</td>
</tr>
<tr>
<td>TargetInterface</td>
<td>1057</td>
</tr>
<tr>
<td>WScript</td>
<td>1059</td>
</tr>
</tbody>
</table>
Introduction

This guide is divided into a number of sections:

**Introduction**
Covers installing SEGGER Embedded Studio for ARM on your machine and verifying that it operates correctly, followed by a brief guide to the operation of the SEGGER Embedded Studio integrated development environment, debugger, and other software supplied in the product.

**SEGGER Embedded Studio User Guide**
Contains information on how to use the SEGGER Embedded Studio development environment to manage your projects, build, and debug your applications.

**C Library User Guide**
Contains documentation for the functions in the standard C library supplied in SEGGER Embedded Studio for ARM.

**ARM target support**
Contains a description of system files used for startup and debugging of ARM applications.
What is SEGGER Embedded Studio for ARM?

SEGGER Embedded Studio for ARM is a complete C/C++ development system for ARM and Cortex, microcontrollers and microprocessors that runs on Windows, Mac OS and Linux.

C/C++ Compiler

SEGGER Embedded Studio for ARM comes with pre-built versions of both GCC and Clang/LLVM C and C++ compilers and assemblers. The GNU linker and librarian are also supplied to enable you to immediately begin developing applications for ARM.

SEGGER Embedded Studio for ARM C Library

SEGGER Embedded Studio for ARM has its own royalty-free ANSI and ISO C compliant C library that has been specifically designed for use within embedded systems.

SEGGER Embedded Studio for ARM C++ Library

SEGGER Embedded Studio for ARM supplies a C++ library that implements STL containers, exceptions and RTTI.

SEGGER Embedded Studio IDE

SEGGER Embedded Studio for ARM is a streamlined integrated development environment for building, testing, and deploying your applications. SEGGER Embedded Studio provides:

- **Source Code Editor**: A powerful source code editor with multi-level undo and redo, makes editing your code a breeze.
- **Project System**: A complete project system organizes your source code and build rules.
- **Build System**: With a single key press you can build all your applications in a solution, ready for them to be loaded onto a target microcontroller.
- **Debugger and Flash Programming**: You can download your programs directly into Flash and debug them seamlessly from within the IDE using a wide range of target interfaces.
- **Help system**: The built-in help system provides context-sensitive help and a complete reference to the SEGGER Embedded Studio IDE and tools.
- **Core Simulator**: As well as providing cross-compilation technology, SEGGER Embedded Studio for ARM provides a PC-based fully functional simulation of the target microcontroller core so you can debug parts of your application without waiting for hardware.
SEGGER Embedded Studio for ARM Tools

SEGGER Embedded Studio for ARM supplies command line tools that enable you to build your application on the command line using the same project file that the IDE uses.
What we don't tell you

This documentation does not attempt to teach the C or assembly language programming; rather, you should seek out one of the many introductory texts available. And similarly the documentation doesn't cover the ARM architecture or microcontroller application development in any great depth.

We also assume that you're fairly familiar with the operating system of the host computer being used.

C programming guides

These are must-have books for any C programmer:


The original C bible, updated to cover the essentials of ANSI C (1990 version).


A nice reference guide to C, including a useful amount of information on ANSI C. Co-authored by Guy Steele, a noted language expert.

ANSI C reference

If you're serious about C programming, you may want to have the ISO standard on hand:


ARM microcontrollers

For ARM technical reference manuals, specifications, user guides and white papers, go to:


GNU compiler collection

For the latest GCC documentation go to:


LLVM/Clang

For the latest LLVM/Clang documentation to to:

[http://www.llvm.org](http://www.llvm.org)
Getting Started

You will need to install a CPU support package:

Choose **Tools > Package Manager**
Choose the CPU support packages you wish to install and complete the dialog.

You will need to create a project:

Choose **File > New Project**
Select the appropriate Executable project type
Specify a location for the project
Complete the dialog selecting the appropriate **Target Processor** value

You will need to build the project:

Choose **Build | Build 'Project'**

To debug on the simulator

Choose **Project | Options...** to show the project options dialog
In the **Search Options** type in **Simulator**
Choose **Simulator** for the **Target Connection** option

To debug on hardware

Choose **Project | Options...** to show the project options dialog
In the **Search Options** type in **J-Link**
Choose **J-Link** for the **Target Connection** option

To start debugging

Choose **Debug | Go**

The debugger will stop the program at the main, you can now debug the application.
Text conventions

Menus and user interface elements

When this document refers to any user interface element, it will do so in bold font. For instance, you will often see reference to the Project Explorer, which is taken to mean the project explorer window. Similarly, you'll see references to the Standard toolbar which is positioned at the top of the SEGGER Embedded Studio window, just below the menu bar on Windows and Linux.

When you are directed to select an item from a menu in SEGGER Embedded Studio, we use the form menu-name > item-name. For instance, File > Save means that you need to click the File menu in the menu bar and then select the Save item. This form extends to items in sub-menus, so File > Open With Binary Editor has the obvious meaning.

Keyboard accelerators

Frequently-used commands are assigned keyboard accelerators to speed up common tasks. SEGGER Embedded Studio uses standard Windows and Mac OS keyboard accelerators wherever possible.

Windows and Linux have three key modifiers which are Ctrl, Alt, and Shift. For instance, Ctrl+Alt+P means that you should hold down the Ctrl and Alt buttons whilst pressing the P key; and Shift+F5 means that you should hold down the Shift key whilst pressing F5.

Mac OS has four key modifiers which are (command), (option), (control), and (shift). Generally there is a one-to-one correspondence between the Windows modifiers and the Mac OS modifiers: Ctrl is , Alt is , and Shift is . SEGGER Embedded Studio on Mac OS has its own set of unique key sequences using (control) that have no direct Windows equivalent.

SEGGER Embedded Studio on Windows and Linux also uses key chords to expand the set of accelerators. Key chords are key sequences composed of two or more key presses. For instance, the key chord Ctrl+T, D means that you should type Ctrl+T followed by D; and Ctrl+K, Ctrl+Z means that you should type Ctrl+T followed by Ctrl+Z. Mac OS does not support accelerator key chords.

Code examples and human interaction

Throughout the documentation, text printed in this typeface represents verbatim communication with the computer: for example, pieces of C text, commands to the operating system, or responses from the computer. In examples, text printed in this typeface is not to be used verbatim: it represents a class of items, one of which should be used. For example, this is the format of one kind of compilation command:

hcl source-file

This means that the command consists of:

The word hcl, typed exactly like that.

A source-file: not the text source-file, but an item of the source-file class, for example myprog.c.
Whenever commands to and responses from the computer are mixed in the same example, the commands (i.e. the items which you enter) will be presented in this typeface. For example, here is a dialog with the computer using the format of the compilation command given above:

```
c:\code\examples>hcl -v myprog.c
```

The user types the text `hcl -v myprog.c` and then presses the enter key (which is assumed and is not shown); the computer responds with the rest.
Release notes

Version 4.52

Build

Fixed and improved definitions of wchar_t, WCHAR_MAX and WCHAR_MIN in header files.
Fixed output of floating point numbers using printf when precision value is 1 and the Code > Printf/Scanf > Printf Floating Point Supported project property is set to Float.
Fixed nRF Connect SDK project configurer error when an attribute value contains a quote character.
Updated the LLVM/Clang tools build to use the 10.0.0 source release.
Added Windows > Project Explorer > Check Solution Target option.
Files now appear in alphabetical order when dragging and dropping them into a project explorer folder.
Updated SEGGER compiler to version 10.0.0.
Fixed Left-justify printf format directive.

Debug

Fixed connection failure when using "IP tunnel" in the J-Link > Host Connection property.
Fixed source file identification when debugging multiple elf files.
Debug > Debug With Ozone now uses the correct default Ozone installation location when no executable is specified (macOS only).
Updated J-Link software to version V6.70a.

Editor

Fixed syntax coloring of hex floating point constants.
Added Text Editor > Visual Appearance > View Whitespace option.
Fixed block editing on lines containing tab characters.
Added Edit > Others > Toggle Column Guide.

IDE

Added new solution launcher that starts the appropriate variant of Embedded Studio if the target attribute is defined in the solution. This can be enabled and disabled using the Environment > Launcher Enabled environment option.
Fixed appearance of memory usage windows when window is small.
Fixed display of non-printable characters in binary editor.
The binary editor font can now be specified using the Environment > User Interface > Application Monospace Font option.
macOS application bundle paths can now be used when specifying the location of external executables.
First run project now created only if one does not exist already.
Fixed first menu item sometimes being selected when project explorer context menu is opened with mouse or track pad (macOS only).
Fixed display of non-printable characters in binary editor.

Version 4.50

Build

Updated SEGGER compiler to version 1.12.

Debug

Fixed crash with ElfDwarf files containing .debug_types section.
Updated J-Link software to version V6.62a.
Fixed timeout problems with TargetInterface.runFromToAddress (and variants) when used with J-Link target interface.

IDE

Fixed crash after deleting entries from clipboard ring window.
Fixed dashboard window's package status box not returning to green after all packages have been updated.

Version 4.42

Build

The User Include Directories are supplied to the assembler to support .include and .incbin directives.
Updated the LLVM/Clang tools build to use the 9.0.1 source release.
Fixed lock up when removing Externally Built projects that have the Clean Command property set.
Corrected integerization of floating values (Thumb-2 instruction set only).
Fixed new project wizard's build configuration selector selecting incorrect configurations with some project templates.

Debug

Add cJTAG to Target Interface Type project property.
Fix cycle counter update for v7m targets.
Fixed displaying breakpoints on code lines in startup completion symbol before startup completion has occurred.
Add embOS, FreeRTOS and ChibiOS threads scripts to $(StudioDir)/samples.
Updated shipped J-Link software to V6.62.
Editor

Fixed block uppercase and lowercase operations when block has been marked from top right to bottom left.

IDE

Fixed .elf files not appearing in the Target > Download File > Download Elf File... and Target > Verify File > Verify Elf File... file browsers.
Keyboard options dialog now displays command descriptions as a tooltip.

Version 4.40

Build

Changed compiler optimization levels. Optimization levels for existing projects will still work as expected.
Fixed hard fault when calling V6M floating point library functions.
Fixed generation of projects from a project template when file nodes have parent folder nodes containing no files.
Updated the GCC/BINUTILS tools build to use the GCC ARM Embedded Toolchain 9-2019-q4-major source release.
Added Use Compiler Driver project property.
Added fixed point implementation of expf selected when Library Optimization is set to Fast.
Project explorer code and data size is now displayed as bytes when less than 1K.
Updated nRF MDK package included with Nordic distribution to version 8.29.0.
Project property Access Variables Within Memory Map Only changed to Restrict Memory Access.
Changing the ARM Core Type project property now updates the ARM Architecture property.
Supplied SEGGER compiler and added SEGGER to the compiler project option.

Debug

Fixed crash when PC Sampling used with J-Link.
Increased the nested structure depth that can be displayed in watch windows.
Add Load Begin Script and Load End Script project properties.
SVD dim elements can now be specified as hex numbers.
Updated shipped J-Link software to V6.60.

Editor

Fixed text editor auto recovery duplicating end-of-line terminators when file being edited is in DOS format on a UNIX system.
Fixed incorrect calculation of caret width when using proportional fonts.
Added missing close button on code editor find and replace dialog.
IDE

Fixed broken dashboard link to first run Hello project.
Fixed menu descriptions not appearing on status bar (Windows and Linux only).
Fixed crash whilst using package manager on some macOS Catalina machines.
Fixed resizing of device selection dialog.
Ctrl+F5 is now mapped to Debug.Break command by default rather than DebugExecuteActive.
Fixed resizing of property editor dialogs.
Fixed IDE not responding after closing project and clicking cancel option if session file cannot be written.

Installer

Linux installer now installs J-Link udev rules file.
Added additional file icons in order to improve appearance at smaller sizes (Linux only).

Licensing

Fixed inability to remove licenses with Linux license manager.

Version 4.30

Build

Fixed display of combined regions in memory usage window.
Updated the LLVM/Clang tools build to use the 9.0.0 source release.
stddef.h offsetof macro now uses _builtin_offsetof.
Fixed Windows > Project Explorer > Read Only Data in Code option working the wrong way around.
Updated SEGGER linker to version 2.34.
Add build environment option to use the compiler driver cc.
Faster floating point performance when linking with libraries optimized for speed.

Debug

Fixed simulator leaving pc on next instruction on b . detection.
Add Load Additional Projects project property to enable a project to load the outputs of other projects.
Fixed single stepping when using GDB server target connection.
Added Debug > GDB Server > Read Timeout and Debug > GDB Server > Write Timeout options.
Fixed GDB server target connection permitting FLASH erase regions to extend past the end of FLASH.
Updated shipped J-Link software to V6.52d.

Editor

Fixed crash when using Edit > Column Tidy and a comment is selected.
Fixed auto-complete suggestion being inserted when enter key is pressed immediately after closing suggestion dialog with mouse click.

IDE

Fixed find in files dialog not disabling find button when search text field is empty.
Fixed crash when using Find References on a preprocessor definition defined on the command line.
Fixed project explorer not sorting tree when files are added using drag and drop.

Version 4.22

Build

Add Generate Assembler Listing File project property.

Debug

Added Debugging > GDB Server Options > Register Access property.
Added support for pyOCD GDB server.
Add support for ARM semihosting.
Updated shipped J-Link software to V6.50a.

Editor

Fixed the Replace in selection option not appearing on the find and replace dialog when only a single line is selected.
Fixed selected text not adjusting size when carrying out a replace in selection.

IDE

Fixed goto definition.
Fixed positioning of dashboard "Install All Updates" button when window width is small.

Version 4.20

Build

Added v8mml libraries to Nordic distribution.
Updated support packages included in Nordic distribution.
Default linker scripts now place segment specific sections first.
Fixed crash when creating an empty solution.
Updated the GCC/BINUTILS tools build to use the GCC ARM Embedded Toolchain 8-2019-q3-update source release.
Debug

Updated shipped J-Link software to V6.48b.
Added GDB server target connection.

Editor

Fixed caret not being visible with certain fonts when located at the end of a line and an underline caret style is selected.  
Editor no longer outputs syntax errors to stderr when carrying out code completion.  
Editor now displays number of lines and columns selected when selecting text.

IDE

PDF documentation now included in help window search results.
Help window search now updates while typing.
Fixed dashboard package status pane not being updated when packages are installed.
Added dashboard button to update all packages.

Version 4.18

Build

Added .init_rodata section to samples/SEGGER_Flash.icf.
Fixed "V8M Mainline" and "FPv5-SP-D16" library build variants.
Pre-Build and Post-Build commands no longer apply to clean operation.
Fixed vfprintf %f format when Printf Floating Point Supported is set to Float.
Added __SEGGER_LINKER and __GNU_LINKER preprocessor definitions.
SEGGER Linker reports blocks that cannot be placed with enhanced error details.
SEGGER Linker Module Summary section of the map file is now sorted in file name order.
Build log window's memory usage summary now displays small sizes in bytes.
Removed fixed point implementations of expf, logf and tanhf from the C library.
Fixed crash when LTO processing of big endian object files.
Fixed -template option on crossbuild.
The preprocessor define __HEAP_SIZE__ is now set to the value of the Heap Size project property.

Debug

Added Debug Location toolbar.
Added additional Debug Symbols File and Debug Symbols Load Address project properties.
Added Run To Control project property and changed Run To to break on entry if empty.
Fix watch window latching the initial size of array variables.
Added **Debug > Debugger > Debug Terminal Log File** project property.
Updated shipped J-Link software to V6.46h.

**Editor**

Fixed caret not being visible with certain fonts when located in virtual space.
Significantly improved speed of selection deletion when editing large files.
Added Hide/Show toolbar on editor context menu.
Fixed goto definition opening a relative include file under certain circumstances.
Code editor will no longer match braces, brackets or parenthesis located within comments.
Fixed incorrect syntax coloring of C comments introduced with the /*! character sequence.
Fixed crash when starting a build with a keyboard accelerator whilst the code suggestion popup is visible.
Fixed breakpointable line markers not appearing on lines that have a temporary breakpoint set.
Fixed delete forward key deleting two characters when code suggestion dialog is visible.

**IDE**

Fixed open with external browser (macOS only).
Package manager can now be opened from the new project dialog.
New project file selection puts files into the folders.
Register window’s group selector popup now displays a scroll bar if there are more groups available than will fit on the display.

**Licensing**

Restrict license check to active project selection.

**Version 4.16**

**Build**

Added C++14 sized deallocation functions.
Updated SEGGER linker to version 2.26.
Fixed crash when calling character type functions and UTF-8 locale codeset has been selected.
Fixed link error when providing user defined __user_find_locale function.
Fixed iswspace function not recognising some characters as spaces when UTF-8 locale codeset has been selected.
Added `programNotSection` parameter to ElfFile.peekBytes and ElfFile.crc32 JavaScript functions.
Fixed `parseInt` and `parseFloat` JavaScript functions.
Fixed `Date.getTime` JavaScript function.
Updated the GCC/BINUTILS tools build to use the GCC ARM Embedded Toolchain 8-2018-q4-major source release.
Updated the LLVM/Clang tools build to use the 7.0.1 source release.  
Pre-Build and Post-Build commands now work with solution and batch builds and on dependent projects.  
Replaced environment options Enable All Warnings Command Line Options and Enforce ANSI Checking Command Line Options with project properties Enable All Warnings Command Line Options, Enable All Warnings C Compiler Only Command Line Options and Enable All Warnings C++ Compiler Only Command Line Options.  
Added Enforce ANSI Checking C Command Line Options and Enforce ANSI Checking C++ Command Line Options.  
Add support for ARMv8-A/R AAarch32 architecture.  
Add __clz*i2, __ctz*i2, __popcount*i2 and __parity*i2 helper functions to C library.  
Add stdatomic.h and helper functions to C library.  
Fix include file dependency generation when Keep Preprocessor Output is enabled.  
Fixed SEGGER linker crash when application contains references to unresolved weak symbols.  
Fixed linker script generation and memory usage display when memory segment ends at 0xFFFFFFFF.

Debug

Added Debug > Target Control > Check Load Sections Fit Target Description project property.  
Fixed incorrect "load section does not fit the target description" error message.  
Fixed saving of TI hex files from memory window.  
Fixed memory window not updating after loading memory from a file.  
Add support for Cortex-A53 executing in AArch32 non-secure state.  
Debugger will display data on debug terminal that is written to the ARM/CortexA/R DCC channel when debugio is not enabled.  
Support ignore count for execution breakpoints.  
Fixed crash clearing execution profile window when not debugging.  
Updated shipped J-Link software to V6.44d.

Editor

Auto-surround text is no longer activated when using block selection.  
Scroll line up and scroll line down operations now move cursor into visible area if it is off-screen.  
Fixed crash when closing editors.  
Fixed code suggestion dialog not highlighting selected item with correct colour when dark theme is selected.  
The colour of column guide bars can now be modified.  
Fixed caret not being visible with certain fonts when an underline caret style is selected.  
Editor keyboard accelerators will now activate when code suggestion popup is visible.  
Added Text Editor > Visual Appearance > Mate Match Off Screen option.

IDE

Fixed crash when dragging a window icon over the icon of another window in the the same docking area.
Fixed register selection pins in register window doing nothing when clicked.
Fixed crash when closing editor windows with the code outline window active.
Added option to open file with external editor to project explorer context menu.
Fixed package manager not correctly uninstalling legacy packages.
Improved terminal emulator receive performance when using high baud rates.
Added `ProjectExplorerExcludeFromBuild` keyboard command.
Debug terminal now handles carriage return characters.
Fixed reassignment of `FindUsingGoogle` command shortcut not being remembered.
Removed text menu separators on macOS.
Fixed the recent projects list not being updated when project is opened from the recent projects window.
Fixed pinning of projects not working in the recent projects window.
Improved resizing of dynamic folder properties window.
Fixed crash when docking windows.
Fixed crash when attempting to create a directory on a Windows drive that does not exist.
Improved appearance of list view titles when using dark theme.
Added `Internet > HTTP Caching` environment option.
Added `Package Manager > Verify Package Downloads` environment option.
Fixed waypoint back and forward keyboard accelerators not working on macOS.

**Licensing**

Added Company-wide License option.

**Version 4.10**

**Build**

Fixed placement of `.data_tcm_run` section in default `flash_to_tcm_placement.xml` file.
Added "Unaligned Access Support" code generation option.
Added "Link Time Optimization" code generation option.
Updated the LLVM/Clang tools build to use the 7.0.0 source release.
Added SEGGER linker version 2.25 and project options to set command line options `--dedupe-code`, `--dedupe-data`, `--inline` and `--merge-strings`.
Added support for ARM Compiler 6 when importing Keil MDK projects.
Fixed output of floating point numbers using `printf` when precision value is 1.
Fixed use of section attributes in source code when the same section has been renamed using the section name project properties.
Fixed linker script not being regenerated when `Code > Linker > Additional Linker Script Generator Options` property is modified.
Setting the `Code > Linker > Check For Memory Segment Overflow` property to No no longer disables unplaced sections checks from the GNU LD linker script. These checks can now be disabled by adding the
-no-check-unplaced-sections option to the Code > Linker > Additional Linker Script Generator Options property.

Fixed --gap-fill option not being passed to objcopy if the Code > Linker > Additional Output File Gap Fill Value property is set to 0x00. Clear this property if existing behaviour is required.

**Debug**

Fixed crash when target connection is lost or reset.
Ozone project filename now includes active project name and build configuration.
Fixed watch window variables not being updated correctly after they have been modified.
Updated shipped J-Link software to V6.34h.

**Editor**

Fixed potential crash when code suggestion popup is displayed and an Alt key combination is pressed.
Changed default text editor font size to 10pt.
Comments are no longer displayed as an italic font by default.
Improved IDE start up and project loading time when a lot of editors are open.

**IDE**

Added Environment > User Interface > Theme option (Windows and Linux versions only).
Fixed sporadic crash when IDE is starting up.
Editor tabs can now be reordered.
Editor tab order is now preserved in session file.
Project files can now be drag and dropped into the project explorer in order to load them.
Fixed missing environment settings when Japanese system locale is selected.
Fixed slow register window search.
Fixed list view windows not using the Environment > User Interface > Application Monospace Font property.

**Version 3.52**

**Build**

Added SEGGER assembler technology preview.
Added --warn-empty-selects command line option for SEGGER linker.

**Debug**

Fixed crash when evaluating certain dwarf information.
Fixed crash while disassembling a line with a long symbol name.
Updated shipped J-Link software to V6.34f.
Editor

Fixed crash if tab size gets set to 0.

IDE

Changed Project > Edit Options... to Project > Options....

Version 3.50

Build

Add support for SEGGER linker.
Fixed crash when building on a machine with more than 16 cores.
Batch builds are now done in parallel.
Updated the GCC/BINUTILS tools build to use the GCC ARM Embedded Toolchain 7-2018-q2-update source release.
Updated the LLVM/Clang tools build to use the 6.0.1 source release.
Added -disable-missing-runin-error linker script generator option.
Added Code > Linker > Additional Linker Script Generator Options project property.
Multiple run in sections can now be specified in section placement file.

Debug

Updated shipped J-Link software to V6.32i.

Editor

Fixed using Alt+Tab when quick search window is visible (Linux only).
Added Text Editor > Programmer Assistance > Code Completion Selection Key option.
Added EditMoveSelectedLineUp and EditMoveSelectedLineDown commands and assigned them to Alt +Up and Alt+Down keys.
Added Text Editor > Programmer Assistance > Code Completion Replaces Existing Word option.
Fixed incorrectly displayed parameters in code suggestion popup when showing overloaded functions.
Fixed use of tab key when function prototypes are displayed in code suggestion popup.

IDE

Add capability to import eclipse .project/.cproject files as supplied with STM32Cube sample projects.
Fixed crash when closing editor windows when code outline window is open.
Fixed processing of multiple backspace characters in text output windows.
Fixed crash in outline window when viewing certain C++ code.
Added terminal emulator window.
Fixed missing clipboard operations from text output window's context menu.
Line edit property editors now initially selected.
Fixed crash in stack usage window viewing certain .elf files.

**Version 3.40**

**Build**

- Added **Environment > Find and Replace > Greedy Regular Expressions** environment option.
- Add support for Cortex-R7, Cortex-R8, Cortex-A15 and Cortex-A17 processor cores.
- Added "V8M Has DSP Instructions" project option.
- Added "V8M Mainline" and "FPv5-SP-D16" library build variants.
- Fix tdata placement in Cortex-M placement files.
- Added c++1z and gnu++1z C++ language standard options.
- Replaced gcc c++ exception handling code with llvm equivalent.
- Fixed command line generated when "Keep Preprocessor Output" is enabled.
- Improved parallel build performance on Linux and macOS hosts.
- Fixed $(FPU) and $(FPAPI) macro definitions for external build commands.
- Add $(Undefines) macro definitions for external build commands.
- Renamed Linker option "Section Placement Segments" to Build option "Memory Segments" and is now available for "Externally Built Executable" project types.
- Added __SES_VERSION preprocessor definition.
- Updated the LLVM/Clang tools build to use the 6.0.0 source release.
- Fixed emScript crash when defining a global variable on the command line without a value.
- Supplied versions of arm_neon.h that are compatible with the supplied gcc and clang compilers.
- Command line builder now implements -verbose, default is to build silently.
- Added Code Generation option "ARM Advanced SIMD Auto Vectorize" to enable loop vectorizing in the compilers.
- Added prototype for __putchar() to stdio.h.
- sys/stat.h now declares mode_t and off_t types.
- Added $(UnixTime) system macro.
- Removed deprecated throw declarations in new header file.

**Debug**

- Fixed crash if something is entered in the disassembly window's expression input when not debugging.
- Word and half-word writes to SWO channel 0 are now displayed in the debug terminal.
- Avoid update of live watch window during J-Link download.
- Avoid deletion of breakpoints when disconnecting from J-Link.
- Improve debug when compiler option "Supply Absolute File Path" is set to "No".
- Fixed crash when quick watch used on a C++ struct containing member functions.
Enhanced Threads window to be able to display additional RTOS queues.
Fixed download progress bar showing negative ETA and 0.0 KB total download size on Linux and macOS versions.
Add support for TargetInterface.runFromAddress in target script.
Fixed inability to add items to the watch window when periodic update is enabled.
Edited source files will not be used by the debugger.
Updated shipped J-Link software to V6.32c.

Editor

The Text Editor > Programmer Assistance > Check Spelling environment option now defaults to No.
Fixed code formatting of a selected block not working correctly when the Tab Cleanup On Save option is enabled.
Fixed NULL being inserted into text file when CTRL+Space is pressed on Linux version of the code editor.
Added replace in selection to the find dialog’s find options summary.
Fixed freezing of IDE when saving a large text file and the Delete Trailing Space On Save option is enabled.
Improved performance of tabify and untabify operations.
Cursor now moved if it is on a location that is deleted by the code formatter.
Added Formatting > Empty Lines At End Of File option.
Fixed Find Extras context menu not correctly showing the text that will be searched for.
Added Text Editor > Formatting > Use .clang-format File formatting option.
Auto comment no longer activates when in block selection mode.
Added Text Editor > International > Auto-Detect UTF-8 option.
Fixed text editor tooltip not working when word starts on first column.
Fixed goto definition not working correctly when definition has been selected.
Tab key now indents only if selection is multi-line.
Find in files dialog can now be opened using Ctrl+Shift+F from the incremental find dialog.

IDE

Added missing close button on code editor find dialog.
Pasting of a block selection is now done as a block insertion even if the text editor has lost focus.
Add environment option to enable the text editor to display section headers of ELF files.
Fixed generation of unexpected characters when composing a character with ` ' or ^ keys.
Reinstated default ALT-F5 keyboard mapping to DebugStartOzone command.
Fixed crash when using a display with a 16-bit color depth.
Fixed HUD windows not closing when all docked windows have been closed or removed.
Fixed crash when changing active projects while Source Navigator is running.
Check boxes in project system dialogs can now be toggled by a single click.
Fixed menu key not opening context menu in watch and register windows.
Improved appearance of list view check boxes when using display scaling.
Property editor dialogs can now be resized.
Added "Debug Entry Point" debugging option.
Remove .plist files created by clang static analyzer.
Fixed long window captions in some popup windows.
Fixed find window's file extension filter not being saved when using Find Extras options.
Fixed find window's additional options summary not being displayed when options are concealed.
Find window's additional options summary now includes file extension filter.
Full file path now displayed in find window's result list.
Fixed ordering of history in package release notes.
Fixed potential problem when multiple processes are accessing settings.
Fixed unresponsive GUI when build generates a lot of output.
Added File Search > Flat Search Result Output environment option.
The file search window now defaults to flat result output and the dock site at the bottom of the main window.
Fixed incorrectly located line edit in list views.
Fixed activation of keyboard accelerators from HUD windows.
Fixed automatic installation of packages when clicking on documentation links.
Fixed debug terminal find only carrying out search once.
Fixed bookmarks window updating bookmark line numbers when lines were inserted into or deleted from a different file.
Fix command line supplied to clang static analyzer for ARM/Cortex-A/Cortex-R devices.
Fixed opening of project files when the File > File Open Action property is set to Web Browser.
Popup error message now displayed if package list cannot be downloaded when refreshing.
Added File Search > Collapse Results environment option.
Fixed crash when refreshing the outline window while editing an empty XML file.
Fixed setting propertyGroup defined properties on folder nodes when importing package files.
Keyboard map is now saved with UTF-8 encoding.
Web browser links to text files now open in web browser rather than text editor.

Installer

Fixed crash when running on an Ubuntu 14.04 system using KDE window manager.
Fixed Windows installer failure when user name contains certain non-ASCII characters.
Fixed loss of icons and inability to start IDE from desktop if umask has been set preventing read and execute permissions from being enabled for others (Linux only).

Version 3.34

Build

Added EmbeddedStudioVersionNumber system macro.
The inttypes.h header file now includes stdint.h as per the c99 standard.

Added "Math Errno" project option.

Dependency files are now deleted on project rebuild/clean.

Does not link Externally Built Executable projects that have a specified Load File.

Does not compile files in Externally Built Executable projects that do not have compile command(s).

### Debug

J-Link Control Panel window now works with multiple J-Links attached.

Fixed generation of symbols when address_symbol and size_symbol attributes are used in a memory map or section placement files.

Fixed crash with IAR v8 generated elf files.

Fixed usage of brackets in debug watch expressions.

Additional load files can be relative to the project directory.

Fixed crash when scrolling the disassembly window with the mouse wheel when debugger is not running.

Documented the file format for the “Type Interpretation File” project property.

Updated shipped J-Link software to V6.22d.

### IDE

Fixed renaming of build configurations not applying when clicking on another build configuration after change.

Fixed crash when pressing Ctrl+Shift+T.

Modified macOS text editor cursor key mapping to be more like other macOS text editors.

Double clicking on company logo images in package manager and new project wizard now has no effect.

Fixed update of registers window status message when no registers are selected.

Fixed display of multi-line messages in output window’s task view.

Code editor suggestions now inserted on all lines when in block edit mode.

Code editor replace all now only replaces text within block when in block selection mode.

WebKit web browser now uses display scaling factor.

**Show Large Icons In Toolbars** option now applies to docking windows.

Added keyboard shortcut editor to environment options dialog.

Fixed crash that occured when cancelling the new project wizard when on the edit common project settings page.

Fixed code editor suggestion popup not restoring opacity when ctrl key is released.

Pasting of a block selection is now done as a block insertion even if text editor is not in block edit mode.

Fixed text terminal not staying at end of file when maximum line limit is reached.

### Licensing

Fixed wireless network interfaces not being included in list of network adapters on Windows.
Version 3.30

Build

- Added missing __powidf2 and __powisf2 compiler helper functions.
- Updated the LLVM/Clang tools build to use the 5.0.0 source release.

Debug

- Updated shipped J-Link software to V6.20d.
- Added missing JLinkDevices.xml file and accompanying flash loaders.

IDE

- Environment option descriptions are now selectable.
- Improved performance of text editor brace matching in long files.
- Added Text Editor > Formatting > Additional Formatting Styles environment option.
- Added case sensitivity, whole word and regular expression options to code edit's incremental search dialog.
- Code editor's incremental search dialog no longer resets find dialog settings.
- Fixed drag and drop of file onto a project explorer file node from an external program.
- Fixed loss of focus when an expanded project explorer node is deleted.

Licensing

- Fix macro expansion of "Target Device" on license check.
- Do license check when active project is changed.
- License check now applies to "Library" project types based on the "Core Type".
- Fixed installation of license strings containing UNICODE characters.

Version 3.28

Build

- Fixed running of build command lines containing a ‘>' output redirection character.
- Fixed command line property editor.
- Added static_assert definition to assert.h.
- Fixed crash when Folder Options node is selected in project explorer and Open Solution in Editor is activated.
- Dynamic macros are now expanded in property editor macro preview.
- Added RelInputDir and FolderPath macros.
Added empty solution project template.

**Debug**

- Fixed crash using Debug | Restart with the simulator target before startup breakpoint is hit.
- Fixed crash using Target | Attach Debugger with J-Trace target.
- Fixed bug in backtracing code which caused local variables to be displayed incorrectly.
- Local, global, auto and watch window columns are now independently configurable.
- Fixed display of signed 32-bit integer variables on 64-bit Linux and macOS hosts.
- Fixed Cortex-M simulator return from exception when using both main and process stack.
- Speed up single stepping of large programs when there are many unfound symbols in watch window.
- Add "Auto" capability to SWO baud rate project property.
- Updated shipped J-Link software to V6.20a.
- Changed values in variable and register windows are now identified by red text rather than a red background.
- Avoid memory and watch window update during download.
- Improve speed of disassembly when source files cannot be found.

**IDE**

- Fixed opening of files using macOS Finder.
- Improved opening of files from the command line.
- Fixed crash in text editor when moving left a word at a time.
- Reinstated View > Dashboard menu entry.
- File extension comparisons for project, package and archive files are now case insensitive.
- External diff tool runs as a detached process.
- Fixed error reporting when source file couldn't be opened by IAR compiler.
- Fixed crash in code outline window when viewing an XML file containing a syntax error.
- Add "Copy Full Path" to context menu in project explorer.
- Find in files on solution will not search object and library files.
- Fixed crash reporter hanging if report submission fails.

**Version 3.26**

**Build**

- Updated the GCC/BINUTILS tools build to use GNU ARM Embedded Toolchain 6-2017-q2-update source release.
- Updated LLVM/Clang to version 4.0.1.
- C runtime start code now has an optional call to an external function named InitializeUserMemorySections if INITIALIZE_USER_SECTIONS is defined.
The 'Hello' first run sample project now defaults to the simulator target connection.

**Debug**

- Fix bug locating to assembly code source files.
- Locals window update when accessing variables that are in restricted memory ranges.
- Stop accessing address zero on debug reset on Cortex-M devices.
- Add "Starting Stack Pointer Value" debug project property.
- Updated shipped J-Link software to V6.18.
- Added **Debug > Debug With Ozone** menu option and an environment option to override the default location of the Ozone executable.

**IDE**

- Fixed crash showing symbols browser for IAR generated executables.
- Opening studio from shortcut when **Allow Multiple Studios** is set to **No** and studio is already running now unminimizes and raises main window to the top.
- Can now close the solution whilst the Source Navigator or Find References windows are active.
- Statistics in the Project Explorer displays the sum of the files sizes of the containing folder node.
- Improved error message reporting when studio startup fails.
- Check syntax option is now enabled on files with .html extension.
- Find extras menu order now the same in context menu as it is in the main menu.
- Fixed installation failure when installing certain packages.
- Added new look dashboard window.
- Fixed certain Alt key accelerators not working on macOS.
- Macro viewer in property editor now has horizontal scroll bar.
- Fixed blank entries in proprety editor's build configuration combo box (macOS only).
- Hyperlinks in property editor's description fields now open in an external browser.
- Highlighted finds are now local to each text editor.
- Fixed text editor match delimiter and extend selection operation (Shift+Ctrl+}).
- Fixed moving of popup windows displayed when project is loading.
- Fixed Delete key not deleting selected text when cursor is at the end of the file.
- Fixed crash running installer on Linux when KDE plugins are installed on the system.
- Windows version no longer requires the Universal C Runtime update to be installed.
- Smaller up and down buttons in property window.
- Shift+Tab now works when text editor is in block selection mode.
- Indent when text editor is in block selection mode now aligns to the indent size setting.
- Highlight all selected text now works when text is selected by keyboard.

**Licensing**

- Add environment option "Check J-Link For Licenses".
Version 3.24

Build

Text files shipped in Windows distributions are now in DOS format.
Reworked compiler driver command line options.

Debug

The debug restart action doesn't load the target.
Holding the shift key while scrolling the memory window with the mouse scroll wheel now locks the start address.
Added Debug > Memory Window > Scroll Wheel Modifies Start Address environment option.
Memory window size warning can be disabled and is now only shown when the size changes.
Fixed crash when the memory diff dialog is shown after download verification has failed.
Updated shipped J-Link software to V6.16g.

IDE

Editor dock positions are now restored when solution is loaded.
Fixed incorrect calculation of memory usage window cell height when using high DPI fonts.
Fixed crashes caused by uncaught exceptions (Linux only).
Fixed incorrect width of editor margin when using Windows scaling.
Fixed pressing tab key while in block edit mode.
Fixed occasional randomly placed tooltips in code editor.
Added Text Editor > Editing > Tab Key Indents Preprocessor Directives environment option.
Fixed text editor crash when selecting and deleting past end of file with virtual space enabled.
Fixed text editor scrolling to the far left column when text is selected and the mouse is moved.
The Code Outline window now uses the same parser as the Source Navigator this has improved C++ support but has removed conditional preprocessor directives.
The Code Outline window can now display doxygen style comments in the Preview pane.

Version 3.22

Build

Change default for "Omit Frame Pointer" to "Yes".
Added "Export Makefile" to project build context menu.

Debug

Removed license check from memory window load and save operations.
Embedded Studio now only uses the J-Link software it ships with, it no longer searches for other installations.
Updated shipped J-Link software to V6.16c.
Fixed crash when auto disconnecting simulator before simulator has stopped.

**IDE**

Fixed text editor always using proportional font rendering irrespective of the Text Editor > Visual Appearance > Font Rendering option.
Fixed text editor cursor color when using CrossWorks Dark color scheme.
Added Insert Cursor and Overwrite Cursor colors to editor color schemes.
Fixed text editor repaint when highlight cursor line mode is enabled.
Fixed display of large toolbar icons.
Fixed activation of popup toolbar icons.
Fixed path property editor when using scaling on Windows.
The text editor line number font size now scales with the main text editor font size.
Project properties dialog now remembers splitter placement.
Improved support for Windows scaling.
Fixed code completion suggestion popup appearing on the wrong display on multi-display systems.
Fixed Edit > Selection > Tabify.
Added text editor block selection and edit.
Fixed File > Recent Files and File > Recent Projects not selecting first element of menu when activated by keyboard.
Fixed Command-W not closing current editor on macOS.
Improved macOS clipboard support.
Avoid auto loading externally modified project file during build.
Grey out goto definition (and others) when indexer is running.
Fixed window group Close All Windows option not recording in session file that windows have been closed.
Fixed excessively fast vertical scroll wheel scrolling in text output windows.

**Version 3.20**

**Build**

Change default for "Emit Relocations" to "Yes".
Add weak attribute to library realloc implementation.
Added sys/stat.h, sys/time.h and sys/types.h for compatibility.
Removed STLPrt from the distribution. This is available as a library package.
Add "Post-Archive Command" project options.
Generic startup code now enables VFP when "ARM FP ABI Type" is not set to "Soft".
RTT files are not added to new ARM/Cortex-A/R generic projects.
Add "Pre-Build Command" and "Post-Build Command" project options.
Default __putchar and __getchar functions are now implemented using debugio.  
Upgraded LLVM/Clang to version 4.0.0.  
Updated the GCC/BINUTILS tools build to use GNU ARM Embedded Toolchain 6-2017-q1-update source release.  
Changed "Printf Floating Point Supported" project option to select between "Float" and "Double" support.

Debug

Fixed crash when downloading non-existent externally built executable.  
Fixed crash when displaying class variables that contain static member data of the same type.  
Added "Copy To Clipboard" to register and variable display windows.  
Added "Copy To Clipboard" to memory window. 
Single stepping will step again if there is more then one instruction sequence associated with a source line of code. 
Disassembler decodes tbb/tbh pc relative jump tables.  
Updated shipped J-Link software to V6.14h.  
The "Auto Update" feature of the execution profile window uses the j-trace pro streaming feature.  
Added locate next/prev source/instruction buttons to execution trace window.  
Added function call and return entries to execution trace window.

IDE

Added "Stack Usage" window.  
Changed "Build Macros" environment option name to "Global Macros".  
Fixed missing bottom line of some characters in Linux version of code editor.  
Clicking on studio icon in macOS dock now unminimizes IDE if it is minimized.  
Add up and down project navigation buttons to project properties dialog.  
Fixed reload solution not loading the correct solution if multiple versions of the studio are running with different projects loaded.  
Added Text Editor > Save > Default Line Endings environment option.  
Fixed text editor asking whether it is OK to overwrite an existing file twice.  
Editor now preserves UTF-8 byte order marks.  
Added different bitmaps to the project window for executable, library and staging project types.  
Build detects when files have been excluded/included and cut/pasted into projects.  
Project explorer paste file onto file will add it to the containing folder.  
Fixed wait parameter on JavaScript function "CWSys.run".  
Fixed text editor crash when using delete while cursor is at the end of file.  
Fixed integer property editor not working correctly on Linux and macOS.  
Fixed text editor Tools > Format Code losing last line of file on Linux and macOS.  
Fixed up and down cursor keys not working correctly with search filters on macOS.  
Fixed naming of Formatting > Always Break After Definition Return Type project property.
Fixed duplicate Debug and Debugging property groups.

*Added* Nordic Semiconductor text formatting style.

The source browser window has renamed the "Stack" column to "Frame Size".

*Added* "Code", "Data" and "Const" size columns to the source browser window.

**Licensing**

The ARM license entitlement is now checked against the "Target Device" project property.

**Version 3.12**

**Build**

The names of temporary files produced by the compiler are now derived from the input filename.

The rm and chmod commands generated by the build are now implemented as internal commands.

*Added* "V8M Has CMSE Instructions" project property.

*Added* "CMSE Import Library File" project property.

Fixed crash caused by clearing build log whilst building.

Fixed ld executable being loaded into an editor on memory segment overflow linker errors (Linux and macOS only).

Fixed build for project Link and unity folder Compile.

Fixed setting strict-aligned clang compiler option for v4t, v5te, v6m and v8m.baseline architectures.

Removed broken linker variant project property.

Fixed C++ signbit implementation.

Fixed build not building newly imported files.

Changed linker and archiver "Use Indirect File" options default.

Fixed using indirect files with filenames containing backslashes.

Show "Batch Build Configurations" property at the solution node.

Fixed crossscript crash when using ElfFile.peekBytes or ElfFile.crc32 functions on a program compiled with the Clang compiler.

Fixed importing IAR EWARM/Keil MDK project files containing dots.

*Added* linker option "Symbols File" to supply a symbols only file to the link command.

*Added* "None" option to "Debugging Level" which is now used for "Release" configuration builds.

Updated the GCC/BINUTILS tools build to use GCC ARM Embedded 6-2016-q4-major source release.

Upgraded LLVM/Clang to version 3.9.1.

*Add* int sized enums library build variants.

Add big endian library build variants for architecture v6M.

*Add* "Library ARM Architecture" project property.

Add environment option "Enable All Warnings Command Line Options".

Add environment option "Enforce ANSI Checking Command Line Options".

Change default for "Allow Multiple Symbol Definition" to "No".
Debug

Added "Auto Refresh" to the context menu of the execution count window.
Fixed set breakpoint on variable from text editor.
Fixed modifying breakpoint properties.
Fixed crash when pressing page down on last page of the disassembly in the disassembly window.
Fixed crash using debug restart with the simulator target.
Add "Ignore .debug_frame Section" debugging property.
Fixed crash if memory write fails during download on OS X.
Fixed crash when connecting to J-Link target interface when no J-Link is attached.
Fixed disassembly of ldr literal instructions.
Improved debug support of clang generated code.
Fixed memory window always evaluating address expression when auto size is selected.
Fixed crash when opening memory window.
Fixed watch window not storing the filename and linenumber context in which to evaluate the expression.
Fixed misc bugs in watch window.
Fixed Cortex-M simulator handling of word aligned stack pointers on exception return.
Fixed crash using debug stop during download.
Control bit zero now shown in CPU registers display for Cortex-M0.
Added environment option "Show Member Functions" to variable windows to enable C++ member functions to be displayed.
Add "RTT Enable" and "RTT Control Block Address" debugging options.
Fixed crash when setting register allocated variables that are less than 4 bytes in length.
Add "ITM Stimulus Port To Display" debugging options.
Fixed implementation of rbit instruction in ARM simulator.
Updated shipped J-Link software to V6.12f
Add "Access Variables Within Memory Map Only" project property.
Remove "Leave Target Running" project property.

IDE

Parallel building of files in projects now shows a progress bar and ETA, both of these can be disabled using new environment options.
Fixed Microsoft IME support in code editor.
Removed project property "Build Dependents in Parallel", replaced it with "Project Can Build In Parallel".
Fixed saving of files from code editor using Shift-JIS encoding.
Improved appearance of build log's memory usage summary when there are many memory segments.
Fixed code completion on files with paths containing UNICODE characters.
Fixed goto definition and find references on files with paths containing UNICODE characters.
File and project history now shared between all major releases.
Fixed save prompt dialog not being displayed when modified file is closed.
Fixed close solution not stopping when the saving of a modified file has failed.
Improved appearance of debugger tooltips.
Fixed build summary not being fully visible when build log is larger than window.
Fixed directory chooser on IAR/Keil project import dialog.
Fixed crash when entering address in disassembly window that is not within the current visible address range.
Fixed text editor slow down when rendering very long lines.
Find dialog auto complete is now case sensitive.
Enter key now behaves the same as return key in find dialog.
Fixed crash when dragging a project folder onto one of its own sub folders.
Register window bitfield entries now use monospace font.
Ctrl+C and Ctrl+A now work in project property dialog’s description field.
Fixed drag not working in project explorer on new files or folders until project has been reloaded.
Fixed use of proportional fonts in code editor.
Added Text Editor > Font Rendering environment option.
Cannot close solution while Source Navigator and References window are active.
Added stop button to Source Navigator window.
Added coloring for #if’ed out code to the editor for C and C++ files. This is computed on file load and save.
Added .svd file type detection to "Register Definition File" property.
Added environment option "Parallel Project Building" to enable dependent project (and solution) builds to be done in parallel.
Fixed text editor cursor not being visible over text drawn with a different background colour.
Fixed the window group context menu sometimes being displayed higher than the mouse pointer.
Search box now focused when package manager is opened.
Fixed project importer not loading shipped jlink.dll when no other is available.
Fixed pasting of file into a project explorer folder that is already contained within that folder.
Fixed file overwrite warning dialog not giving the option to cancel.
Added Title Bar > Show Full Solution Path environment option.
Fixed crash when closing all editor windows, right clicking and then moving the mouse with the empty dock window.
Added tooltips on symbols in text editor.
Fixed editor search not clearing highlights when search string is cleared.
Added Text Editor > Highlight All Selected Text environment option.
Added Windows > Debug Terminal > Backscroll Buffer Lines environment option.
Added Text Editor > Programmer Assistance > Inactive Code Opacity environment option.
Fixed incorrectly placed resize grip in property editor.
Fixed previous search not being selected when starting a multi-line find and replace.
Fixed bookmark not being removed from bookmarks window when the line it is on has been deleted.
New generic projects now set the "Section Placement Segments" option rather than use memory map files.
Added new generic projects that enable use of an external GNU toolchain.
Fixed selection in text editor not being completely cleared under certain circumstances.
Find and replace window now remembers last search context and file type settings.
Add support for BE-8 executables.
Fixed opening of example projects from dashboard and contents window.
Added dependent files to quick open (Ctrl+o) editor action.
Help system now uses WebKit web browser to display content.
Brace matching now takes into account inactive code lines.
Inactive code highlighting now updates as you type.
Detects project file has changed on disk and prompts for reload.
Function keys can now be displayed permanently on OS X Touch Bar when IDE is running. See Using function keys on MacBook Pro with Touch Bar for more information on how to do this.
Removed "Target" option from Generic project types and added "Target Device" option to enable target device selection.
Reintroduce tools on project explorer context menu for folder and file nodes.
Added "Stack" column to the symbol browser window.

Licensing

The product build date is now compared with the SUA expiry date. The License manager displays licenses that have expired SUA’s.

Version 3.10

Build

Added "Keep Linker Script File" project property.
Added "Tool Chain Directory" project property.
Added "Externally Built Library" to the "Project Type" enumeration.
Added "Section Placement Segments" project property.
Added "Post-Link Output File" project property.
Added "C Compile Command", "C++ Compile Command", "Assemble Command", "Link Command" and "Archive Command" project properties to enable user defined build steps.
Added capability to build "Externally Built Executable" and “Externally Built Library” project types using command lines in project properties.
Supplied ascii only version of ctype functions in ctype_no_wchar.c in the $(StudioDir)/source directory.
Supplied non threaded version errno in errno_no_thread.c in the $(StudioDir)/source directory.
Fixed 0.0 >= 0.0 and 0.0 <= 0.0 not returning true on some architectures.
Fixed crash when building libraries using indirect files with no files to archive.
Opening IAR EWARM/Keil MDK project files will now create internal and external build configurations.
The external build configuration will use the IAR/Keil tools. The internal build configuration will use the internal tools.
Fixed setting errno to EDOM with invalid arguments to acos(f), asin(f) and fmod(f).
Fixed setting errno to ERANGE when overflow occurs with ldexp(f).
Updated the GCC/BINUTILS tools build to use GCC ARM Embedded 5-2016-q2-update source release.
Link will now fail with an error if a section has been placed at the end of a segment using the
place_from_segment_start attribute and the preceding section overlaps it.

Debug

Removed "J-Link DLL File" project property.
Added "Use Built-in RTT support" project property to enable RTT to be used on all devices.
Add "Ignore .debug_ranges Section" debugging property.
Fixed crash while using memory window when not connected to target.
Added "Load ELF Sections" loader project property.
Added "Stop On Memory Error" simulator project property.
Removed "Debugger Initial Breakpoint" environment options.
Can now optionally specify the main load file to download using the "Load File" project property.
Added "Go To Disassembly" to code editor context menu when debugging.
Added "Export As Text" to variable display windows.
Variable display windows now display char * as null terminated strings by default.
Variable display windows now carry out numerical sort when sorting on size.
Fixed watch window not moving blank entry to end when using add to watch from editor.
Adding simulator project property "Stop On Memory Error".
Enabling of exception breakpoints is preserved in session file.

IDE

Project properties dialog now defaults to the active build configuration when opened.
Fixed crash when dragging a project folder onto itself.
Fixed display of string properties that contain line feeds.
The "Project Type" property can now be set on a per configuration basis.
Fixed reload project not working correctly when the project has been opened from the Recent Projects window.
Project properties editor dialog is now modal.
Memory window address field now expands to fill available space.
Fixed blank filenames in Open File From Solution dialog.
Fixed crash when right clicking in empty area of build configuration dialog.
Fixed crash when changing configuration with a property editor focused in properties dialog.
Changed the way modified and inherited properties are shown in the properties dialog/window.
Code editor no longer displays file modified warning if file has been externally deleted.
Properties dialog, removed "All" group, deselecting the group/page will show all properties.
Properties dialog, changed the graphic to indicate that a project property has been modified or is set in another node or configuration.
Properties dialog, removed the build macros and added a means to display the set of macros on individual property editors.

Project explorer, added "Exclude From Build" option to folders.

Add support for ARMv8-M Mainline and ARMv8-M Baseline architecture.

Properties dialog, added option to show modified properties only.

Fixed drag and drop in project explorer only working on a new folder after project has been reloaded.

Project explorer removed automatic usage of memory map, section placement and linker script files in the build.

Added "Use This Memory Map", "Use This Section Placement" and "Use This Linker Script File" to the appropriate files in the project explorer context menu.

Files of type Memory Map, Section Placement and Linker Script that are added to the project now prompt to update the appropriate project property.

Added "Use Manual Linker Script" and "Linker Script File" project properties.

Removed Calcpad, Debug Immediate, Downloads, Properties, Script Console and Terminal Emulator windows.

Reorganised menu entries, all window show actions are available on the View menu.

File path property editor now applies change when enter key is pressed.

The Application Monospace Font property editor will now only allow monospace fonts to be selected.
SEGGER Embedded Studio User Guide

This is the user guide for the SEGGER Embedded Studio integrated development environment (IDE). The SEGGER Embedded Studio IDE consists of:

- A project system to organize your source files
- A build system to build your applications
- Programmer aids to navigate and work effectively
- A target programmer to download applications into RAM or flash
- A debugger to pinpoint bugs
SEGGER Embedded Studio standard layout

SEGGER Embedded Studio's main window is divided into the following areas:

- **Title bar**: Displays the name of the current solution.
- **Menu bar**: Menus for editing, building, and debugging your program.
- **Toolbars**: Frequently used actions are quickly accessible on toolbars below the menu bar.
- **Editing area**: A tabbed view of any open editor windows and the HTML viewer.
- **Docked windows**: SEGGER Embedded Studio has many windows that dock to the left, right, or below the editing area. You can configure which windows will be visible, and their placement, when editing and debugging.
- **Status bar**: At the bottom of the main window, the status bar contains useful information about the current editor, build status, and debugging environment.
Menu bar

The menu bar contains menus for editing, building, and debugging your program. You can navigate menus using the keyboard or the mouse.

Navigating menus using the mouse

To navigate menus using the mouse:

1. Click a menu title in the menu bar to show the related menu.
2. Click the desired command in the menu to execute that command.

or

1. Click and hold the mouse on a menu title in the menu bar to show the related menu.
2. Drag the mouse to the desired command in the menu.
3. Release the mouse while it is over the command to execute that command.

Navigating menus with the keyboard

To navigate menus using the keyboard:

1. Tap the Alt key activate the menu bar.
2. Tap Return to display the menu.
3. Use the Left and Right keys to select the required menu.
4. Use the Up or Down key to select the required command or submenu.
5. Press Enter to execute the selected command.
6. Press Alt or Esc at any time to cancel menu selection.

After you press the Alt key once, each menu on the menu bar has one letter underlined its shortcut key. So, to activate a menu using the keyboard:

While holding down the Alt key, type the desired menu's shortcut key.

After the menu appears, you can navigate it using the cursor keys:

Use Up and Down to move up and down the list of menu items.
Use Esc to cancel a menu.
Use Right or Enter to open a submenu.
Use Left or Esc to close a submenu and return to the parent menu.
Type the underlined letter in a command's name to execute that command.
Title bar

The first item shown in the title bar is SEGGER Embedded Studio's name. Because SEGGER Embedded Studio can be used to target different processors, the name of the target processor family is also shown, to help you distinguish between instances of SEGGER Embedded Studio when debugging multi-processor or multi-core systems.

The filename of the active editor follows SEGGER Embedded Studio's name; you can configure the presentation of this filename as described below.

After the filename, the title bar displays status information on SEGGER Embedded Studio's state:

- **[building]** SEGGER Embedded Studio is building a solution, building a project, or compiling a file.
- **[run]** An application is running under control of SEGGER Embedded Studio’s debugger.
- **[break]** The debugger is stopped at a breakpoint.
- **[autostep]** The debugger is single stepping the application without user interaction (*autostepping*).
Status bar

At the bottom of the window, the status bar contains useful information about the current editor, build status, and debugging environment. The status bar is divided into two regions: one contains a set of fixed panels and the other is used for messages.

The message area

The leftmost part of the status bar is a message area used for things such as status tips, progress information, warnings, errors, and other notifications.

Status bar panels

You can show or hide the following panels on the status bar:

<table>
<thead>
<tr>
<th>Panel</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target device status</td>
<td>Displays the connected target interface. When connected, this panel contains the selected target interface's name and, if applicable, the processor to which the target interface is connected. The LED icon flashes green when a program is running, is solid red when stopped at a breakpoint, and is yellow when connected to a target but not running a program. Double-clicking this panel displays the Targets pane, and right-clicking it invokes the Target shortcut menu.</td>
</tr>
<tr>
<td>Cycle count panel</td>
<td>Displays the number of processor cycles used by the executing program. This panel is only visible if the connected target supports performance counters that can report the total number of cycles executed. Double-clicking this panel resets the cycle counter to zero, and right-clicking it brings up the Cycle Count shortcut menu.</td>
</tr>
<tr>
<td>Insert/overwrite status</td>
<td>Indicates whether the current editor is in insert or overwrite mode. In overwrite mode, the panel displays &quot;OVR&quot;; in insert mode, the panel displays &quot;INS&quot;.</td>
</tr>
<tr>
<td>Read-only status</td>
<td>Indicates whether the editor is in read-only mode. If the editor is editing a read-only file or is in read-only mode, the panel display &quot;R/O&quot;; if the editor is in read-write mode, the panel displays &quot;R/W&quot;.</td>
</tr>
<tr>
<td>Build status</td>
<td>Indicates the success or failure of the last build. If the last build completed without errors or warnings, the build status pane contains Built OK; otherwise, it contains the number of errors and warnings reported. If there were errors, double-clicking this panel displays the Build Log in the Output pane.</td>
</tr>
</tbody>
</table>
Caret position | Indicates the insertion position in the editor window. For text files, the **caret position** pane displays the line number and column number of the insertion point in the active window; when editing binary files, it displays the address being edited.

Time panel | Displays the current time.

---

**Configuring the status bar panels**

To configure which panels are shown on the status bar:

Choose **View > Status Bar**.

From the status bar menu, select the panels to display and deselect the ones you want hidden.

or

Right-click the status bar.

From the status bar menu, select the panels to display and deselect the ones you want to hide.

**To show or hide the status bar:**

Choose **View > Status Bar**.

From the status bar menu, select or deselect the **Status Bar** item.

You can choose to hide or display the **size grip** when SEGGER Embedded Studio's main window is not maximized. (The size grip is never shown in full-screen mode or when maximized.)

**To show or hide the size grip**

Choose **View > Status Bar**.

From the status bar menu, select or deselect the **Size Grip** item.
Editing workspace

The main area of SEGGER Embedded Studio is the editing workspace. It contains any files being edited, the on-line help system's HTML browser, and the Dashboard.
Docking windows

SEGGER Embedded Studio has a flexible docking system you can use to position windows as you like them. You can dock windows in the SEGGER Embedded Studio window or in the four head-up display windows. SEGGER Embedded Studio will remember the position of the windows when you leave the IDE and will restore them when you return.

Window groups

You can organize SEGGER Embedded Studio windows into window groups. A window group has multiple windows docked in it, only one of which is active at a time. The window group displays the active window’s title for each of the windows docked in the group.

Clicking on the window icons in the window group’s header changes the active window. Hovering over a docked window’s icon in the header will display that window’s title in a tooltip.

To dock a window to a different window group:

- Press and hold the left mouse button over the title of the window you wish to move.
- As you start dragging, all window groups, including hidden window groups, become visible.
- Drag the window over the window group to dock in.
- Release the mouse button.

Holding Ctrl when moving the window will prevent the window from being docked. If you do not dock a window on a window group, the window will float in a new window group.

Perspectives

SEGGER Embedded Studio remembers the dock position and visibility of each window in each perspective. The most common use for this is to lay your windows out in the Standard perspective, which is the perspective used when you are editing and not debugging. When SEGGER Embedded Studio starts to debug a program, it switches to the Debug perspective. You can now lay out your windows in this perspective and SEGGER Embedded Studio will remember how you laid them them out. When you stop debugging, SEGGER Embedded Studio will revert to the Standard perspective and that window layout for editing; when you return to Debug perspective on the next debug session, the windows will be restored to how you laid them out in that for debugging.

SEGGER Embedded Studio remembers the layout of windows, in all perspectives, such that they can be restored when you run SEGGER Embedded Studio again. However, you may wish to revert back to the standard docking positions; to do this:
Dashboard

When SEGGER Embedded Studio starts, it presents the Dashboard, a collection of panels that provide useful information, one-click loading of recent projects, and at-a-glance summaries of activity relevant to you.

Tasks

The Tasks panel indicates tasks you need to carry out before SEGGER Embedded Studio for ARM is fully functional—for instance, whether you need to activate SEGGER Embedded Studio for ARM, install packages, and so on.

Updates

The Updates panel indicates whether any packages you have installed are now out of date because a newer version is available. You can install each new package individually by clicking the Install button under each notification, or install all packages by clicking the Install all updates link at the bottom of the panel.

Projects

The Projects panel contains links to projects you have worked on recently. You can load a project by clicking the appropriate link, or clear the project history by clicking the Clear List button. To manage the contents of the list, click the Manage Projects link and edit the list of projects in the Recent Projects window.

News

The News panel summarizes the activity of any RSS and Atom feeds you have subscribed to. Clicking a link will display the published article in an external web browser. You can manage your feed subscriptions to by clicking the Manage Feeds link at the end of the News panel and pinning the feeds in the Favorites window you are only subscribed to the pinned feeds.

Links

The Links panel is a handy set of links to your favorite websites. If you pin a link in the Favorites window, it appears in the Links panel.
SEGGER Embedded Studio help and assistance

SEGGER Embedded Studio provides context-sensitive help in increasing detail:

**Tooltips**
When you position the pointer over a button and keep it still, a small window displays a brief description of the button and its keyboard shortcut, if it has one.

**Status tips**
In addition to tooltips, SEGGER Embedded Studio provides a longer description in the status bar when you hover over a button or menu item.

**Online manual**
SEGGER Embedded Studio has links from all windows to the online help system.

The browser

Documentation pages are shown in the **Browser**.

Help using SEGGER Embedded Studio

SEGGER Embedded Studio provides an extensive, HTML-based help system that is available at all times.

**To view the help text for a particular window or other user-interface element:**
- Click to select the item with which you want assistance.
- Choose **Help > Help** or press F1.

Help within the text editor

The text editor is linked to the help system in a special way. If you place the insertion point within a word and press F1, the help-system page most likely to be useful is displayed in the HTML browser. This a great way to quickly find the help text for functions provided in the library.

Browsing the documentation

The **Contents** window lists all the topics in the SEGGER Embedded Studio for ARM documentation and gives a way to search through them.

The highlighted entry indicates the current help topic. When you click a topic, the corresponding page appears in the **Browser** window.
The **Next Topic** and **Previous Topic** items in the **Help** menu, or the buttons on the **Contents** window toolbar, help navigate through topics.

To search the online documentation, type a search phrase into the **Search** box on the **Contents** window toolbar.

**To search the online documentation:**

Choose **Help > Contents** or press Ctrl+Alt+F1.

Enter your search phrase in the **Search** box and press **Enter** (or **Return** on Macs).

The search commences and the table of contents is replaced by links to pages matching your query, listed in order of relevance. To clear the search and return to the table of contents, click the clear icon in the **Search** box.
Creating and managing projects

A SEGGER Embedded Studio project is a container for everything required to build your applications. It contains all the assorted resources and maintains the relationships between them.

A project is a convenient place to find every file and piece of information associated with your work. You place projects into a solution, which can contain one or more projects.

This chapter introduces the various parts of a project, shows how to create projects, and describes how to organize the contents of a project. It describes how to use the Project Explorer and Project Manager for project-management tasks.
Solutions and projects

To develop a product using SEGGER Embedded Studio, you must understand the concepts of projects and solutions.

A project contains and organizes everything you need to create a single application or a library.

A solution is a collection of projects and configurations.

Organizing your projects into a solution allows you to build all the projects in a solution with a single keystroke, and to load them onto the target ready for debugging.

In your SEGGER Embedded Studio for ARM project, you

organize build-system inputs for building a product.
add information about items in the project, and their relationships, to assist you in the development process.

Projects in a solution can reside in the same or different directories. Project directories are always relative to the directory of the solution file, which enables you to more-easily move or share project-file hierarchies.

The Project Explorer organizes your projects and files, and provides quick access to the commands that operate on them. A toolbar at the top of the window offers quick access to commonly used commands.

Solutions

When you have created a solution, it is stored in a project file. Project files are text files, with the file extension emProject, that contain an XML description of your project. See Project file format for a description of the project-file format.

Projects

The projects you create within a solution have a project type SEGGER Embedded Studio uses to determine how to build the project. The project type is selected when you use the New Project dialog. The available project types depend on the SEGGER Embedded Studio for ARM variant you are using, but the following are present in most SEGGER Embedded Studio for ARM variants:

Executable: a program that can be loaded and executed.
Externally Built Executable: an executable that is not built by the SEGGER Embedded Studio for ARM internal build process.
Library: a group of object files collected into a single file (sometimes called an archive).
Externally Built Library: a library that is not built by the SEGGER Embedded Studio for ARM internal build process.
Object File: the result of a single compilation.
Staging: a project that will apply a user-defined command to each file in a project.
Combining: a project that can be used to apply a user-defined command when any files in a project have changed.

Project options and configurations

Project options are attached to project nodes. They are usually used in the build process, for example, to define C preprocessor symbols. You can assign different values to the same project option, based on a configuration: for example, you can assign one value to a C preprocessor symbol for release build and a different value for a debug build.

Folders and Dynamic Folders

Projects can contain folders, which are used to group related files. Automated grouping uses the files' extensions to, for example, put all .c files in one folder, etc. Grouping also can be done manually by explicitly creating a file within a folder. Note that these project folders do not map onto directories in the file system, they are used solely to structure the display of content shown in the Project Explorer.

Projects can also contain dynamic folders which will show the directories and files contained in the file system in the project explorer. You can specify if the dynamic folder is recursive and use wildcards to include and exclude files.

Source files

Source files are all the files used to build a product. These include source code files and also section-placement files, memory-map files, and script files. All the source files you use for a particular product, or for a suite of related products, are managed in a SEGGER Embedded Studio project. A project can also contain files that are not directly used by SEGGER Embedded Studio to build a product but contain information you use during development, such as documentation. You edit source files during development using SEGGER Embedded Studio's built-in text editor, and you organize files into a target (described next) to define the build-system inputs for creating the product.

The source files of your project can be placed in folders or directly in the project. Ideally, the paths to files placed in a project should be relative to the project directory, but at times you might want to refer to a file in an absolute location and this is supported by the project system.

When you add a file to a project, the project system detects whether the file is in the project directory. If a file is not in the project directory, the project system tries to make a relative path from the file to the project directory. If the file isn't relative to the project directory, the project system detects whether the file is relative to
the `$(StudioDir)` directory; if so, the filename is defined using `$(StudioDir)`. If a file is not relative to the project directory or to `$(StudioDir)`, the full, absolute pathname is used.

The project system will allow (with a warning) duplicate files to be put into a project.

The project system uses a file's extension to determine the appropriate build action to perform on the file:

- A file with the extension `.c` will be compiled by a C compiler.
- A file with the extension `.cpp` or `.cxx` will be compiled by a C++ compiler.
- A file with the extension `.s` or `.asm` will be compiled by an assembler.
- A file with the object-file extension `.o` will be linked.
- A file with the library-file extension `.a` will be linked.
- A file with the extension `.xml` will be opened and its file type determined by the XML document type.
- Files with other file extensions will not be compiled or linked.

You can modify this behavior by setting a file's File Type project option with the Common configuration selected, which enables files with non-standard extensions to be compiled by the project system.

**Externally Built Executables**

You can use an external build process for Externally Built Executable project types by setting the Build Command project option, for example to make target. Alternatively you can set command lines for specific build steps to compile/assemble and link. When you create an Externally Built Executable project type configurations will be created that create command lines for a variety of external tool chains.

**Solution links**

You can create links to existing project files from a solution, which enables you to create hierarchical builds. For example, you could have a solution that builds a library together with a stub test driver executable. You can link to that solution from your current solution by right-clicking the solution node of the Project Explorer and selecting Add Existing Project. Your current solution can then use the library built by the other project.

**Session files**

When you exit SEGGER Embedded Studio for ARM, details of your current session are stored in a session file. Session files are text files, with the file extension emSession, that contain details such as which files you have opened in the editor and what breakpoints you have set in the Breakpoint window.
Creating a project

You can create a new solution for each project or place multiple projects in an existing solution.

To create a new project in an existing solution:

1. Choose Project > Add New Project.
2. In the New Project wizard, select the type of project you wish to create and specify where it will be placed.
3. Ensure that Add the project to current solution is checked.
4. Click OK to go to next stage or Cancel to cancel the project's creation.

The project name must be unique to the solution and, ideally, the project directory should be relative to the solution directory. The project system will use the project directory as the current directory when it builds your project. Once complete, the Project Explorer displays the new solution, project, and files contained in the project. To add another project to the solution, repeat the above steps.

To create a new project in a new solution:

1. Choose File > New Project or press Ctrl+Shift+N.
2. Select the type of project you wish to create and where it will be placed.
3. Click OK.
Adding existing files to a project

You can add existing files to a project in a number of ways.

To add existing files to the active project:

Choose Project > Add Existing File or press Ctrl+P, A.

Using the Open File dialog, navigate to the directory containing the files and select the ones you wish to add to the project.

Click OK.

The selected files are added to the folders whose filter matches the extension of each of the files. If no filter matches a file's extension, the file is placed underneath the project node.

To add existing files to a specific project:

1. In the Project Explorer, right-click the project to which you wish to add a new file.
2. Choose Add Existing File.

To add existing files to a specific folder:

1. In the Project Explorer, right-click the folder to which you wish to add a new file.
2. Choose Add Existing File.

The files are added to the specified folder without using filter matching.

To create a dynamic folder:

1. In the Project Explorer, right click on the project to which you wish to add a new folder.
2. Choose New Folder....
3. Using the New Folder dialog name the folder and then show the dynamic folder options.
4. Specify the required Source Folder and the Filter Specification.

The files that match the filter specification in the source folder will appear in the newly created folder.
Adding new files to a project

You can add new files to a project in a number of ways.

To add new files to the active project:

Choose **Project > Add New File** or press **Ctrl+N**.

To add a new file to a project:

1. In the **Project Explorer**, right-click the project to which you wish to add a new file.
2. Choose **Add New File**.

When adding a new file, SEGGER Embedded Studio displays the **New File** dialog, from which you can choose the type of file to add, its filename, and where it will be stored. Once created, the new file is added to the folder whose filter matches the extension of the newly added file. If no filter matches the newly added file extension, the new file is placed underneath the project node.

To add new files to a folder:

1. In the **Project Explorer**, right-click the folder to which you wish to add a new file.
2. Choose **Add New File**.

The new file is added to the folder without using filter matching.
Removing a file, folder, project, or project link

You can remove whole projects, folders, or files from a project, or you can remove a project from a solution, using the Remove button on the Project Explorer toolbar. Note that removing a source file from a project does not remove it from disk.

To remove an item from the solution:

1. In the Project Explorer, select the item to remove.
2. Choose Edit > Delete or press Del.

or

1. In the Project Explorer, right-click the item to remove.
2. Choose Remove.
Building your application

SEGGER Embedded Studio builds your application using the resources and build rules it finds in your solution.

When SEGGER Embedded Studio builds your application, it tries to avoid building files that have not changed since they were last built. It does this by comparing the modification dates of the generated files with the modification dates of the dependent files together with the modification dates of the project options that pertain to the build. But if you are copying files, sometimes the modification dates may not be updated when the file is copied. In this instance, it is wise to use the Rebuild command rather than the Build command.

You can see the build rationale SEGGER Embedded Studio currently is using by setting the Environment Options > Building > Show Build Information environment option. To see the build commands themselves, set the Environment Options > Building > Echo Build Command environment option.

You may have a solution that contains several interdependent projects. Typically, you might have several executable projects and some library projects. The Project Dependencies dialog specifies the dependencies between projects and to see the effect of those dependencies on the solution build order. Note that dependencies can be set on a per-configuration basis, but the default is for dependencies to be defined in the Common configuration.

You will also notice that a new folder titled Dependencies has appeared in the Project Explorer. This folder contains the list of newly generated files and the files from which they were generated. To see if one of files can be decoded and displayed in the editor, right-click the file to see if the View command is available on the shortcut menu.

If you have the Symbols window open, it will be updated with the symbol and section information of all executable files built in the solution.

To generalize your builds, you can define macro values that are substituted when the project options are used. These macro values can be defined globally at the solution and project level, and can be defined on a per-configuration basis.

The combination of configurations, project options with inheritance, dependencies, and macros provides a very powerful build-management system. However, such systems can become complicated. To understand the implications of changing build settings, right-click a node in the Project Explorer and select Options to view a dialog that shows which macros and project options apply to that project node.

To build all projects in the solution:

1. Choose Build > Build Solution or press Shift+F7.

or

1. Right-click the solution in the Project Explorer window.
2. Choose Build from the shortcut menu.
To build a single project:

1. Select the required project in the Project Explorer.
2. Choose Build > Build or press F7.

or

1. Right-click the project in the Project Explorer.
2. Choose Build.

To compile a single file:

1. In the Project Explorer, click to select the source file to compile.
2. Choose Build > Compile or press Ctrl+F7.

or

1. In the Project Explorer, right-click the source file to compile.
2. Choose Compile from the shortcut menu.

Correcting errors after building

The results of a build are recorded in a Build Log that is displayed in the Output window. Errors are highlighted in red, warnings are highlighted in yellow. Double-clicking an error, warning, or note will move the insertion point to the line of source code that triggered that log entry.

You can move forward and backward through errors using Search > Next Location and Search > Next Location.

When you build a single project in a single configuration, the Transcript will display the memory used by the application and a summary for each memory area.
Creating variants using configurations

SEGGER Embedded Studio provides a facility to build projects in various configurations. Project configurations are used to create different software builds for your projects.

A configuration defines a set of project options. For example, the output of a compilation can be put into different directories, dependent upon the configuration. When you create a solution, some default project configurations are created.

Build configurations and their uses

Configurations are typically used to differentiate debug builds from release builds. For example, the compiler options for debug builds will differ from those of a release build: a debug build will set options so the project can be debugged easily, whereas a release build will enable optimization to reduce program size or to increase its speed. Configurations have other uses; for example, you can use configurations to produce variants of software, such as custom libraries for several different hardware variants.

Configurations inherit project options from other configurations. This provides a single point of change for definitions common to several configurations. A particular project option can be overridden in a particular configuration to provide configuration-specific settings.

When a solution is created, two configurations are generated **Debug** and **Release** and you can create additional configurations by choosing **Build > Build Configurations**. Before you build, ensure that the appropriate configuration is set using **Build > Set Active Build Configuration** or, alternatively, the **Active Configuration** combo box in the **Project Explorer**.

Selecting a configuration

To set the configuration that affects your building and debugging, use the combo box in the **Project Explorer** or select **Build > Set Active Build Configuration**.

Creating a configuration

To create your own configurations, select **Build > Build Configurations** to invoke the **Configurations** dialog. The **New** button will produce a dialog allowing you to name your configuration. You can now specify the existing configurations from which your new configuration will inherit values.

Deleting a configuration

You can delete a configuration by selecting it and clicking the **Remove** button. This deletion cannot be undone or canceled, so beware.
Private configurations

Some configurations are defined purely for inheriting and, as such, should not appear in the **Build** combo box. When you select a configuration in the **Configuration** dialog, you can choose to hide that configuration.
Project options

For solutions, projects, folders, and files, project options can be defined that are used by the project system in the build process. These project options can be viewed and modified by using the Options dialog in conjunction with the Project Explorer.

Some project options are only applicable to a given item type. For example, linker project options are only applicable to a project that builds an executable file. However, other project options can be applied either at the file, project, or solution project node. For example, a compiler project option can be applied to a solution, project, or individual file. By setting a project option at the solution level, you enable all files of the solution to use that project option's value.

Unique project options

A unique project option has one value. When a build is done, the value of a unique project option is the first one defined in the project hierarchy. For example, the Treat Warnings As Errors project option could be set to Yes at the solution level, which would then be applicable to every file in the solution that is compiled, assembled, and linked. You can then selectively define project options for other project items. For example, a particular source file may have warnings you decide are allowable, so you set the Treat Warnings As Errors to No for that particular file.

<table>
<thead>
<tr>
<th>solution</th>
<th>Treat Warnings As Errors = Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>project1</td>
<td>Treat Warnings As Errors = Yes</td>
</tr>
<tr>
<td>file1</td>
<td>Treat Warnings As Errors = Yes</td>
</tr>
<tr>
<td>file2</td>
<td>Treat Warnings As Errors = No</td>
</tr>
<tr>
<td>project2</td>
<td>Treat Warnings As Errors = No</td>
</tr>
<tr>
<td>file1</td>
<td>Treat Warnings As Errors = No</td>
</tr>
<tr>
<td>file2</td>
<td>Treat Warnings As Errors = Yes</td>
</tr>
</tbody>
</table>

In the above example, the files will be compiled with these values for Treat Warnings As Errors:

<table>
<thead>
<tr>
<th>project1/file1</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>project1/file2</td>
<td>No</td>
</tr>
<tr>
<td>project2/file1</td>
<td>No</td>
</tr>
<tr>
<td>project2/file2</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Aggregate project options

An aggregating project option collects all the values defined for it in the project hierarchy. For example, when a C file is compiled, the Preprocessor Definitions project option will take all the values defined at the file, project, and solution levels.

<p>| solution       | Preprocessor Definitions = SolutionDef |</p>
<table>
<thead>
<tr>
<th>project1/file1</th>
<th>SolutionDef</th>
</tr>
</thead>
<tbody>
<tr>
<td>project1/file2</td>
<td>SolutionDef, File1Def</td>
</tr>
<tr>
<td>project2/file1</td>
<td>SolutionDef, ProjectDef</td>
</tr>
<tr>
<td>project2/file2</td>
<td>SolutionDef, ProjectDef, File2Def</td>
</tr>
</tbody>
</table>
Configurations and project options

Project options are defined for a configuration so you can have different values for a project option for different builds. A given configuration can inherit the project options of other configurations. When the project system requires a project option value, it checks for the existence of the project option value in the current configuration and then in the set of inherited configurations. You can specify the set of inherited configurations using the Configurations dialog.

A special configuration named Common is always inherited by a configuration. The Common configuration allows you to set project options that will apply to all configurations you create. If you are modifying a project option of your project, you almost certainly want each configuration to inherit it, so ensure that the Common configuration is selected.

If the project option is unique, the build system will use the one defined for the particular configuration. If the project option isn't defined for this configuration, the build system uses an arbitrary one from the set of inherited configurations.

If the option is still undefined, the build system uses the value for the Common configuration. If it is still undefined, the build system tries to find the value in the next higher level of the project hierarchy.

In the above example, the files will be compiled with these preprocessor definitions when in Debug configuration.

<table>
<thead>
<tr>
<th>File</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>project1/file1</td>
<td>CommonSolutionDef, DebugSolutionDef</td>
</tr>
<tr>
<td>project1/file2</td>
<td>CommonSolutionDef, DebugSolutionDef, CommonFile1Def, DebugFile1Def</td>
</tr>
<tr>
<td>project2/file1</td>
<td>CommonSolutionDef, DebugSolutionDef, ProjectDef</td>
</tr>
</tbody>
</table>
project2/file2 | ComonSolutionDef, DebugSolutionDef, ProjectDef, File2Def

and the files will be compiled with these **Preprocessor Definitions** when in **Release** configuration:

<table>
<thead>
<tr>
<th>File</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>project1/file1</td>
<td>CommonSolutionDef, ReleaseSolutionDef</td>
</tr>
<tr>
<td>project1/file2</td>
<td>CommonSolutionDef, ReleaseSolutionDef, CommonFile1Def</td>
</tr>
<tr>
<td>project2/file1</td>
<td>CommonSolutionDef, ReleaseSolutionDef, ProjectDef</td>
</tr>
<tr>
<td>project2/file2</td>
<td>CommonSolutionDef, ReleaseSolutionDef, ProjectDef, File2Def</td>
</tr>
</tbody>
</table>
Project macros

You can use macros to modify the way the project system refers to files.

Macros are divided into four classes:

System macros defined by SEGGER Embedded Studio relay information about the environment, such as paths to common directories.

Global macros are saved in the environment and are shared across all solutions and projects. Typically, you would set up paths to libraries and any external items here.

Project macros are saved as project options in the project file and can define values specific to the solution or project in which they are defined.

Build macros are generated by the project system when you build your project.

System macros

System macros are defined by SEGGER Embedded Studio itself and as such are read-only. System macros can be used in project options, environment settings and to refer to files. See System macros list for the list of System macros.

Global macros

Global macros are store in the environment option Build Macros.

To define a global macro:

1. Use Tools > Options to show the environment options dialog.
2. In the Environment Options dialog’s Building group, select the Build Macros option.
3. Click the ellipsis button on the right.
4. Set the macro using the syntax name = replacement text.

Project macros

To define a project macro:

To set the project macros:

1. Select the appropriate solution/project in the Project Explorer.
2. Use Project > Options to show the project options dialog.
3. In the Project Options dialog’s General Options group, select the Macros option.
4. Click the ellipsis button on the right.
5. Set the macro using the syntax name = replacement text.
Build macros

Build macros are defined by the project system for a build of a given project node. See Build macros list for the list of build macros.

Using macros

You can use a macro for a project option or environment setting by using the $(macro) syntax. For example, the Object File Name option has a default value of $(IntDir)/$(InputName)$(OBJ).

You can also specify a default value for a macro if it is undefined using the $(macro:default) syntax. For example, $(MyMacro:0) would expand to 0 if the macro MyMacro has not been defined.
Dependencies and build order

You can set up dependency relationships between projects using the Project Dependencies dialog. Project dependencies make it possible to build solutions in the correct order and, where the target permits, to load and delete applications and libraries in the correct order. A typical usage of project dependencies is to make an executable project dependent upon a library executable. When you elect to build the executable, the build system will ensure that the library it depends upon is up to date. In the case of a dependent library, the output file of the library build is supplied as an input to the executable build, so you don’t have to worry about it.

Project dependencies are stored as project options and, as such, can be defined differently based upon the selected configuration. You almost always want project dependencies to be independent of the configuration, so the Project Dependencies dialog selects the Common configuration by default.

To make one project dependent upon another:

1. Choose Project > Project Dependencies.
2. From the Project dropdown, select the target project that depends upon other projects.
3. In the Depends Upon list box, select the projects the target project depends upon and deselect the projects it does not depend upon.

Some items in the Depends Upon list box may be dimmed, indicating that a circular dependency would result if any of those projects were selected. In this way, SEGGER Embedded Studio prevents you from constructing circular dependencies using the Project Dependencies dialog.

If your target supports loading multiple projects, the Build Order also reflects the order in which projects are loaded onto the target. Projects will load, in order, from top to bottom. Generally, libraries need to be loaded before the applications that use them, and you can ensure this happens by making the application dependent upon the library. With this dependency set, the library gets built and loaded before the application does.

Applications are deleted from a target in reverse of their build order; in this way, applications are removed before the libraries on which they depend.
Linking and section placement

Executable programs consist of a number of sections. Typically, there are program sections for code, initialized data, and zeroed data. There is often more than one code section and they must be placed at specific addresses in memory.

To describe how the program sections of your program are positioned in memory, the SEGGER Embedded Studio for ARM project system uses *memory-map* files and *section-placement* files. These XML-formatted files are described in *Memory Map file format* and *Section Placement file format*. They can be edited with the SEGGER Embedded Studio for ARM text editor. The memory-map file specifies the start address and size of target memory segments. The section-placement file specifies where to place program sections in the target’s memory segments. Separating the memory map from the section-placement scheme enables a single hardware description to be shared across projects and also enables a project to be built for a variety of hardware descriptions.

For example, a memory-map file representing a device with two memory segments called *FLASH* and *SRAM* could look something like this in the memory-map editor.

```xml
<Root name="Device1">
  <MemorySegment name="FLASH" start="0x10000000" size="0x10000" />
  <MemorySegment name="SRAM" start="0x20000000" size="0x1000" />
</Root>
```

A corresponding section-placement file will refer to the memory segments of the memory-map file and will list the sections to be placed in those segments. This is done by using a memory-segment name in the section-placement file that matches the corresponding memory-segment name in the memory-map file.

For example, a section-placement file that places a section called *.stack* in the *SRAM* segment and the *.vectors* and *.text* sections in the *FLASH* segment would look like this:

```xml
<Root name="Flash Section Placement">
  <MemorySegment name="FLASH">
    <ProgramSection name=".vectors" load="Yes" />
    <ProgramSection name=".text" load="Yes" />
  </MemorySegment>
  <MemorySegment name="SRAM">
    <ProgramSection name=".stack" load="No" />
  </MemorySegment>
</Root>
```

Note that the order of section placement within a segment is top down; in this example *.vectors* is placed at lower addresses than *.text*. The order memory segments are processed is bottom up; so in this example the sections in the *SRAM* segment will be placed prior to the sections in the *FLASH* segment.

Multiple memory segments can be specified by separating them with a semicolon. In the following example, the *.stack* section will be placed in the SRAM2 memory segment if it exists in the memory map, otherwise it will be placed in the SRAM memory segment. Sections can only be placed in one segment, they will not be placed in a second segment when the first is full.

```xml
<Root name="Flash Section Placement">
  <MemorySegment name="FLASH">
    <ProgramSection name=".vectors" load="Yes" />
    <ProgramSection name=".text" load="Yes" />
  </MemorySegment>
  <MemorySegment name="SRAM">
    <ProgramSection name=".stack" load="No" />
  </MemorySegment>
</Root>
```
The memory-map file and section-placement file to use for linkage can be included as a part of the project or, alternatively, they can be specified in the project's linker options.

You can create a new program section using either the assembler or the compiler. For the C/C++ compiler, this can be achieved using `__attribute__` on declarations. For example:

```c
void foobar(void) __attribute__((section(".foo")));
```

This will allocate `foobar` in the section called `.foo`. Alternatively, you can specify the names for the code, constant, data, and zeroed-data sections of an entire compilation unit by using the `Section Options` options.

You can now place the section into the section placement file using the editor so that it will be located after the vectors sections as follows:

```xml
<Root name="Flash Section Placement">
  <MemorySegment name="FLASH">
    <ProgramSection name=".vectors" load="Yes"/>
    <ProgramSection name=".text" load="Yes"/>
  </MemorySegment>
  <MemorySegment name="SRAM">
    <ProgramSection name=".stack" load="No"/>
  </MemorySegment>
</Root>
```

If you are modifying a section-placement file that is supplied in the SEGGER Embedded Studio for ARM distribution, you will need to import it into your project using the Project Explorer.

Sections containing code and constant data should have their load project option set to Yes. Some sections don't require any loading, such as stack sections and zeroed-data sections; such sections should have their load project option set to No.

Some sections that are loaded then need to be copied to sections that aren't yet loaded. This is required for initialized data sections and to copy code from slow memory regions to faster ones. To do this, the `runin` attribute should contain the name of a section in the section-placement file to which the section will be copied.

For example, initialized data is loaded into the .data section and then is copied into the .data_run section using:

```xml
<Root name="Flash Section Placement">
  <MemorySegment name="FLASH">
    <ProgramSection name=".vectors" load="Yes"/>
    <ProgramSection name=".text" load="Yes"/>
    <ProgramSection name=".data" load="Yes" runin=".data_run"/>
  </MemorySegment>
</Root>
```
The startup code will copy the contents of the .data section to the .data_run section. To enable this, symbols named __section-name_start__, __section-name_end__, __section-name_load_start__ and __section-name_load_end__ are generated marking the section start, end, load start and load end addresses of each section. The startup code uses these symbols to copy the sections from their load positions to their run positions.

You can also create your own load and run section, for example the following placement file adds a .mydata section:

```xml
<Root name="Flash Section Placement">
    <MemorySegment name="FLASH">
        <ProgramSection name=".vectors" load="Yes" />
        <ProgramSection name=".text" load="Yes" />
        <ProgramSection name=".data" load="Yes" runin=".data_run" />
        <ProgramSection name=".mydata" load="Yes" runin=".mydata_run" />
    </MemorySegment>
    <MemorySegment name="SRAM">
        <ProgramSection name=".data_run" load="No" />
        <ProgramSection name=".mydata_run" load="No" />
        <ProgramSection name=".stack" load="No" />
    </MemorySegment>
</Root>
```

As the startup code doesn't know about this section, the following code will need to be added to the program to initialise the section:

```c
/* Section image located in flash */
extern const unsigned char __mydata_load_start__[];
extern const unsigned char __mydata_load_end__[];

/* Where to locate the section image in RAM. */
extern unsigned char __mydata_start__[];
extern unsigned char __mydata_end__[];
...

/* Copy image from flash to RAM. */
memcpy((__mydata_start__,
       __mydata_load_start__,
       __mydata_end__ - __mydata_start__));
```
Using source control

Source control is an essential tool for individuals or development teams. SEGGER Embedded Studio integrates with several popular source-control systems to provide this feature for files in your SEGGER Embedded Studio for ARM projects.

Source-control capability is implemented by a number of third-party providers, but the set of functions provided by SEGGER Embedded Studio for ARM aims to be provider independent.
Source control capabilities

The source-control integration capability provides:

- Connecting to the source-control repository and mapping files in the SEGGER Embedded Studio for ARM project to those in source control.
- Showing the source-control status of files in the project.
- Adding files in the project to source control.
- Fetching files in the project from source control.
- Optionally locking and unlocking files in the project for editing.
- Comparing a file in the project with the latest version in source control.
- Updating a file in the project by merging changes from the latest version in source control.
- Committing changes made to project files into source control.
Configuring source-control providers

SEGGER Embedded Studio supports Subversion, Git, and Mercurial as source-control systems. To enable SEGGER Embedded Studio to utilize source-control features, you need to install, on your operating system, the appropriate command line client for the source-control systems that you will use.

Once you have installed the command line client, you must configure SEGGER Embedded Studio to use it.

To configure Subversion:

1. Choose Tools > Options or press Alt+,.
2. Select the Source Control category in the options dialog.
3. Set the Executable environment option of the Subversion Options group to point to Subversion svn command. On Windows operating systems, the Subversion command is svn.exe.

To configure Git:

1. Choose Tools > Options or press Alt+,.
2. Select the Source Control category in the options dialog.
3. Set the Executable environment option of the Git Options group to point to Git git command. On Windows operating systems, the Git command is git.exe.

To configure Mercurial:

1. Choose Tools > Options or press Alt+,.
2. Select the Source Control category in the options dialog.
3. Set the Executable environment option of the Mercurial Options group to point to Git hg command. On Windows operating systems, the Git command is hg.exe.
Connecting to the source-control system

When SEGGER Embedded Studio loads a project, it examines the file system folder that contains the project to determine the source-control system the project uses. If SEGGER Embedded Studio cannot determine, from the file system, the source-control system in use, it disables source-control integration.

That is, if you have not set up the paths to the source-control command line clients, even if a working copy exists and the appropriate command line client is installed, SEGGER Embedded Studio cannot establish source-control integration for the project.

User credentials

You can set the credentials that the source-control system uses, for commands that require credentials, using VCS > Options > Configure. From here you can set the user name and password. These details are saved to the session file (the password is encrypted) so you won’t need to specify this information each time the project is loaded.

Note

SEGGER Embedded Studio has no facility to create repositories from scratch, nor to clone, pull, or checkout repositories to a working copy: it is your responsibility to create a working copy outside of SEGGER Embedded Studio using your selected command-line client or Windows Explorer extension.

The “Tortoise” products are a popular set of tools to provide source-control facilities in the Windows shell. Use Google to find TortoiseSVN, TortoiseGit, and TortoiseHG and see if you like them.
File source-control status

Determining the source-control status of a file can be expensive for large repositories, so SEGGER Embedded Studio for ARM updates the source-control status in the background. Priority is given to items that are displayed.

A file will be in one of the following states:

- **Clean**: The file is in source control and matches the tip revision.
- **Not Controlled**: The file is not in source control.
- **Conflicted**: The file is in conflict with changes made to the repository.
- **Locked**: The file is locked.
- **Update Available**: The file is older than the most-recent version in source control.
- **Added**: The file is scheduled to be added to the repository.
- **Removed**: The file is scheduled to be removed from the repository.

If the file has been modified, its status is displayed in red in the Project Explorer. Note that if a file is not under the local root, it will not have a source-control status.

You can reset any stored source-control file status by choosing VCS > Refresh.
Source-control operations

Source-control operations can be performed on single files or recursively on multiple files in the Project Explorer hierarchy. Single-file operations are available on the Source Control toolbar and on the text editor's shortcut menu. All operations are available using the VCS menu. The operations are described in terms of the Project Explorer shortcut menu.
Adding files to source control

To add files to the source-control system:

1. In the Project Explorer, select the file to add. If you select a folder, project, or solution, any eligible child items will also be added to source control.
2. choose Source Control > Add or press Ctrl+R, A.
3. The dialog will list the files that can be added.
4. In that dialog, you can deselect any files you don’t want to add to source control.
5. Click Add.

Note

Files are scheduled to be added to source control and will only be committed to source control (and seen by others) when you commit the file.

Enabling the VCS > Options > Add Immediately option will bypass the dialog and immediately add (but not commit) the files.
Updating files

To update files from source control:

1. In the **Project Explorer**, select the file to update. If you select a folder, project, or solution, any eligible child items will also be updated from source control.
2. choose **Source Control > Update** or press **Ctrl+R, U**.
3. The dialog will list the files that can be updated.
4. In that dialog, you can deselect any files you don’t want to update from source control.
5. Click **Update**.

**Note**

Enabling the **VCS > Options > Update Immediately** option will bypass the dialog and immediately update the files.
Committing files

To commit files:

1. In the Project Explorer, select the file to commit. If you select a folder, project, or solution, any eligible child items will also be committed.
2. Choose Source Control > Commit or press Ctrl+R, C.
3. The dialog will list the files that can be committed.
4. In that dialog, you can deselect any files you don’t want to commit and enter an optional comment.
5. Click Commit.

Note

Enabling the VCS > Options > Commit Immediately option will bypass the dialog and immediately commit the files without a comment.
Reverting files

To revert files:

1. In the Project Explorer, select the file to revert. If you select a folder, project, or solution, any eligible child items will also be reverted.
2. Choose Source Control > Revert or press Ctrl+R, V.
3. The dialog will list the files that can be reverted.
4. In that dialog, you can deselect any files you don’t want to revert.
5. Click Revert.

Note

Enabling the VCS > Options > Revert Immediately option will bypass the dialog and immediately revert files.
Locking files

To lock files:

1. In the Project Explorer, select the file to lock. If you select a folder, project, or solution, any eligible child items will also be locked.
2. Choose Source Control > Lock or press Ctrl+R, L.
3. The dialog will list the files that can be locked.
4. In that dialog, you can deselect any files you don’t want to lock and enter an optional comment.
5. Click Lock.

Note

Enabling the VCS > Options > Lock Immediately option will bypass the dialog and immediately lock files without a comment.
Unlocking files

To unlock files:

1. In the Project Explorer, select the file to lock. If you select a folder, project, or solution, any eligible child items will also be unlocked.
2. Choose Source Control > Unlock or press Ctrl+R, N.
3. The dialog will list the files that can be unlocked.
4. In that dialog, you can deselect any files you don't want to unlock.
5. Click Unlock.

Note

Enabling the VCS > Options > Unlock Immediately option will bypass the dialog and immediately unlock files.
Removing files from source control

To remove files from source control:

1. In the Project Explorer, select the file to remove. If you select a folder, project, or solution, any eligible child items will also be removed.
2. Choose Source Control > Remove or press Ctrl+R, R.
3. The dialog will list the files that can be removed.
4. In that dialog, you can deselect any files you don’t want to remove.
5. Click Remove.

Note

Files are scheduled to be removed from source control and will still be and seen by others, giving you the opportunity to revert the removal. When you commit the file, the file is removed from source control.

Enabling the VCS > Options > Remove Immediately option will bypass the dialog and immediately remove (but not commit) files.
Showing differences between files

To show the differences between the file in the project and the version checked into source control, do the following:

1. In the **Project Explorer**, right-click the file.
2. From the shortcut menu, choose **Source Control > Show Differences**.

You can use an external diff tool in preference to the built-in SEGGER Embedded Studio for ARM diff tool. To define the diff command line SEGGER Embedded Studio for ARM generates, choose **Tools > Options > Source Control > Diff Command Line**. The command line is defined as a list of strings to avoid problems with spaces in arguments. The diff command line can contain the following macros:

- **$(localfile)**: The filename of the file in the project.
- **$(remotefile)**: The filename of the latest version of the file in source control.
- **$(localname)**: A display name for **$(localfile)**.
- **$(remotename)**: A display name for **$(remotefile)**.
Source-control properties

When a file in the project is in source control, the **Properties** window shows the following properties in the **Source Control Options** group:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEGGER Embedded Studio Status</td>
<td>The source-control status of working copy as viewed by SEGGER Embedded Studio.</td>
</tr>
<tr>
<td>last Author</td>
<td>The author of the file's head revision.</td>
</tr>
<tr>
<td>Path: Relative</td>
<td>The item's path relative to the repository root.</td>
</tr>
<tr>
<td>Path: Repository</td>
<td>The pathname of the file in the source-control system, typically a URL.</td>
</tr>
<tr>
<td>Path: Working Copy</td>
<td>The pathname of the file in the working copy.</td>
</tr>
<tr>
<td>Provider</td>
<td>The name of the source-control system managing this file.</td>
</tr>
<tr>
<td>Provider Status</td>
<td>The status of the file as reported by the source-control provider.</td>
</tr>
<tr>
<td>Revision: Local</td>
<td>The revision number/name of the local file.</td>
</tr>
<tr>
<td>Revision: Remote</td>
<td>The revision number/name of the most-recent version in source control.</td>
</tr>
<tr>
<td>Status: In Conflict?</td>
<td>If <strong>Yes</strong>, updates merged into the file using <strong>Update</strong> conflict with the changes you made locally; if <strong>No</strong>, the file is not locked. When conflicted, must resolve the conflicts and mark them <strong>Resolved</strong> before committing the file.</td>
</tr>
<tr>
<td>Status: Locked?</td>
<td>If <strong>Yes</strong>, the file is lock by you; if <strong>No</strong>, the file is not locked.</td>
</tr>
<tr>
<td>Status: Modified?</td>
<td>If <strong>Yes</strong>, the checked-out file differs from the version in the source control system; if <strong>No</strong>, they are identical.</td>
</tr>
<tr>
<td>Status: Update Available?</td>
<td>If <strong>Yes</strong>, the file in the project location is an old version compared to the latest version in the source-control system use <strong>Update</strong> to merge in the latest changes.</td>
</tr>
</tbody>
</table>
Subversion provider

The Subversion source-control provider has been tested with SVN 1.4.3.

Provider-specific options

The following environment options are supported:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executable</td>
<td>The path to the <code>svn</code> executable.</td>
</tr>
<tr>
<td>Lock Supported</td>
<td>If Yes, check out and undo check out operations are supported. Check out will issue the <code>svn lock</code> command; check in and undo check out will issue the <code>svn unlock</code> command.</td>
</tr>
<tr>
<td>Authentication</td>
<td>Selects whether authentication (user name and password) is sent with every command.</td>
</tr>
<tr>
<td>Show Updates</td>
<td>Selects whether the update (<code>-u</code> flag) is sent with status requests in order to show that new versions are available in the repository. Note that this requires a live connection to the repository: if you are working without a network connection to your repository, you can disable this switch and continue to enjoy source control status information in the Project Explorer and Pending Changes windows.</td>
</tr>
</tbody>
</table>

Connecting to the source-control system

When connecting to source control, the provider checks if the local root is in SVN control. If this is the case, the local and remote root will be set accordingly. If the local root is not in SVN control after you have set the remote root, a `svn checkout -N` command will be issued to make the local root SVN controlled. This command will also copy any files in the remote root to the local root.

The user name and password you enter will be supplied with each `svn` command the provider issues.

Source control operations

The SEGGER Embedded Studio for ARM source-control operations are implemented using Subversion commands. Mapping SEGGER Embedded Studio for ARM source-control operations to Subversion source-control operations is straightforward:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit</td>
<td><code>svn commit</code> for the file, with optional comment.</td>
</tr>
<tr>
<td>Update</td>
<td><code>svn update</code> for each file.</td>
</tr>
<tr>
<td>Action</td>
<td>Command</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Revert</td>
<td><code>svn revert</code> for each file.</td>
</tr>
<tr>
<td>Resolved</td>
<td><code>svn resolved</code> for each file.</td>
</tr>
<tr>
<td>Lock</td>
<td><code>svn lock</code> for each file, with optional comment.</td>
</tr>
<tr>
<td>Unlock</td>
<td><code>svn unlock</code> for each file.</td>
</tr>
<tr>
<td>Add</td>
<td><code>svn add</code> for each file.</td>
</tr>
<tr>
<td>Remove</td>
<td><code>svn remove</code> for each file.</td>
</tr>
<tr>
<td>Source Control Explorer</td>
<td><code>svn list</code> with a remote root. <code>svn mkdir</code> to create directories in the repository.</td>
</tr>
</tbody>
</table>
CVS provider

The CVS source-control provider has been tested with CVSNT 2.5.03. The CVS source-control provider uses the CVS rls command to browse the repository; this command is implemented in CVS 1.12 but usage of . as the root of the module name is not supported.

Provider-specific options

The following environment options are supported:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVSROOT</td>
<td>The CVSROOT value to access the repository.</td>
</tr>
<tr>
<td>Edit/Unedit Supported</td>
<td>If Yes, Check Out and Undo Check Out commands are supported. Any check-out operation will issue the cvs edit command; any check-in or undo-check-out operation will issue the cvs unedit command; the status operation will issue the cvs ss command.</td>
</tr>
<tr>
<td>Executable</td>
<td>The path to the cvs executable.</td>
</tr>
<tr>
<td>Login/Logout Required</td>
<td>If Yes, Connect will issue the cvs login command.</td>
</tr>
</tbody>
</table>

Connecting to the source-control system

When connecting to source control, the provider checks if the local root is in CVS control. If this is the case, the local and remote root will be set accordingly. If the local root is not in CVS control after you have set the remote root, a cvs checkout -l -d command will be issued to make the local root CVS controlled. This command will also copy any files in the remote root to the local root.

Source-control operations

The SEGGER Embedded Studio for ARM source-control operations have been implemented using CVS commands. There are no multiple-file operations, each operation is done on a single file and committed as part of the operation.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Status</td>
<td>cvs status and optional cvs editors for local directories in CVS control. cvs rls -e for directories in the repository.</td>
</tr>
<tr>
<td>Add To Source Control</td>
<td>cvs add for each directory not in CVS control.</td>
</tr>
<tr>
<td></td>
<td>cvs add for the file. cvs commit for the file and directories.</td>
</tr>
<tr>
<td>Get Latest</td>
<td><code>cvs update -l -d</code> for each directory not in CVS control. <code>cvs update</code> to merge the local file. <code>cvs update -C</code> to overwrite the local file.</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Check Out</td>
<td>Optional <code>cvs update -C</code> to get the latest version. <code>cvs edit</code> to lock the file.</td>
</tr>
<tr>
<td>Undo Check Out</td>
<td><code>cvs unedit</code> to unlock the file. Optional <code>cvs update</code> to get the latest version.</td>
</tr>
<tr>
<td>Check In</td>
<td><code>cvs commit</code> for the file.</td>
</tr>
<tr>
<td>Source Control Explorer</td>
<td><code>cvs rls -e</code> with a remote root starting with <code>..</code> <code>cvs import</code> to create directories in the repository.</td>
</tr>
</tbody>
</table>
Package management

Additional target-support functions can be added to, and removed from, SEGGER Embedded Studio for ARM with packages.

A SEGGER Embedded Studio for ARM package is an archive file containing a collection of target-support files. Installing a package involves copying the files it contains to an appropriate destination directory and registering the package with SEGGER Embedded Studio for ARM's package system. Keeping target-support files separate from the main SEGGER Embedded Studio for ARM installation allows us to support new hardware and issue bug fixes for existing hardware-support files between SEGGER Embedded Studio for ARM releases, and it allows third parties to develop their own support packages.

Installing packages

Use the Package Manager to automate the download, installation, upgrade and removal of packages.

To activate the Package Manager:

Choose Tools > Manage Packages.

In some situations, such as using SEGGER Embedded Studio for ARM on a computer without Internet access or when you want to install packages that are not on the website, you cannot use the Package Manager to install packages and it will be necessary to manually install them.

To manually install a package:

1. Choose Tools > Manually Install Packages.
2. Select one or more package files you want to install.
3. Click Open to install the packages.

Choose Tools > Show Installed Packages to see more information on the installed packages.

The Package Manager window will remove manually installed packages.

The package manager

The Package Manager manages the support packages installed on your system. It lists the available packages, shows the installed packages, and allows you to install, update, reinstall, and remove them.
To activate the Package Manager:

Choose Tools > Manage Packages.

Filtering the package list

By default, the Package Manager lists all available and installed packages. You can filter the displayed packages in a number of ways.

To filter by package status:

Click on the disclosure icon near the top-right corner of the dialog.

Use the pop-up menu to choose how to filter the list of packages.

The list-filter choices are:

Display All Show all packages irrespective of their status.
Display Not Installed Show packages that are available but are not currently installed.
**Display Installed** Only show packages that are installed.

**Display Updates** Only show packages that are installed but are not up-to-date because a newer version is available.

You can also filter the list of packages by the text in the package’s title and documentation.

**To filter packages by keyword:**

Type the keyword into the **Search Packages** box at the top-left corner of the dialog.

**Installing a package**

The package-installation operation downloads a package to `$(PackagesDir)/downloads`, if it has not been downloaded already, and unpacks the files contained within the package to their destination directory.

**To install a package:**

1. Choose **Tools > Package Manager** and set the status filter to **Display Not Installed**.
2. Select the package or packages you wish to install.
3. Right-click the selected packages and choose **Install Selected Packages** from the shortcut menu.
4. Click **Next**; you will see the actions the **Package Manager** is about to carry out.
5. Click **Next** and the **Package Manager** will install the selected packages.
6. When installation is complete, click **Finish** to close the **Package Manager**.

**Updating a package**

The package-update operation first removes existing package files, then it downloads the updated package to `$(PackagesDir)/downloads` and unpacks the files contained within the package to their destination directory.

**To update a package:**

1. Choose **Tools > Package Manager** and set the status filter to **Display Updates**.
2. Select the package or packages you wish to update.
3. Right-click the selected packages and choose **Update Selected Packages** from the shortcut menu.
4. Click **Next**; you will see the actions the **Package Manager** is about to carry out.
5. Click **Next** and the **Package Manager** will update the package(s).
6. When the update is complete, click **Finish** to close the **Package Manager**.

**Removing a package**

The package-remove operation removes all the files that were extracted when the package was installed.
To remove a package:

1. Choose Tools > Package Manager and set the status filter to Display Installed.
2. Select the package or packages you wish to remove.
3. Right-click the selected packages and choose Remove Selected Packages from the shortcut menu.
4. Click Next; you will see the actions the Package Manager is about to carry out.
5. Click Next and the Package Manager will remove the package(s).
6. When the operation is complete, click Finish to close the Package Manager.

Reinstalling a package

The package-reinstall operation carries out a package-remove operation followed by a package-install operation.

To reinstall a package:

1. Choose Tools > Package Manager and set the status filter to Display Installed.
2. Select the package or packages you wish to reinstall.
3. Right-click the packages to reinstall and choose Reinstall Selected Packages from the shortcut menu.
4. Click Next; you will see the actions the Package Manager is about to carry out.
5. Click Next and the Package Manager will reinstall the packages.
6. When the operation is complete, click Finish to close the Package Manager.
Exploring your application

In this section, we discuss the SEGGER Embedded Studio tools that help you examine how your application is built.
Project explorer

The Project Explorer is the user interface of the SEGGER Embedded Studio for ARM project system. It organizes your projects and files and provides access to the commands that operate on them. A toolbar at the top of the window offers quick access to commonly used commands for the selected project node or the active project. Right-click to reveal a shortcut menu with a larger set of commands that will work on the selected project node, ignoring the active project.

The selected project node determines what operations you can perform. For example, the Compile operation will compile a single file if a file project node is selected; if a folder project node is selected, each of the files in the folder are compiled.

You can select project nodes by clicking them in the Project Explorer. Additionally, as you switch between files in the editor, the selection in the Project Explorer changes to highlight the file you're editing.

To activate the Project Explorer:

Choose View > Project Explorer or press Ctrl+Alt+P.

Left-click operations

The following operations are available in the Project Explorer with a left-click of the mouse:

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single click</td>
<td>Select the node. If the node is already selected and is a solution, project, or folder node, a rename editor appears.</td>
</tr>
<tr>
<td>Double click</td>
<td>Double-clicking a solution node or folder node will reveal or hide the node's children. Double-clicking a project node selects it as the active project. Double-clicking a file opens the file with the default editor for that file's type.</td>
</tr>
</tbody>
</table>

Toolbar commands

The following buttons are on the toolbar:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Add a new file to the active project using the New File dialog.</td>
</tr>
<tr>
<td></td>
<td>Add existing files to the active project.</td>
</tr>
</tbody>
</table>
Remove files, folders, projects, and links from the project.

Create a new folder in the active project.

Menu of build operations.

Disassemble the active project.

Menu of Project Explorer options.

Display the properties dialog for the selected item.

### Shortcut menu commands

The shortcut menu, displayed by right-clicking, contains the commands listed below.

**For solutions:**

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build and Batch Build</td>
<td>Build all projects under the solution in the current or batch build configuration.</td>
</tr>
<tr>
<td>Rebuild and Batch Rebuild</td>
<td>Rebuild all projects under the solution in the current or batch build configuration.</td>
</tr>
<tr>
<td>Clean and Batch Clean</td>
<td>Remove all output and intermediate build files for the projects under the solution in the current or batch build configuration.</td>
</tr>
<tr>
<td>Export Build and Batch Export Build</td>
<td>Create an editor with the build commands for the projects under the solution in the current or batch build configuration.</td>
</tr>
<tr>
<td>Add New Project</td>
<td>Add a new project to the solution.</td>
</tr>
<tr>
<td>Add Existing Project</td>
<td>Create a link from an existing solution to this solution.</td>
</tr>
<tr>
<td>Paste</td>
<td>Paste a copied project into the solution.</td>
</tr>
<tr>
<td>Remove</td>
<td>Remove the link to another solution from the solution.</td>
</tr>
<tr>
<td>Rename</td>
<td>Rename the solution node.</td>
</tr>
<tr>
<td>Source Control Operations</td>
<td>Source-control operations on the project file and recursive operations on all files in the solution.</td>
</tr>
<tr>
<td>Edit Solution As Text</td>
<td>Create an editor containing the project file.</td>
</tr>
<tr>
<td>Save Solution As</td>
<td>Change the filename of the project filenote that the saved project file is not reloaded.</td>
</tr>
<tr>
<td>Properties</td>
<td>Show the Properties dialog with the solution node selected.</td>
</tr>
</tbody>
</table>
## For projects:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build and Batch Build</td>
<td>Build the project in the current or batch build configuration.</td>
</tr>
<tr>
<td>Rebuild and Batch Rebuild</td>
<td>Rebuild the project in the current or batch build configuration.</td>
</tr>
<tr>
<td>Clean and Batch Clean</td>
<td>Remove all output and intermediate build files for the project in the current or batch build configuration.</td>
</tr>
<tr>
<td>Export Build and Batch Export Build</td>
<td>Create an editor with the build commands for the project in the current or batch build configuration.</td>
</tr>
<tr>
<td>Link</td>
<td>Perform the project node build operation: link for an Executable project type, archive for a Library project type, and the combine command for a Combining project type.</td>
</tr>
<tr>
<td>Set As Active Project</td>
<td>Set the project to be the active project.</td>
</tr>
<tr>
<td>Debugging Commands</td>
<td>For Executable and Externally Built Executable project types, the following debugging operations are available on the project node: <strong>Start Debugging</strong>, <strong>Step Into Debugging</strong>, <strong>Reset And Debug</strong>, <strong>Start Without Debugging</strong>, <strong>Attach Debugger</strong>, and <strong>Verify</strong>.</td>
</tr>
<tr>
<td>Memory-Map Commands</td>
<td>For Executable project types that don't have memory-map files in the project and have the memory-map file project option set, there are commands to view the memory-map file and to import it into the project.</td>
</tr>
<tr>
<td>Section-Placement Commands</td>
<td>For Executable project types that don't have section-placement files in the project but have the section-placement file project option set, there are commands to view the section-placement file and to import it into the project.</td>
</tr>
<tr>
<td>Target Processor</td>
<td>For Executable and Externally Built Executable project types that have a Target Processor option group, the selected target can be changed.</td>
</tr>
<tr>
<td>Add New File</td>
<td>Add a new file to the project.</td>
</tr>
<tr>
<td>Add Existing File</td>
<td>Add an existing file to the project.</td>
</tr>
<tr>
<td>New Folder</td>
<td>Create a new folder in the project.</td>
</tr>
<tr>
<td>Cut</td>
<td>Cut the project from the solution.</td>
</tr>
<tr>
<td>Copy</td>
<td>Copy the project from the solution.</td>
</tr>
<tr>
<td>Paste</td>
<td>Paste a copied folder or file into the project.</td>
</tr>
<tr>
<td>Remove</td>
<td>Remove the project from the solution.</td>
</tr>
<tr>
<td>Rename</td>
<td>Rename the project.</td>
</tr>
</tbody>
</table>
### Source Control Operations
Source-control, recursive operations on all files in the project.

### Find in Project Files
Run **Find in Files** in the project directory.

### Properties
Show the **Project Manager** dialog and select the project node.

<table>
<thead>
<tr>
<th>For folders:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Item</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>Add New File</td>
<td>Add a new file to the folder.</td>
</tr>
<tr>
<td>Add Existing File</td>
<td>Add an existing file to the folder.</td>
</tr>
<tr>
<td>New Folder</td>
<td>Create a new folder in the folder.</td>
</tr>
<tr>
<td>Cut</td>
<td>Cut the folder from the project or folder.</td>
</tr>
<tr>
<td>Copy</td>
<td>Copy the folder from the project or folder.</td>
</tr>
<tr>
<td>Paste</td>
<td>Paste a copied folder or file into the folder.</td>
</tr>
<tr>
<td>Remove</td>
<td>Remove the folder from the project or folder.</td>
</tr>
<tr>
<td>Rename</td>
<td>Rename the folder.</td>
</tr>
<tr>
<td>Source Control Operations</td>
<td>Source-control recursive operations on all files in the folder.</td>
</tr>
<tr>
<td>Compile</td>
<td>Compile each file in the folder.</td>
</tr>
<tr>
<td>Properties</td>
<td>Show the properties dialog with the folder node selected.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>For files:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Item</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>Open</td>
<td>Edit the file with the default editor for the file's type.</td>
</tr>
<tr>
<td>Open With</td>
<td>Edit the file with a selected editor. You can choose from the <em>Binary Editor</em>, <em>Text Editor</em>, and <em>Web Browser</em>.</td>
</tr>
<tr>
<td>Select in File Explorer</td>
<td>Create a operating system file system window with the file selected.</td>
</tr>
<tr>
<td>Compile</td>
<td>Compile the file.</td>
</tr>
<tr>
<td>Export Build</td>
<td>Create an editor window containing the commands to compile the file in the active build configuration.</td>
</tr>
<tr>
<td>Exclude From Build</td>
<td>Set the <strong>Exclude From Build</strong> option to <strong>Yes</strong> for this project node in the active build configuration.</td>
</tr>
<tr>
<td>Disassemble</td>
<td>Disassemble the output file of the compile into an editor window.</td>
</tr>
<tr>
<td>Preprocess</td>
<td>Run the C preprocessor on the file and show the output in an editor window.</td>
</tr>
<tr>
<td>Cut</td>
<td>Cut the file from the project or folder.</td>
</tr>
<tr>
<td>Action</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>Copy</td>
<td>Copy the file from the project or folder.</td>
</tr>
<tr>
<td>Remove</td>
<td>Remove the file from the project or folder.</td>
</tr>
<tr>
<td>Import</td>
<td>Import the file into the project.</td>
</tr>
<tr>
<td>Source Control Operations</td>
<td>Source-control operations on the file.</td>
</tr>
<tr>
<td>Properties</td>
<td>Show the properties dialog with the file node selected.</td>
</tr>
</tbody>
</table>
Source navigator window

One of the best ways to find your way around your source code is using the Source Navigator. It parses the active project's source code and organizes classes, functions, and variables in various ways.

To activate the Source Navigator:

Choose View > Source Navigator or press Ctrl+Alt+N.

The main part of the Source Navigator window provides an overview of your application's functions, classes, and variables.

SEGGER Embedded Studio displays these icons to the left of each object:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>A C or C++ structure or a C++ namespace.</td>
</tr>
<tr>
<td>🚀</td>
<td>A C++ class.</td>
</tr>
<tr>
<td>🚀</td>
<td>A C++ member function declared private or a function declared with static linkage.</td>
</tr>
<tr>
<td>🚀</td>
<td>A C++ member function declared protected.</td>
</tr>
<tr>
<td>🚀</td>
<td>A C++ member function declared public or a function declared with extern linkage.</td>
</tr>
<tr>
<td>🚀</td>
<td>A C++ member variable declared private or a variable declared with static linkage.</td>
</tr>
<tr>
<td>🚀</td>
<td>A C++ member variable declared protected.</td>
</tr>
<tr>
<td>🚀</td>
<td>A C++ member variable declared public or a variable declared with extern linkage.</td>
</tr>
</tbody>
</table>

Re-parsing after editing

The Source Navigator does not update automatically, only when you ask it to. To parse source files manually, click the Refresh button on the Source Navigator toolbar.

SEGGER Embedded Studio re-parses all files in the active project, and any dependent project, and updates the Source Navigator with the changes. Parsing progress is shown as a progress bar in the in the Source Navigator window. Errors and warnings detected during parsing are sent to the Source Navigator Log in the Output window. You can show the log quickly by clicking the Show Source Navigator Log tool button on the Source Navigator toolbar.
Setting indexing threads

You can configure how many threads SEGGER Embedded Studio launches to index your project.

To set the number of threads launched when indexing a project:

Choose View > Source Navigator or press Ctrl+Alt+N.
Click the Options dropdown button at the right of the toolbar.
Move the slider to select the number of threads to launch.

Increasing the number of threads will complete indexing faster, but may reduce the responsiveness of SEGGER Embedded Studio when editing, for example. You should choose a setting that you are comfortable with for your PC. By default, SEGGER Embedded Studio launches 16 threads to index the project and is a good compromise for a desktop quad-core PC.

Sorting and grouping

You can group objects by their type; that is, whether they are classes, functions, namespaces, structures, or variables. Each object is placed into a folder according to its type.

To group objects by type:

1. On the Source Navigator toolbar, click the arrow to the right of the Cycle Grouping button.
2. Choose Group By Type
References window

The References window shows the results of the last Find References operation. The Find References facility is closely related to the Source Navigator in that it indexes your project and searches for references within the active source code regions.

To activate the References window:

If you have hidden the References window and want to see it again:

Choose View > References or press Ctrl+Alt+R.

To find all references in a project:

1. Open a source file that is part of the active project, or one of its dependent projects.
2. In the editor, move the insertion point within the name of the function, variable, method, or macro to find.
3. Choose Navigate > Find References or press Alt+R.
4. SEGGER Embedded Studio shows the References window, without moving focus, and searches your project in the background.

You can also find references directly from the text editor’s context menu: right-click the item to find and choose Find References. As a convenience, SEGGER Embedded Studio is configured to also run Find References when you Alt+Right-click in the text editorsee Mouse-click accelerators.

To search within the results:

Type the text to search for in the Reference window's search box. As you type, the search results are narrowed.
Click the close button to clear the search text and show all references.

To replace within the results:

Type the replacement text in the Reference window's replace box.
Use the buttons to navigate and replace the text.
The documents that have had replaced text will appear unsaved in the text editor.
Symbol browser window

The Symbol Browser shows useful information about your linked application and complements the information displayed in the Project Explorer window. You can select different ways to filter and group the information in the Symbol Browser to provide an at-a-glance overview of your application. You can use the Symbol Browser to drill down to see the size and location of each part of your program. The way symbols are sorted and grouped is saved between runs; so, when you rebuild an application, SEGGER Embedded Studio automatically updates the Symbol Browser so you can see the effect of your changes on the memory layout of your program.

User interface

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Group symbols by source filename." /></td>
<td>Group symbols by source filename.</td>
</tr>
<tr>
<td><img src="image" alt="Group symbols by symbol type (equates, functions, labels, sections, and variables)." /></td>
<td>Group symbols by symbol type (equates, functions, labels, sections, and variables).</td>
</tr>
<tr>
<td><img src="image" alt="Group symbols by the section where they are defined." /></td>
<td>Group symbols by the section where they are defined.</td>
</tr>
<tr>
<td><img src="image" alt="Move the insertion point to the statement that defined the symbol." /></td>
<td>Move the insertion point to the statement that defined the symbol.</td>
</tr>
<tr>
<td><img src="image" alt="Select columns to display." /></td>
<td>Select columns to display.</td>
</tr>
</tbody>
</table>

The main part of the Symbol Browser displays each symbol (both external and static) that is linked into an application. SEGGER Embedded Studio displays the following icons to the left of each symbol:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Private Equate" /></td>
<td>Private Equate A private symbol not defined relative to a section.</td>
</tr>
<tr>
<td><img src="image" alt="Public Equate" /></td>
<td>Public Equate A public symbol that is not defined relative to a section.</td>
</tr>
<tr>
<td><img src="image" alt="Private Function" /></td>
<td>Private Function A private function symbol.</td>
</tr>
<tr>
<td><img src="image" alt="Public Function" /></td>
<td>Public Function A public function symbol.</td>
</tr>
<tr>
<td><img src="image" alt="Private Label" /></td>
<td>Private Label A private data symbol, defined relative to a section.</td>
</tr>
<tr>
<td><img src="image" alt="Public Label" /></td>
<td>Public Label A public data symbol, defined relative to a section.</td>
</tr>
<tr>
<td><img src="image" alt="Section" /></td>
<td>Section A program section.</td>
</tr>
</tbody>
</table>
Choosing what to show

To activate the Symbol Browser window:

Choose View > Symbol Browser or press Ctrl+Alt+Y.

You can choose to display the following fields for each symbol:

- **Value**: The value of the symbol. For labels, code, and data symbols, this will be the address of the symbol. For absolute or symbolic equates, this will be the value of the symbol.
- **Range**: The range of addresses the code or data item covers. For code symbols that correspond to high-level functions, the range is the range of addresses used for that function’s code. For data addresses that correspond to high-level static or extern variables, the range is the range of addresses used to store that data item. These ranges are only available if the corresponding source file was compiled with debugging information turned on: if no debugging information is available, the range will simply be the first address of the function or data item.
- **Size**: The size, in bytes, of the code or data item. The Size column is derived from the Range of the symbol: if the symbol corresponds to a high-level code or data item and has a range, Size is calculated as the difference between the start and end addresses of the range. If a symbol has no range, the size column is blank.
- **Section**: The section in which the symbol is defined. If the symbol is not defined within a section, the Section column is blank.
- **Type**: The high-level type for the data or code item. If the source file that defines the symbol is compiled with debugging information turned off, type information is not available and the Type column is blank.
- **Frame Size**: The amount of stack space used by a call to the function symbol. If the source file that defines the symbol is compiled with debugging information turned off, frame size information is not available and the Type column is blank.

Initially the Range and Size columns are shown in the Symbol Browser. To select which columns to display, use the Field Chooser button on the Symbol Browser toolbar.

To select the fields to display:

1. Click the Field Chooser button on the Symbol Browser toolbar.
2. Select the fields you wish to display and deselect the fields you wish to hide.

Organizing and sorting symbols

When you group symbols by section, each symbol is grouped underneath the section in which it is defined. Symbols that are absolute or are not defined within a section are grouped beneath (No Section).

To group symbols by section:

1. On the Symbol Browser toolbar, click the arrow next to the Cycle Grouping button.
2. From the pop-up menu, choose Group By Section.

The Cycle Grouping icon will change to indicate that the Symbol Browser is grouping symbols by section.

When you group symbols by type, each symbol is classified as one of the following:

- An Equate has an absolute value and is not defined as relative to, or inside, a section.
- A Function is defined by a high-level code sequence.
- A Variable is defined by a high-level data declaration.
- A Label is defined by an assembly language module. Label is also used when high-level modules are compiled with debugging information turned off.

When you group symbols by source file, each symbol is grouped underneath the source file in which it is defined. Symbols that are absolute, are not defined within a source file, or are compiled without debugging information, are grouped beneath (Unknown).

To group symbols by type:

1. On the Symbol Browser toolbar, click the arrow next to the Cycle Grouping button.
2. Choose Group By Type from the pop-up menu.

The Cycle Grouping icon will change to indicate that the Symbol Browser is grouping symbols by type.

To group symbols by source file:

1. On the Symbol Browser toolbar, click the arrow next to the Cycle Grouping button.
2. Choose Group By Source File.

The Cycle Grouping icon will change to indicate that the Symbol Browser is grouping symbols by source file.

When you sort symbols alphabetically, all symbols are displayed in a single list in alphabetical order.

To list symbols alphabetically:

1. On the Symbol Browser toolbar, click the arrow next to the Cycle Grouping button.
2. Choose Sort Alphabetically.

The Cycle Grouping icon will change to indicate that the Symbol Browser is grouping symbols alphabetically.

Filtering and finding symbols

When you’re dealing with big projects with hundreds, or even thousands, of symbols, a way to filter those symbols in order to isolate just the ones you need is very useful. The Symbol Browser’s toolbar provides an editable combo box you can use to specify the symbols you’d like displayed. You can type * to match a sequence of zero or more characters and ? to match exactly one character.
The symbols are filtered and redisplayed as you type into the combo box. Typing the first few characters of a symbol name is usually enough to narrow the display to the symbol you need. Note: the C compiler prefixes all high-level language symbols with an underscore character, so the variable `extern int u` or the function `void fn(void)` have low-level symbol names `_u` and `_fn`. The **Symbol Browser** uses the low-level symbol name when displaying and filtering, so you must type the leading underscore to match high-level symbols.

**To display symbols that start with a common prefix:**

Type the desired prefix text into the combo box, optionally followed by a "*".

For instance, to display all symbols that start with "i2c_", type "i2c_" and all matching symbols are displayed—you don’t need to add a trailing "*" in this case, because it is implied.

**To display symbols that end with a common suffix:**

Type * into the combo box, followed by the required suffix.

For instance, to display all symbols that end in _data, type *_data and all matching symbols are displayed—in this case, the leading * is required.

When you have found the symbol you’re interested in and your source files have been compiled with debugging information turned on, you can jump to a symbol’s definition using the **Go To Definition** button.

**To jump to the definition of a symbol:**

1. Select the symbol from the list of symbols.
2. On the **Symbol Browser** toolbar, click **Go To Definition**.

or

1. Right-click the symbol in the list of symbols.
2. Choose **Go To Definition** from the shortcut menu.

**Watching symbols**

If a symbol’s range and type is known, you can add it to the most recently opened **Watch** window or **Memory** window.

**To add a symbol to the Watch window:**

1. In the **Symbol Browser**, right-click the symbol you wish to add to the **Watch** window.
2. On the shortcut menu, choose **Add To Watch**.

**To add a symbol to the Memory window:**

1. In the **Symbol Browser**, right-click the symbol you wish to add to the **Memory** window.
2. Choose Locate Memory from the shortcut menu.

Using size information

Here are a few common ways to use the Symbol Browser:

What function uses the most code space? What requires the most data space?

1. Choose View > Symbol Browser or press Ctrl+Alt+Y.
2. In the Grouping button menu on the Symbol Browser toolbar, select Group By Type.
3. Ensure the Size field is checked in the Field Chooser button's menu.
4. Ensure that the filter on the Symbol Browser toolbar is empty.
5. Click on the Size field in the header to sort by data size.
6. The sizes of variables and of functions are shown in separate lists.

What's the overall size of my application?

1. Choose View > Symbol Browser or press Ctrl+Alt+Y.
2. In the Grouping button menu on the Symbol Browser toolbar, select Group By Section.
3. Ensure the Range and Size fields are checked in the Field Chooser button's menu.
4. Read the section sizes and ranges of each section in the application.
Stack usage window

The Stack Usage Window finds the call paths of your linked application and displays them as a call tree together with their minimal stack requirements. A call path of your application is any function that has been linked in but has no direct call made to it but will make calls to other functions. The main function is the most obvious example of a call path, an interrupt handler or a function that is called only as a function pointer are other examples. To use the stack usage window your linked application must be compiled with debugging information enabled.

User interface

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Arrow Up]</td>
<td>Move the insertion point to the statement that defined the symbol.</td>
</tr>
<tr>
<td>![Arrow Down]</td>
<td>Collapse the selected open call tree.</td>
</tr>
<tr>
<td>![Arrow Right]</td>
<td>Open the selected open call tree.</td>
</tr>
<tr>
<td>![Arrow Right and Left]</td>
<td>Show only the deepest call path through the selected call tree.</td>
</tr>
</tbody>
</table>
Memory usage window

The Memory Usage window displays a graphical summary of how memory has been used in each memory segment of a linked application.

Each bar represents an entire memory segment. Green represents the area of the segment that contains code or data.

To activate the Memory Usage window:

Choose View > Memory Usage or press Ctrl+Alt+Z.

The memory-usage graph will only be visible if your active project's target is an executable file and the file exists. If the executable file has not been linked by SEGGER Embedded Studio, memory-usage information may not be available.

Displaying section information

The Memory Usage window can also be used to visualize how program sections have been placed in memory. To display the program sections, simply click the memory segment to expand it; or, alternatively, right-click and choose Show Memory Sections from the shortcut menu.
Each bar represents an entire memory segment. Green represents the area of the segment that contains the program section.

**Displaying segment overflow**

The Memory Usage window also displays segment overflows when the total size of the program sections placed in a segment is larger than the segment size. When this happens, the segment and section bars represent the total memory used, green areas represent the code or data within the segment, and red areas represent code or data placed outside the segment.
Getting more-detailed information

If you require more-detailed information than that provided by the Memory Usage window, such as the location of specific objects within memory, use the Symbol browser window.
**Bookmarks window**

The **Bookmarks** window contains a list of bookmarks that are set in the project. The bookmarks are stored in the session file associated with the project and persist across runs of SEGGER Embedded Studio if you remove the session file, the bookmarks associated with the project are lost.

**User interface**

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Bookmark" /></td>
<td>Toggle a bookmark at the insertion point in the active editor. Equivalent to choosing <em>Edit &gt; Bookmarks &gt; Toggle Bookmark</em> or pressing Ctrl+F2.</td>
</tr>
<tr>
<td><img src="image" alt="Previous Bookmark" /></td>
<td>Go to the previous bookmark in the bookmark list. Equivalent to choosing <em>Edit &gt; Bookmarks &gt; Previous Bookmark</em> or pressing Alt+Shift+F2.</td>
</tr>
<tr>
<td><img src="image" alt="Next Bookmark" /></td>
<td>Go to the next next bookmark in the bookmark list. Equivalent to choosing <em>Edit &gt; Bookmarks &gt; Next Bookmark</em> or pressing Alt+F2.</td>
</tr>
<tr>
<td><img src="image" alt="Clear All Bookmarks" /></td>
<td>Clear all bookmarks you confirm the action using a dialog. Equivalent to choosing <em>Edit &gt; Bookmarks &gt; Clear All Bookmarks</em> or pressing Ctrl+K, Alt+F2.</td>
</tr>
<tr>
<td><img src="image" alt="Fill Color" /></td>
<td>Selects the fill color for newly created bookmarks.</td>
</tr>
</tbody>
</table>

Double-clicking a bookmark in the bookmark list moves focus to the the bookmark.

You can set bookmarks with the mouse or using keystrokes see **Using bookmarks**.
Code Outline Window

The Code Outline window shows the structure of the text of the focused code editor. For C and C++ documents the top level symbols and types are displayed, for XML documents the nodes are displayed. For C and C++ documents the Preview tab can display documentation on the top level symbols and types. The de facto standard doxygen commands are supported for example:

```c
/**
 * \brief Convert a given full parsed comment to an XML document.
 * 
 * A Relax NG schema for the XML can be found in comment-xml-schema.rng file
 * inside clang source tree.
 * 
 * \param Comment a \c CXComment_FullComment AST node.
 * 
 * \returns string containing an XML document.
 */
CINDEX_LINKAGE CXString clang_FullComment_getAsXML(CXComment Comment);
```
Editing your code

SEGGER Embedded Studio has a built-in editor that allows you to edit text, but some features make it particularly well suited to editing code.

You can open multiple code editors to browse or edit project source code, and you can copy and paste among them. The Windows menu contains a list of all open code editors.

The code editor supports the language of the source file it is editing, showing code with syntax highlighting and offering smart indenting.

You can open a code editor in several ways, some of which are:

- By double-clicking a file in the Project Explorer or by right-clicking a file and selecting Open from the shortcut menu.
- Using the File > New File or File > Open commands.

Elements of the code editor

The code editor is composed of several elements, which are described here.

Code pane:The area where you edit code. You can set options that affect the code pane's text indents, tabs, drag-and-drop behavior, and so forth.

Margin gutter:A gray area on the left side of the code editor where margin indicators such as breakpoints, bookmarks, and shortcuts are displayed. Clicking this area sets a breakpoint on the corresponding line of code.

Horizontal and vertical scroll bars:You can scroll the code pane horizontally and vertically to view code that extends beyond the edges of the pane.
Basic editing

This section is a whirlwind tour of the basic editing features SEGGER Embedded Studio's code editor provides.

Whether you are editing code, HTML, or plain text, the code editor is just like many other text editors or word processors. For code that is part of a project, the project’s programming language support provides syntax highlighting (colorization), indentation, and so on.

This section is not a reference for everything the code editor provides; for that, look in the following sections.
Moving the insertion point

The most common way to navigate through text is to use the mouse or the keyboard's cursor keys.

Using the mouse

You can move the insertion point within a document by clicking the mouse inside the editor window.

Using the keyboard

The keystrokes most commonly used to navigate through a document are:

<table>
<thead>
<tr>
<th>Keystroke</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up</td>
<td>Move the insertion point up one line</td>
</tr>
<tr>
<td>Down</td>
<td>Move the insertion point down one line</td>
</tr>
<tr>
<td>Left</td>
<td>Move the insertion point left one character</td>
</tr>
<tr>
<td>Right</td>
<td>Move the insertion point right one character</td>
</tr>
<tr>
<td>Home</td>
<td>Move the insertion point to the first non-whitespace character on the line pressing Home a second time moves the insertion point to the leftmost column</td>
</tr>
<tr>
<td>End</td>
<td>Move the insertion point to the end of the line</td>
</tr>
<tr>
<td>PageUp</td>
<td>Move the insertion point up one page</td>
</tr>
<tr>
<td>PageDown</td>
<td>Move the insertion point down one page</td>
</tr>
<tr>
<td>Ctrl+Home</td>
<td>Move the insertion point to the start of the document</td>
</tr>
<tr>
<td>Ctrl+End</td>
<td>Move the insertion point to the end of the document</td>
</tr>
<tr>
<td>Ctrl+Left</td>
<td>Move the insertion point left one word</td>
</tr>
<tr>
<td>Ctrl+Right</td>
<td>Move the insertion point right one word</td>
</tr>
</tbody>
</table>

SEGGER Embedded Studio offers additional movement keystrokes, though most users are more comfortable using repeated simple keystrokes to accomplish the same thing:

<table>
<thead>
<tr>
<th>Keystroke</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+Up</td>
<td>Move the insertion point up five lines</td>
</tr>
<tr>
<td>Alt+Down</td>
<td>Move the insertion point down five lines</td>
</tr>
<tr>
<td>Alt+Home</td>
<td>Move the insertion point to the top of the window</td>
</tr>
<tr>
<td>Alt+End</td>
<td>Move the insertion point to the bottom of the window</td>
</tr>
<tr>
<td>Ctrl+Up</td>
<td>Scroll the document up one line in the window without moving the insertion point</td>
</tr>
</tbody>
</table>
**Ctrl+Down**  
Scroll the document down one line in the window without moving the insertion point

If you are editing source code, there are source-related keystrokes too:

<table>
<thead>
<tr>
<th>Keystroke</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+PgUp</td>
<td>Move the insertion point backwards to the previous function or method.</td>
</tr>
<tr>
<td>Ctrl+PgDn</td>
<td>Move the insertion point forwards to the next function or method.</td>
</tr>
</tbody>
</table>
Adding text

The editor has two text-input modes:

- **Insertion mode:** As you type on the keyboard, text is entered at the insertion point and any text to the right of the insertion point is shifted along. A visual indication of insertion mode is that the cursor is a flashing line.

- **Overstrike mode:** As you type on the keyboard, text at the insertion point is replaced with your typing. A visual indication of insertion mode is that the cursor is a flashing block.

Insert and overstrike modes are common to all editors: if one editor is in insert mode, all editors are in insert mode. To configure the cursor appearance, choose **Tools > Options**.

**To toggle between insertion and overstrike mode:**

Click **Insert**.

When overstrike mode is enabled, the mode indicator changes from **INS** to **OVR** and the cursor will change to the overstrike cursor.

**To add or insert text:**

1. Move the insertion point to the place text is to be inserted.
2. Enter the text using the keyboard.

To overwrite characters in an existing line, press the **Insert** key to place the editor into overstrike mode.

**To add or insert text on multiple lines:**

1. Hold down the **Alt** key and use block selection to mark the place text is to be inserted.
2. Enter the text using the keyboard.
Deleting text

The text editor supports the following common editing keystrokes:

<table>
<thead>
<tr>
<th>Keystroke</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backspace</td>
<td>Delete the character before the insertion point</td>
</tr>
<tr>
<td>Delete</td>
<td>Delete the character after the insertion point</td>
</tr>
<tr>
<td>Ctrl+Backspace</td>
<td>Delete one word before the insertion point</td>
</tr>
<tr>
<td>Ctrl+Delete</td>
<td>Delete one word after the insertion point</td>
</tr>
</tbody>
</table>

To delete characters or words:
1. Place the insertion point before the word or letter you want to delete.
2. Press **Delete** as many times as needed.

or

1. Place the insertion point after the letter or word you want to delete.
2. Press **Backspace** as many times as needed.

To delete text that spans more than a few characters:
1. Select the text you want to delete.
2. Press **Delete** or **Backspace** to delete it.

To delete a text block:
1. Hold down the **Alt** key and use block selection to mark the text you want to delete.
2. Press **Delete** or **Backspace** to delete it.

To delete characters on multiple lines:
1. Hold down the **Alt** key and use block selection to mark the lines.
2. Press **Delete** or **Backspace** as many times as needed to delete the characters.
Using the clipboard

You can select text by using the keyboard or the mouse.

**To select text with the keyboard:**

Hold down the **Shift** key while using the cursor keys.

**To select text with the mouse:**

1. Click the start of the selection.
2. Drag the mouse to mark the selection.
3. Release the mouse to end selecting.

**To select a block of text with the keyboard:**

Hold down the **Shift+Alt** keys while using the cursor keys.

**To select a block of text with the mouse:**

1. Hold down the **Alt** key.
2. Click the start of the selection.
3. Drag the mouse to mark the selection.
4. Release the mouse to end selecting.

**To copy selected text to the clipboard:**

Choose **Edit > Copy** or press **Ctrl+C**.

The standard Windows key sequence **Ctrl+Ins** also copies text to the clipboard.

**To cut selected text to the clipboard:**

Choose **Edit > Cut** or press **Ctrl+X**.

The standard Windows key sequence **Shift+Del** also cuts text to the clipboard.

**To insert the clipboard content at the insertion point:**

Choose **Edit > Paste** or press **Ctrl+V**.

The standard Windows key sequence **Shift+Ins** also inserts the clipboard content at the insertion point.
Undo and redo

The editor has an *undo* facility to undo previous editing actions. The *redo* feature can be used to re-apply previously undone actions.

**To undo one editing action:**

Choose *Edit > Undo* or press **Ctrl+Z**.

The standard Windows key sequence **Alt+Backspace** also undoes an edit.

**To undo multiple editing actions:**

1. On the *Standard* toolbar, click the arrow next to the *Undo* button.
2. Select the editing operations to undo.

**To undo all edits:**

Choose *Edit > Others > Undo All* or press **Ctrl+K, Ctrl+Z**.

**To redo one editing action:**

Choose *Edit > Redo* or press **Ctrl+Y**.

The standard Windows key sequence **Alt+Shift+Backspace** also redoes an edit.

**To redo multiple editing actions:**

1. On the *Standard* toolbar, click the arrow next to the *Redo* tool button.
2. From the pop-up menu, select the editing operations to redo.

**To redo all edits:**

Choose *Edit > Others > Redo All* or press **Ctrl+K, Ctrl+Y**.
Drag and drop

You can select text, then drag it to another location. You can drop the text at a different location in the same window or in another one.

To drag and drop text:

1. Select the text you want to move.
2. Press and hold the mouse button to drag the selected text to where you want to place it.
3. Release the mouse button to drop the text.

Dragging text moves it to the new location. To copy it to a new location, hold down the Ctrl key while dragging the text: the mouse pointer changes to indicate a copy operation. Press the Esc key while dragging text to cancel the drag-and-drop edit.

By default, drag-and-drop editing is disabled and you must enable it if you want to use it.

To enable or disable drag-and-drop editing:

1. Choose Tools > Options or press Alt+,.
2. Click Text Editor.
3. Set Allow Drag and Drop Editing to Yes to enable or to No to disable drag-and-drop editing.
Searching

To find text in the current file:

1. Press Ctrl+F.
2. Enter the string to search for.

As you type, the editor searches the file for a match. The pop-up shows how many matches are in the current file. To move through the matches while the Find box is still active, press Tab or F3 to move to the next match and Shift+Tab or Shift+F3 to move to the previous match.

If you press Ctrl+F a second time, SEGGER Embedded Studio pops up the standard Find dialog to search the file. If you wish to bring up the Find dialog without pressing Ctrl+F twice, choose Search > Find.
Advanced editing

You can do anything using its basic code-editing features, but the SEGGER Embedded Studio text editor has a host of labor-saving features that make editing programs a snap.

This section describes the code-editor features intended to make editing source code easier.
Indenting source code

The editor uses the Tab key to increase or decrease the indentation level of the selected text.

**To increase indentation:**

Select the text to indent.
Choose **Selection > Increase Line Indent** or press Tab.

**To decrease indentation:**

Select the text to indent.
Choose **Selection > Decrease Line Indent** or press Shift+Tab.

The indentation size can be changed in the **Language Properties** pane of the editor’s **Properties** window, as can all the indent-related features listed below.

**To change indentation size:**

Choose **Tools > Options** or press Alt+.,
Select the **Languages** page.
Set the **Indent Size** environment option for the required language.

You can choose to use spaces or tab characters to fill whitespace when indenting.

**To set tab or space fill when indenting:**

Choose **Tools > Options** or press Alt+.,
Select the **Languages** page.
Set the **Use Tabs** environment option for the required language. **Note:** changing this setting does not add or remove existing tabs from files, the change will only affect new indents.

The editor can assist with source code indentation while inserting text. There are three levels of indentation assistance:

- **None:** The indentation of the source code is left to the user.
- **Indent:** This is the default. The editor maintains the current indentation level. When you press Return or Enter, the editor moves the insertion point down one line and indented to the same level as the now-previous line.
- **Smart:** The editor analyzes the source code to compute the appropriate indentation level for each line. You can change how many lines before the insertion point will be analyzed for context. The smart-indent mode can be configured to indent either open and closing braces or the lines following the braces.

**Changing indentation options:**

To change the indentation mode:
Set the **Indent Mode** environment option for the required language.

To change whether opening braces are indented in smart-indent mode:

Set the **Indent Opening Brace** environment option for the required language.

To change whether closing braces are indented in smart-indent mode:

Set the **Indent Closing Brace** environment option for the required language.

To change the number of previous lines used for context in smart-indent mode:

Set the **Indent Context Lines** environment option for the required language.
Commenting out sections of code

To comment selected text:

Choose Selection > Comment or press Ctrl+/. 

To uncomment selected text:

Choose Selection > Uncomment or press Ctrl+Shift+/. 

You can also toggle the commenting of a selection by typing / . This has no menu equivalent.
Adjusting letter case

The editor can change the case of the current word or the selection. The editor will change the case of the selection, if there is a selection, otherwise it will change the case of word at the insertion point.

To change text to uppercase:

Choose Selection > Make Uppercase or press Ctrl+Shift+U.

This changes, for instance, Hello to HELLO.

To change text to lowercase:

Choose Selection > Make Lowercase or press Ctrl+U.

This changes, for instance, Hello to hello.

To switch between uppercase and lowercase:

Choose Selection > Switch Case.

This changes, for instance, Hello to hELLO.

With large software teams or imported source code, sometimes identifiers don't conform to your local coding style. To assist in conversion between two common coding styles for identifiers, SEGGER Embedded Studio's editor offers the following two shortcuts:

To change from split case to camel case:

Choose Selection > Camel Case or press Ctrl+K, Ctrl+Shift+U.

This changes, for instance, this_is_wrong to thisIsWrong.

To change from camel case to split case:

Choose Selection > Split Case or press Ctrl+K, Ctrl+U.

This changes, for instance, thisIsWrong to this_is_wrong.
Using bookmarks

To edit a document elsewhere and then return to your current location, add a bookmark. The Bookmarks window maintains a list of the bookmarks set in source files see Bookmarks window.

To place a bookmark:

1. Move the insertion point to the line you wish to bookmark.
2. Choose Edit > Bookmarks > Toggle Bookmark or press Ctrl+F2.

A bookmark symbol appears next to the line in the indicator margin to show the bookmark is set.

To place a bookmark using the mouse:

1. Right-click the margin gutter where the bookmark should be set.
2. Choose Toggle Bookmark.

The default color to use for new bookmarks is configured in the Bookmarks window. You can choose a specific color for the bookmark as follows:

1. Press and hold the Alt key.
2. Click the margin gutter where the bookmark should be set.
3. From the palette, click the bookmark color to use for the bookmark.

To navigate forward through bookmarks:

2. The editor moves the insertion point to the next bookmark in the document.

If there is no following bookmark, the insertion point moves to the first bookmark in the document.

To navigate backward through bookmarks:

1. Choose Edit > Bookmarks > Previous Bookmark In Document or press Shift+F2.
2. The editor moves the insertion point to the previous bookmark in the document.

If there is no previous bookmark, the insertion point moves to the last bookmark in the document.

To remove a bookmark:

1. Move the insertion point to the line containing the bookmark.
2. Choose Edit > Bookmarks > Toggle Bookmark or press Ctrl+F2.

The bookmark symbol disappears, indicating the bookmark is no longer set.

To remove all bookmarks in a document:

# Quick reference for bookmark operations

<table>
<thead>
<tr>
<th>Keystroke</th>
<th>Menu</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+F2</td>
<td>Edit &gt; Bookmarks &gt; Toggle Bookmark</td>
<td>Toggle a bookmark at the insertion point.</td>
</tr>
<tr>
<td>Ctrl+K, 0</td>
<td>Clear the bookmark at the insertion point.</td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td>Move the insertion point to next bookmark in the document.</td>
<td></td>
</tr>
<tr>
<td>Shift+F2</td>
<td>Move the insertion point to previous bookmark in the document.</td>
<td></td>
</tr>
<tr>
<td>Ctrl+Q, F2</td>
<td>Move the insertion point to the first bookmark in the document.</td>
<td></td>
</tr>
<tr>
<td>Ctrl+Q, Shift+F2</td>
<td>Move the insertion point to the last bookmark in the document.</td>
<td></td>
</tr>
<tr>
<td>Ctrl+K, F2</td>
<td>Clear all bookmarks in the document.</td>
<td></td>
</tr>
<tr>
<td>Alt+F2</td>
<td>Move the insertion point to the next bookmark in the Bookmarks list.</td>
<td></td>
</tr>
<tr>
<td>Alt+Shift+F2</td>
<td>Move the insertion point to the previous bookmark in the Bookmarks list.</td>
<td></td>
</tr>
<tr>
<td>Ctrl+Q, Alt+F2</td>
<td>Move the insertion point to the first bookmark in the Bookmarks list.</td>
<td></td>
</tr>
<tr>
<td>Ctrl+Q, Alt+Shift+F2</td>
<td>Move the insertion point to the last bookmark in the Bookmarks list.</td>
<td></td>
</tr>
<tr>
<td>Ctrl+K, Alt+F2</td>
<td>Clear all bookmarks in all documents.</td>
<td></td>
</tr>
</tbody>
</table>
Find and Replace window

The Find and Replace window allows you to search for and replace text in the current document or in a range of specified files.

To activate the Find and Replace window:

Choose Search > Replace in Files or press Ctrl+Alt+F.

To find text in a single file:

Select Current Document in the context combo box. Enter the string to be found in the text edit input. If the search will be case sensitive, set the Match case option. If the search will be for a whole word (i.e., there will be whitespace, such as spaces or the beginning or end of the line, on both sides of the string being searched for) set the Whole word option. If the search string is a regular expression, set the Use regexp option. Click the Find button to find all occurrences of the string in the current document.

To find and replace text in a single file:

Click the Replace button on the toolbar. Enter the string to search for into the Find what input. Enter the replacement string into the Replace with input. If the search string is a regular expression, the \n back-reference can be used in the replacement string to reference captured text. If the search will be case sensitive, set the Match case option. If the search will be for a whole word (i.e., there will be whitespace, such as spaces or the beginning or end of the line, on both sides of the string being searched for) set the Match whole word option. If the search string is a regular expression, set the Use regular expression option. Click the Find Next button to find next occurrence of the string, then click the Replace button to replace the found string with the replacement string; or click Replace All to replace all occurrences of the search string without prompting.

To find text in multiple files:

Click the Find In Files button on the toolbar. Enter the string to search for into the Find what input. Select the appropriate option in the Look in input to select whether to carry out the search in all open documents, all documents in the current project, all documents in the current solution, or all files in a specified folder. If you have specified that you want to search in a folder, select the folder you want to search by entering its path in the Folder input and use the Look in files matching input to specify the type of files you want to search.
If the search will be case sensitive, set the **Match case** option.
If the search will be for a whole word—i.e., there will be whitespace, such as spaces or the beginning or end of the line, on both sides of the string being searched for—set the **Match whole word** option.
If the search string is a regular expression, set the **Use regular expression** option.
Click the **Find All** button to find all occurrences of the string in the specified files, or click the **Bookmark All** button to bookmark all the occurrences of the string in the specified files.

**To replace text in multiple files:**

Click the **Replace In Files** button on the toolbar.
Enter the string to search for into the **Find what** input.
Enter the replacement string into the **Replace with** input. If the search string is a regular expression, the \( n \) back-reference can be used in the replacement string to reference captured text.
Select the appropriate option in the **Look in** input to select whether you want to carry out the search and replace in the current or in all open documents.
If you have specified that you want to search in a folder, select the folder you want to search by entering its path in the **Folder** input and use the **Look in files matching** input to specify the type of files you want to search.
If the search will be case sensitive, set the **Match case** option.
If the search will be for a whole word—i.e., there will be whitespace, such as spaces or the beginning or end of the line, on both sides of the string being searched for—set the **Match whole word** option.
If the search string is a regular expression, set the **Use regular expression** option.
Click the **Replace All** button to replace all occurrences of the string in the specified files.
Clipboard Ring window

The code editor captures all cut and copy operations, and stores the cut or copied item on the clipboard ring. The clipboard ring stores the last 20 cut or copied text items, but you can configure the maximum number by using the environment options dialog. The clipboard ring is an excellent place to store scraps of text when you’re working with many documents and need to cut and paste between them.

To activate the clipboard ring:

Choose Edit > Clipboard Ring > Clipboard Ring or press Ctrl+Alt+C.

To paste from the clipboard ring:

1. Cut or copy some text from your code. The last item you cut or copy into the clipboard ring is the current item for pasting.
2. Press Ctrl+Shift+V to paste the clipboard ring's current item into the current document.
3. Repeatedly press Ctrl+Shift+V to cycle through the entries in the clipboard ring until you get to the one you want to permanently paste into the document. Each time you press Ctrl+Shift+V, the editor replaces the last entry you pasted from the clipboard ring, so you end up with just the last one you selected. The item you stop on then becomes the current item.
4. Move to another location or cancel the selection. You can use Ctrl+Shift+V to paste the current item again or to cycle the clipboard ring to a new item.

Clicking an item in the clipboard ring makes it the current item.

To paste a specific item from the clipboard ring:

1. Move the insertion point to the position to paste the item in the document.
2. Click the arrow at the right of the item to paste.
3. Choose Paste from the pop-up menu.

or

1. Click the item to paste to make it the current item.
2. Move the insertion point to the position to paste the item in the document.
3. Press Ctrl+Shift+V.

To paste all items into a document:

To paste all items on the clipboard ring into the current document, move the insertion point to where you want to paste the items and do one of the following:

Choose Edit > Clipboard Ring > Paste All.

or
On the Clipboard Ring toolbar, click the Paste All button.

**To remove an item from the clipboard ring:**

1. Click the arrow at the right of the item to remove.
2. Choose Delete from the pop-up menu.

**To remove all items from the clipboard ring:**

Choose Edit > Clipboard Ring > Clear Clipboard Ring.

or

On the Clipboard Ring toolbar, click the Clear Clipboard Ring button.

**To configure the clipboard ring:**

1. Choose Tools > Options or press Alt+,.
2. Click the Windows category to show the Clipboard Ring Options group.
3. Select Preserve Contents Between Runs to save the content of the clipboard ring between runs, or deselect it to start with an empty clipboard ring.
4. Change Maximum Items Held In Ring to configure the maximum number of items stored on the clipboard ring.
Mouse-click accelerators

SEGGER Embedded Studio provides a number of mouse-click accelerators in the editor that speed access to commonly used functions. The mouse-click accelerators are user configurable using Tools > Options.

Default mouse-click assignments

<table>
<thead>
<tr>
<th>Click</th>
<th>Default</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Not configurable start selection.</td>
<td></td>
</tr>
<tr>
<td>Shift+Left</td>
<td>Not configurable extend selection.</td>
<td></td>
</tr>
<tr>
<td>Ctrl+Left</td>
<td>Select word.</td>
<td></td>
</tr>
<tr>
<td>Alt+Left</td>
<td>Execute Go To Definition.</td>
<td></td>
</tr>
<tr>
<td>Middle</td>
<td>No action.</td>
<td></td>
</tr>
<tr>
<td>Shift+Middle</td>
<td>Display Go To Include menu.</td>
<td></td>
</tr>
<tr>
<td>Ctrl+Middle</td>
<td>No action.</td>
<td></td>
</tr>
<tr>
<td>Alt+Middle</td>
<td>Display Go To Method menu.</td>
<td></td>
</tr>
<tr>
<td>Right</td>
<td>Not configurable show context menu.</td>
<td></td>
</tr>
<tr>
<td>Shift+Right</td>
<td>No action.</td>
<td></td>
</tr>
<tr>
<td>Ctrl+Right</td>
<td>No action.</td>
<td></td>
</tr>
<tr>
<td>Alt+Right</td>
<td>Execute Find References.</td>
<td></td>
</tr>
</tbody>
</table>

Each accelerator can be assigned one of the following actions:

- **Default:** The system default for that click.
- **Go To Definition:** Go to the definition of the item clicked, equivalent to choosing Navigate > Go To Definition or pressing Alt+G.
- **Find References:** Find references to the item clicked, equivalent to choosing Navigate > Find References or pressing Alt+R.
- **Find in Solution:** Textually find the item clicked in all the files in the solution, equivalent to choosing Search > Find Extras > Find in Solution or pressing Alt+U.
- **Find Help:** Use F1-help on the item clicked, equivalent to choosing Help > Help or pressing F1.
- **Go To Method:** Display the Go To Method menu, equivalent to choosing Navigate > Find Method or pressing Ctrl+M.
- **Go To Include:** Display the Go To Include menu, equivalent to choosing Navigate > Find Include or pressing Ctrl+Shift+M.
- **Paste:** Paste the clipboard at the position clicked, equivalent to choosing Edit > Paste or pressing Ctrl+V.

Configuring Mac OS X

On Mac OS X you must configure the mouse to pass middle clicks and right clicks to the application if you wish to use mouse-click accelerators in SEGGER Embedded Studio. Configure the mouse preferences in the Mouse control panel in Mac OS X System Preferences to the following:
Right mouse button set to **Secondary Button**.
Middle mouse button set to **Button 3**.
Regular expressions

The editor can search and replace text using regular expressions. A regular expression is a string that uses special characters to describe and reference patterns of text. The regular expression system used by the editor is modeled on Perl’s regexp language. For more information on regular expressions, see Mastering Regular Expressions, Jeffrey E F Freidl, ISBN 0596002890.

Summary of special characters

The following table summarizes the special characters the SEGGER Embedded Studio editor supports

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\d</td>
<td>Match a numeric character.</td>
</tr>
<tr>
<td>\D</td>
<td>Match a non-numeric character.</td>
</tr>
<tr>
<td>\s</td>
<td>Match a whitespace character.</td>
</tr>
<tr>
<td>\S</td>
<td>Match a non-whitespace character.</td>
</tr>
<tr>
<td>\w</td>
<td>Match a word character.</td>
</tr>
<tr>
<td>\W</td>
<td>Match a non-word character.</td>
</tr>
<tr>
<td>[c]</td>
<td>Match set of characters; e.g., [ch] matches characters c or h. A range can be specified using the - character; e.g., [0-27-9] matches if the character is 0, 1, 2, 7, 8, or 9. A range can be negated using the ^ character; e.g., [^a-z] matches if the character is anything other than a lowercase alphabetic character.</td>
</tr>
</tbody>
</table>
| \c      | Match the literal character c. For example, you would use \* to match the character *.
| \a      | Match ASCII bell character (ASCII code 7).
| \f      | Match ASCII form feed character (ASCII code 12).
| \t      | Match ASCII horizontal tab character (ASCII code 9).
| \v      | Match ASCII vertical tab character.
| \xhhh   | Match Unicode character specified by hexadecimal number hhhh. |
| .       | Match any character. |
| *       | Match zero or more occurrences of the preceding expression. |
| +       | Match one or more occurrences of the preceding expression. |
| ?       | Match zero or one occurrences of the preceding expression. |
### Examples

The following regular expressions can be used with the editor's search-and-replace operations. To use the regular expression mode, the **Use regular expression** checkbox must be set in the search-and-replace dialog. Once enabled, regular expressions can be used in the **Find what** search string. The **Replace With** strings can use the "n" back-reference string to reference any captured strings.

<table>
<thead>
<tr>
<th>&quot;Find what&quot;</th>
<th>&quot;Replace With&quot;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>u\w.d</code></td>
<td></td>
<td>Search for any-length string containing one or more word characters beginning with the character u and ending in the character d.</td>
</tr>
<tr>
<td><code>^.*;$</code></td>
<td></td>
<td>Search for any lines ending in a semicolon.</td>
</tr>
<tr>
<td><code>(typedef.+\s+)(\S+);</code></td>
<td><code>\1TEST_\2;</code></td>
<td>Find C type definition and insert the string TEST onto the beginning of the type name.</td>
</tr>
</tbody>
</table>
Locals window

The Locals window displays a list of all variables that are in scope of the selected stack frame in the Call Stack.

The Locals window has a toolbar and a main data display.

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X2</td>
<td>Display the selected item in binary.</td>
</tr>
<tr>
<td>X8</td>
<td>Display the selected item in octal.</td>
</tr>
<tr>
<td>X10</td>
<td>Display the selected item in decimal.</td>
</tr>
<tr>
<td>X16</td>
<td>Display the selected item in hexadecimal.</td>
</tr>
<tr>
<td>X10+</td>
<td>Display the selected item as a signed decimal.</td>
</tr>
<tr>
<td>'X'</td>
<td>Display the selected item as a character or Unicode character.</td>
</tr>
<tr>
<td>🎉</td>
<td>Set the range displayed in the active Memory window to span the memory allocated to the selected item.</td>
</tr>
<tr>
<td>A</td>
<td>Z</td>
</tr>
<tr>
<td>0</td>
<td>9</td>
</tr>
</tbody>
</table>

Using the Locals window

The Locals window shows the local variables of the active function when the debugger is stopped. The contents of the Locals window changes when you use the Debug Location toolbar items or select a new frame in the Call Stack window. When the program stops at a breakpoint, or is stepped, the Locals window updates to show the active stack frame. Items that have changed since they were previously displayed are highlighted in red.

To activate the Locals window:

Choose View > Locals or press Ctrl+Alt+L.

When you select a variable in the main part of the display, the display-format button highlighted on the Locals window toolbar changes to show the selected item’s display format.

To change the display format of a local variable:

Right-click the item to change.
From the shortcut menu, choose the desired display format.
To modify the value of a local variable:

Click the value of the local variable to modify.
Enter the new value for the local variable. Prefix hexadecimal numbers with 0x, binary numbers with 0b, and octal numbers with 0.

or

Right-click the value of the local variable to modify.
From the shortcut menu, select one of the commands to modify the local variable's value.
Globals window

The **Globals** window displays a list of all variables that are global to the program. The operations available on the entries in this window are the same as the **Watch** window, except you cannot add or delete variables from the **Globals** window.

Globals window user interface

The **Globals** window consists of a toolbar and main data display.

Globals toolbar

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Binary" /></td>
<td>Display the selected item in binary.</td>
</tr>
<tr>
<td><img src="image" alt="Octal" /></td>
<td>Display the selected item in octal.</td>
</tr>
<tr>
<td><img src="image" alt="Decimal" /></td>
<td>Display the selected item in decimal.</td>
</tr>
<tr>
<td><img src="image" alt="Hexadecimal" /></td>
<td>Display the selected item in hexadecimal.</td>
</tr>
<tr>
<td><img src="image" alt="Signed Decimal" /></td>
<td>Display the selected item as a signed decimal.</td>
</tr>
<tr>
<td><img src="image" alt="Character" /></td>
<td>Display the selected item as a character or Unicode character.</td>
</tr>
<tr>
<td><img src="image" alt="Set Range" /></td>
<td>Set the range displayed in the active <strong>Memory</strong> window to span the memory allocated to the selected item.</td>
</tr>
<tr>
<td><img src="image" alt="Sort Alphabetically" /></td>
<td>Sort variables alphabetically by name.</td>
</tr>
<tr>
<td><img src="image" alt="Sort Numerically" /></td>
<td>Sort variables numerically by address or register number (default).</td>
</tr>
</tbody>
</table>

Using the Globals window

The **Globals** window shows the global variables of the application when the debugger is stopped. When the program stops at a breakpoint, or is stepped, the **Globals** window updates to show the active stack frame and new variable values. Items that have changed since they were previously displayed are highlighted in red.

To activate the Globals window:

Choose View > **Globals** or press Ctrl+Alt+G.
Changing the display format

When you select a variable in the main part of the display, the display-format button highlighted on the **Globals** window toolbar changes to show the item's display format.

**To change the display format of a global variable:**

- Right-click the item to change.
  - From the shortcut menu, choose the desired display format.

  or

- Click the item to change.
  - On the **Globals** window toolbar, select the desired display format.

**To modify the value of a global variable:**

- Click the value of the global variable to modify.
  - Enter the new value for the global variable. Prefix hexadecimal numbers with **0x**, binary numbers with **0b**, and octal numbers with **0**.
Watch window

The Watch window provides a means to evaluate expressions and to display the results of those expressions. Typically, expressions are just the name of a variable to be displayed, but they can be considerably more complex; see Debug expressions. Note: expressions are always evaluated when your program stops, so the expression you are watching is the one that is in scope of the stopped program position.

The Watch window is divided into a toolbar and the main data display.

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X₂</td>
<td>Display the selected item in binary.</td>
</tr>
<tr>
<td>X₈</td>
<td>Display the selected item in octal.</td>
</tr>
<tr>
<td>X₁₀</td>
<td>Display the selected item in decimal.</td>
</tr>
<tr>
<td>X₁₆</td>
<td>Display the selected item in hexadecimal.</td>
</tr>
<tr>
<td>X⁻₁₀</td>
<td>Display the selected item as a signed decimal.</td>
</tr>
<tr>
<td>X'</td>
<td>Display the selected item as a character or Unicode character.</td>
</tr>
<tr>
<td></td>
<td>Set the range displayed in the active Memory window to span the memory allocated to the selected item.</td>
</tr>
<tr>
<td></td>
<td>Remove the selected watch item.</td>
</tr>
<tr>
<td></td>
<td>Remove all the watches.</td>
</tr>
</tbody>
</table>

Right-clicking a watch item shows a shortcut menu with commands that are not available from the toolbar.

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;&quot;</td>
<td>View pointer or array as a null-terminated string.</td>
</tr>
<tr>
<td>[]</td>
<td>View pointer or array as an array.</td>
</tr>
<tr>
<td>🍀</td>
<td>View pointer value.</td>
</tr>
<tr>
<td>🍀 =0</td>
<td>Set watch value to zero.</td>
</tr>
<tr>
<td>🍀 =1</td>
<td>Set watch value to one.</td>
</tr>
<tr>
<td>↑</td>
<td>Increment watched variable by one.</td>
</tr>
<tr>
<td>↓</td>
<td>Decrement watched variable by one.</td>
</tr>
</tbody>
</table>
Negated watched variable.

Invert watched variable.

View the properties of the watch value.

You can view details of the watched item using the Properties dialog.

**Filename**
- The filename context of the watch item.

**Line number**
- The line number context of the watch item.

**(Name)**
- The name of the watch item.

**Address**
- The address or register of the watch item.

**Expression**
- The debug expression of the watch item.

**Previous Value**
- The previous watch value.

**Size In Bytes**
- The size of the watch item in bytes.

**Type**
- The type of the watch item.

**Value**
- The value of the watch item.

### Using the Watch window

Each expression appears as a row in the display. Each row contains the expression and its value. If the value of an expression is structured (for example, an array), you can open the structure to see its contents.

The display updates each time the debugger locates to source code. So it will update each time your program stops on a breakpoint, or single steps, and whenever you traverse the call stack. Items that have changed since they were previously displayed are highlighted in red.

**To activate the Watch window:**

Choose View > Watch > Watch 1 or press Ctrl+T, W, 1.
You can show other Watch windows similarly.

You can add a new expression to be watched by clicking and typing into the last entry in the Watch window. You can change an expression by clicking its entry and editing its contents.

When you select a variable in the main part of the display, the display format button highlighted on the Watch window toolbar changes to show the item's display format.

**To change the display format of an expression:**

Right-click the item to change.
From the shortcut menu, choose the desired display format.

or

Click the item to change.
On the Watch window toolbar, select the desired display format.

The selected display format will then be used for all subsequent displays and will be preserved after the debug session stops.

For C programs, the interpretation of pointer types can be changed by right-clicking and selecting from the shortcut menu. A pointer can be interpreted as:

- a null-terminated ASCII string
- an array
- an integer
dereferenced

**To modify the value of an expression:**

Click the value of the local variable to modify.
Enter the new value of the local variable. Prefix hexadecimal numbers with 0x, binary numbers with 0b, and octal numbers with 0.

or

Right-click the value of the local variable to modify.
From the shortcut menu, choose one of the commands to modify the variable's value.
Register window

The Register windows show the values of both CPU registers and the processor’s special function or peripheral registers. Because microcontrollers are becoming very highly integrated, it’s not unusual for them to have hundreds of special function registers or peripheral registers, so SEGGER Embedded Studio provides four register windows. You can configure each register window to display one or more register groups for the processor being debugged.

A Register window has a toolbar and a main data display.

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="button1" alt="Display the CPU, special function register, and peripheral register groups." /></td>
<td>Display the CPU, special function register, and peripheral register groups.</td>
</tr>
<tr>
<td><img src="button2" alt="Display the CPU registers." /></td>
<td>Display the CPU registers.</td>
</tr>
<tr>
<td><img src="button3" alt="Hide the CPU registers." /></td>
<td>Hide the CPU registers.</td>
</tr>
<tr>
<td><img src="button4" alt="Force-read a register, ignoring the access attribute of the register." /></td>
<td>Force-read a register, ignoring the access attribute of the register.</td>
</tr>
<tr>
<td><img src="button5" alt="Update the selected register group." /></td>
<td>Update the selected register group.</td>
</tr>
<tr>
<td><img src="button6" alt="Set the active memory window to the address and size of the selected register group." /></td>
<td>Set the active memory window to the address and size of the selected register group.</td>
</tr>
</tbody>
</table>

Using the registers window

Both CPU registers and special function registers are shown in the main part of the Registers window. When the program stops at a breakpoint, or is stepped, the Registers windows update to show the current values of the registers. Items that have changed since they were previously displayed are highlighted in red.

To activate the first register window:

Choose View > Registers > Registers 1 or press Ctrl+T, R, 1.

Other register windows can be similarly activated.

Displaying CPU registers

The values of the CPU registers displayed in the Registers window depend up upon the selected context. The selected context can be:

- The register state the CPU stopped in.
- The register state when a function call occurred using the Call Stack window.
The register state of the currently selected thread using the Threads window.
The register state you supplied with the Debug > Locate operation.

To display a group of CPU registers:

On the Registers window toolbar, click the Groups button.
From the pop-up menu, select the register groups to display and deselect the ones to hide.

You can deselect all CPU register groups to allow more space in the display for special function registers or peripheral registers. So, for instance, you can have one register window showing the CPU registers and other register windows showing different peripheral registers.

Displaying special function or peripheral registers

The Registers window shows the set of register groups defined in the memory-map file the application was built with. If there is no memory-map file associated with a project, the Registers window will show only the CPU registers.

To display a special function or peripheral register:

On the Registers toolbar, click the Groups button.
From the pop-up menu, select the register groups to display and deselect the ones to hide.

Changing display format

When you select a register in the main part of the display, the display-format button highlighted on the Registers window toolbar changes to show the item's display format.

To change the display format of a register:

Right-click the item to change.
From the shortcut menu, choose the desired display format.

or

Click the item to change.
On the Registers window toolbar, select the desired display format.

Modifying register values

To modify the value of a register:

Click the value of the register to modify.
Enter the new value for the register. Prefix hexadecimal numbers with 0x, binary numbers with 0b, and octal numbers with 0.

or

Right-click the value of the register to modify.
From the shortcut menu, choose one of the commands to modify the register value.

Modifying the saved register value of a function or thread may not be supported.
Memory window

The Memory window shows the contents of the connected target's memory areas and allows the memory to be edited. SEGGER Embedded Studio provides four memory windows, you can configure each memory window to display different memory ranges.

The Memory window has a toolbar and a data display/edit area

<table>
<thead>
<tr>
<th>Field/Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Address</strong></td>
<td>Address to display. This can be a numeric value or a debug expression.</td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td>Number of bytes to display. This can be a number or a debug expression. If unspecified, the number of bytes required to fill the window will be automatically calculated.</td>
</tr>
<tr>
<td><strong>Columns</strong></td>
<td>Number of columns to display. If unspecified, the number of columns required to fill the window will be automatically calculated.</td>
</tr>
<tr>
<td><strong>X₂</strong></td>
<td>Select binary display.</td>
</tr>
<tr>
<td><strong>X₈</strong></td>
<td>Select octal display.</td>
</tr>
<tr>
<td><strong>X₁₀</strong></td>
<td>Select unsigned decimal display.</td>
</tr>
<tr>
<td><strong>X₁₀⁻¹</strong></td>
<td>Select signed decimal display.</td>
</tr>
<tr>
<td><strong>X₁₆</strong></td>
<td>Select hexadecimal display (default).</td>
</tr>
<tr>
<td><strong>₈⁻⁻</strong></td>
<td>Select byte display (default).</td>
</tr>
<tr>
<td><strong>₁₆⁻⁻</strong></td>
<td>Select 2-byte display.</td>
</tr>
<tr>
<td><strong>₃₂⁻⁻</strong></td>
<td>Select 4-byte display.</td>
</tr>
<tr>
<td>💡💡💡💡</td>
<td>Display both data and text (default).</td>
</tr>
<tr>
<td>💡💡💡</td>
<td>Display data only.</td>
</tr>
<tr>
<td>💡💡</td>
<td>Display text only.</td>
</tr>
<tr>
<td>💡💡💡💍</td>
<td>Display an incrementing address range that starts from the selected address (default).</td>
</tr>
<tr>
<td>💡💡💡💍💍</td>
<td>Display a decrementing address range that starts from the selected address.</td>
</tr>
</tbody>
</table>
Using the memory window

The memory window does not show the complete address space of the target, instead you must enter both the address and the number of bytes to display. You can specify the address and size using numeric values or debug expressions which enable you to position the memory display at the address of a variable or at the value of a register. You can also specify whether you want the expressions to be evaluated each time the memory window is updated, or you can re-evaluate them yourself with the press of a button. Memory windows update each time your program stops on a breakpoint, after a single step and whenever you traverse the call stack. If any values that were previously displayed have changed, they are highlighted in red.

To activate the first Memory window:

Choose View > Memory > Memory 1 or press Ctrl+T, M, 1.

Other register windows can be similarly activated.

Using the mouse

You can move the memory window’s edit cursor by clicking on a data or text entry.

The vertical scroll bar can be used to modify the address being viewed by clicking the up and down buttons, the page up and down areas or using the vertical scroll wheel when the scroll bar is at it’s furthest extent. Holding down the Shift key while scrolling will prevent the address being modified.

Using the keyboard

<table>
<thead>
<tr>
<th>Keystroke</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up</td>
<td>Move the cursor up one line, or if the cursor is on the first line, move the address up one line.</td>
</tr>
<tr>
<td>Down</td>
<td>Move the cursor down one line, or if the cursor is on the last line, move the address down line line.</td>
</tr>
<tr>
<td>Left</td>
<td>Move the cursor left one character.</td>
</tr>
<tr>
<td>Right</td>
<td>Move the cursor right one character.</td>
</tr>
<tr>
<td>Home</td>
<td>Move the cursor to the first entry.</td>
</tr>
<tr>
<td>End</td>
<td>Move the cursor to the last entry.</td>
</tr>
</tbody>
</table>
PageUp  | Move the cursor up one page, or if the cursor is on first page, move the address up one page.
---|---
PageDown  | Move the cursor down one page, or if the cursor is on the last page, move the address down one page.
Ctrl+E  | Toggle the cursor between data and text editing.

**Editing memory**

To edit memory, simply move the cursor to the data or text entry you want to modify and start typing. The memory entry will be written and read back as you type.

**Shortcut menu commands**

The shortcut menu contains the following commands:

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Memory By Display Width</td>
<td>Access memory in terms of the display width.</td>
</tr>
<tr>
<td>Address Order</td>
<td>Specify whether the address range shown uses Address as the start or end address and whether addresses should increment or decrement.</td>
</tr>
<tr>
<td>Auto Evaluate</td>
<td>Re-evaluate Address and Size each time the Memory window is updated.</td>
</tr>
<tr>
<td>Auto Refresh</td>
<td>Specify how frequently the memory window should automatically refresh.</td>
</tr>
<tr>
<td>Export To Binary Editor</td>
<td>Create a binary editor with the current Memory window contents.</td>
</tr>
<tr>
<td>Save As</td>
<td>Save the current Memory window contents to a file. Supported file formats are Binary File, Motorola S-Record File, Intel Hex File, TI Hex File, and Hex File.</td>
</tr>
<tr>
<td>Load From</td>
<td>Load the current Memory window from a file. Supported file formats are Binary File, Motorola S-Record File, Intel Hex File, TI Hex File, and Hex File.</td>
</tr>
</tbody>
</table>

**Display formats**

You can set the Memory window to display 8-bit, 16-bit, and 32-bit values that are formatted as hexadecimal, decimal, unsigned decimal, octal, or binary. You can also specify how many columns to display.

**Saving memory contents**

You can save the displayed contents of the memory window to a file in various formats. Alternatively, you can export the contents to a binary editor to work on them.
You can save the displayed memory values as a binary file, Motorola S-record file, Intel hex file, or a Texas Instruments TXT file.

**To save the current state of memory to a file:**

Select the start address and number of bytes to save by editing the **Start Address** and **Size** fields in the Memory window toolbar.

Right-click the main memory display.

From the shortcut menu, select **Save As**, then choose the format from the submenu.

**To export the current state of memory to a binary editor:**

Select the start address and number of bytes to save by editing the **Start Address** and **Size** fields in the Memory window toolbar.

Right-click the main memory display.

Choose **Export to Binary Editor** from the shortcut menu.

Note that subsequent modifications in the binary editor will not modify memory in the target.

**Copying to clipboard**

You can copy the contents of the memory window to the clipboard as text. If an address range is selected, the data or text of the selected range will be copied to the clipboard depending on whether the selection has been made in the data or text view. If no address range is selected, the current memory window view will be copied to the clipboard.
Breakpoints window

The **Breakpoints** window manages the list of currently set breakpoints on the solution. Using the **Breakpoints** window, you can:

- Enable, disable, and delete existing breakpoints.
- Add new breakpoints.
- Show the status of existing breakpoints.

Breakpoints are stored in the session file, so they will be remembered each time you work on a particular project. When running in the debugger, you can set breakpoints on assembly code addresses. These low-level breakpoints appear in the **Breakpoints** window for the duration of the debug run but are not saved when you stop debugging.

When a breakpoint is reached, the matching breakpoint is highlighted in the **Breakpoints** window.

Breakpoints window layout

The **Breakpoints** window has a toolbar and a main breakpoint display.

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Create New" /></td>
<td>Create a new breakpoint using the <strong>New Breakpoint</strong> dialog.</td>
</tr>
<tr>
<td><img src="image" alt="Toggle" /></td>
<td>Toggle the selected breakpoint between enabled and disabled states.</td>
</tr>
<tr>
<td><img src="image" alt="Remove" /></td>
<td>Remove the selected breakpoint.</td>
</tr>
<tr>
<td><img src="image" alt="Move" /></td>
<td>Move the insertion point to the statement where the selected breakpoint is set.</td>
</tr>
<tr>
<td><img src="image" alt="Delete All" /></td>
<td>Delete all breakpoints.</td>
</tr>
<tr>
<td><img src="image" alt="Disable All" /></td>
<td>Disable all breakpoints.</td>
</tr>
<tr>
<td><img src="image" alt="Enable All" /></td>
<td>Enable all breakpoints.</td>
</tr>
<tr>
<td><img src="image" alt="Create Group" /></td>
<td>Create a new breakpoint group and makes it active.</td>
</tr>
</tbody>
</table>

The main part of the **Breakpoints** window shows what breakpoints are set and the state they are in. You can organize breakpoints into folders, called **breakpoint groups**.

SEGGER Embedded Studio displays these icons to the left of each breakpoint:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Icon" /></td>
<td></td>
</tr>
</tbody>
</table>

174
<table>
<thead>
<tr>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Enabled breakpoint</strong></td>
<td>An enabled breakpoint will stop your program running when the breakpoint condition is met.</td>
</tr>
<tr>
<td><strong>Disabled breakpoint</strong></td>
<td>A disabled breakpoint will not stop the program when execution passes through it.</td>
</tr>
<tr>
<td><strong>Invalid breakpoint</strong></td>
<td>An invalid breakpoint is one where the breakpoint cannot be set; for example, no executable code is associated with the source code line where the breakpoint is set or the processor does not have enough hardware breakpoints.</td>
</tr>
</tbody>
</table>

### Showing the Breakpoints window

**To activate the Breakpoints window:**

Choose **Breakpoints > Breakpoints** or press **Ctrl+Alt+B**.

### Managing single breakpoints

You can manage breakpoints in the **Breakpoint** window.

**To delete a breakpoint:**

In the **Breakpoints** window, click the breakpoint to delete.
From the **Breakpoints** window toolbar, click the **Delete Breakpoint** button.

**To edit a breakpoint:**

In the **Breakpoints** window, right-click the breakpoint to edit.
Choose **Edit Breakpoint** from the shortcut menu.
Edit the breakpoint in the **New Breakpoint** dialog.

**To toggle the enabled state of a breakpoint:**

In the **Breakpoints** window, right-click the breakpoint to enable or disable.
Choose **Enable/Disable Breakpoint** from the shortcut menu.

or

In the **Breakpoints** window, click the breakpoint to enable or disable.
Press **Ctrl+F9**.
Breakpoint groups

Breakpoints are divided into breakpoint groups. You can use breakpoint groups to specify sets of breakpoints that are applicable to a particular project in the solution or for a particular debug scenario. Initially, there is a single breakpoint group, named Default, to which all new breakpoints are added.

To create a new breakpoint group:
- From the Breakpoints window toolbar, click the New Breakpoint Group button.
- From the Debug menu, choose Breakpoints then New Breakpoint Group.
- Right-click anywhere in the Breakpoints window.
  Choose New Breakpoint Group from the shortcut menu.

In the New Breakpoint Group dialog, enter the name of the breakpoint group.

When you create a breakpoint, it is added to the active breakpoint group.

To make a group the active group:
- In the Breakpoints window, right-click the breakpoint group to make active.
  Choose Set as Active Group from the shortcut menu.

To delete a breakpoint group:
- In the Breakpoints window, right-click the breakpoint group to delete.
  Choose Delete Breakpoint Group from the shortcut menu.

You can enable all breakpoints within a group at once.

To enable all breakpoints in a group:
- In the Breakpoints window, right-click the breakpoint group to enable.
  Choose Enable Breakpoint Group from the shortcut menu.

You can disable all breakpoints within a group at once.

To disable all breakpoints in a group:
- In the Breakpoints window, right-click the breakpoint group to disable.
  Choose Disable Breakpoint Group from the shortcut menu.

Managing all breakpoints

You can delete, enable, or disable all breakpoints at once.
To delete all breakpoints:

Choose Breakpoints > Clear All Breakpoints or press Ctrl+Shift+F9.

or

On the Breakpoints window toolbar, click the Delete All Breakpoints button.

To enable all breakpoints:

Choose Breakpoints > Enable All Breakpoints or press Ctrl+B, N.

or

On the Breakpoints window toolbar, click the Enable All Breakpoints button.

To disable all breakpoints:

Choose Breakpoints > Disable All Breakpoints or press Ctrl+B, X.

or

On the Breakpoints window toolbar, click the Disable All Breakpoints button.
Call Stack window

The Call Stack window displays the list of function calls (stack frames) that were active when program execution halted. When execution halts, SEGGER Embedded Studio populates the call-stack window from the active (currently executing) task. For simple, single-threaded applications not using the SEGGER Embedded Studio for ARM tasking library, there is only a single task; but for multi-tasking programs that use the SEGGER Embedded Studio for ARM Tasking Library, there may be any number of tasks. SEGGER Embedded Studio updates the Call Stack window when you change the active task in the Threads window.

The Call Stack window has a toolbar and a main call-stack display.

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Move the insertion point to where the call was made to the selected frame.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Set the debugger context to the selected stack frame.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Move the debugger context down one stack to the called function.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Move the debugger context up one stack to the calling function.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Select the fields to display for each entry in the call stack.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Set the debugger context to the most recent stack frame and move the insertion point to the currently executing statement.</td>
</tr>
</tbody>
</table>

The main part of the Call Stack window displays each unfinished function call (active stack frame) at the point when program execution halted. The most recent stack frame is displayed at the bottom of the list and the oldest is displayed at the top of the list.

SEGGER Embedded Studio displays these icons to the left of each function name:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Indicates the stack frame of the current task.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Indicates the stack frame selected for the debugger context.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Indicates that a breakpoint is active and when the function returns to its caller.</td>
</tr>
</tbody>
</table>

These icons can be overlaid to show, for instance, the debugger context and a breakpoint on the same stack frame.
Showing the call-stack window

To activate the Call Stack window:
Choose View > Call Stack or press Ctrl+Alt+S.

Configuring the call-stack window

Each entry in the Call Stack window displays the function name and, additionally, parameter names, types, and values. You can configure the Call Stack window to show varying amounts of information for each stack frame. By default, SEGGER Embedded Studio displays all information.

To show or hide a field:
1. On the Call Stack toolbar, click the Options button on the far right.
2. Select the fields to show, and deselect the ones that should be hidden.

Changing the debugger context

You can select the stack frame for the debugger context from the Call Stack window.

To move the debugger context to a specific stack frame:

In the Call Stack window, double-click the stack frame to move to.

or

In the Call Stack window, select the stack frame to move to.
On the Call Stack window's toolbar, click the Switch To Frame button.

or

In the Call Stack window, right-click the stack frame to move to.
Choose Switch To Frame from the shortcut menu.

The debugger moves the insertion point to the statement where the call was made. If there is no debug information for the statement at the call location, SEGGER Embedded Studio opens a disassembly window at the instruction.

To move the debugger context up one stack frame:

On the Call Stack window's toolbar, click the Up One Stack Frame button.

or
On the Debug Location toolbar, click the Up One Stack Frame button.

or

Press Alt+-.

The debugger moves the insertion point to the statement where the call was made. If there is no debug information for the statement at the call location, SEGGER Embedded Studio opens a disassembly window at the instruction.

To move the debugger context down one stack frame:

On the Call Stack window's toolbar, click the Down One Stack Frame button.

or

On the Debug Location toolbar, click the Down One Stack Frame button.

or

Press Alt++.

The debugger moves the insertion point to the statement where the call was made. If there is no debug information for the statement at the call location, SEGGER Embedded Studio opens a disassembly window at the instruction.

Setting a breakpoint on a return to a function

To set a breakpoint on return to a function:

In the Call Stack window, click the stack frame on the function to stop at on return.
On the Build toolbar, click the Toggle Breakpoint button.

or

In the Call Stack window, click the stack frame on the function to stop at on return.
Press F9.

or

In the Call Stack window, right-click the function to stop at on return.
Choose Toggle Breakpoint from the shortcut menu.
Threads window

The Threads window displays the set of executing contexts on the target processor structured as a set of queues.

To activate the Threads window:

Choose View > More Debug Windows > Threads or press Ctrl+Alt+H.

The window is populated using the threads script, which is a JavaScript program store in a file whose file-type project option is "Threads Script" (or is called threads.js) and is in the project that is being debugged.

When debugging starts, the threads script is loaded and the function init() is called to determine which columns are displayed in the Threads window.

When the application stops on a breakpoint, the function update() is called to create entries in the Threads window corresponding to the columns that have been created together with the saved execution context (register state) of the thread. By double-clicking one of the entries, the debugger displays its saved execution context to put the debugger back into the default execution context, use Show Next Statement.

Writing the threads script

The threads script controls the Threads window with the Threads object.

The methods Threads.setColumns, Threads.setSortByNumber and Threads setColor can be called from the function init().

```javascript
function init() {
  Threads.setColumns("Name", "Priority", "State", "Time");
  Threads.setSortByNumber("Time");
  Threads.setColor("State", "Ready", "Executing", "Waiting");
}
```

The above example creates the named columns Name, Priority, State, and Time in the Threads window, with the Time column sorted numerically rather than alphabetically. The states Ready, Executing and Waiting will have yellow, green and red colored pixmaps respectively.

If you don't supply the function init() in the threads script, the Threads window will create the default columns Name, Priority, and State.

The methods Threads.clear(), Threads.newqueue(), and Threads.add() can be called from the function update().

The Threads.clear() method clears the Threads window.

The Threads.newqueue() function takes a string argument and creates a new, top-level entry in the Threads window. Subsequent entries added to this window will go under this entry. If you don't call this, new entries will all be at the top level of the Threads window.
The `Threads.add()` function takes a variable number of string arguments, which should correspond to the number of columns displayed by the `Threads` window. The last argument to the `Threads.add()` function should be an array (possibly empty) containing the registers of the thread or, alternatively, a handle that can be supplied a call to the threads script `function getregs(handle)`, which will return an array when the thread is selected in the `Threads` window. The array containing the registers should have elements in the same order in which they are displayed in the CPU `Registers` display typically this will be in register-number order, e.g., r0, r1, and so on.

```javascript
function update()
{
    Threads.clear();
    Threads.newqueue("My Tasks");
    Threads.add("Task1", "0", "Executing", "1000", [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]);
    Threads.add("Task2", "1", "Waiting", "2000", [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]);
}
```

The above example will create a fixed output on the `Threads` window and is here to demonstrate how to call the methods.

To get real thread state, you need to access the debugger from the threads script. To do this, you can use the JavaScript method `Debug.evaluate("expression")`, which will evaluate the string argument as a debug expression and return the result. The returned result will be an object if you evaluate an expression that denotes a structure or an array. If the expression denotes a structure, each field can be accessed by using its field name.

So, if you have structs in the application as follows

```c
struct task {
    char *name;
    unsigned char priority;
    char *state;
    unsigned time;
    struct task *next;
    unsigned registers[17];
    unsigned thread_local_storage[4];
};

struct task task2 =
{
    "Task2",
    1,
    "Waiting",
    2000,
    0,
    { 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 },
    { 0,1,2,3 }
};

struct task task1 =
{
    "Task1",
    0,
    "Executing",
    1000,
    & task2,
    { 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 },
};
```
you can update() the Threads window using the following:

task1 = Debug.evaluate("task1");
Threads.add(task1.name, task1.priority, task1.state, task1.time, task1registers);

You can use pointers and C-style cast to enable linked-list traversal.

var next = Debug.evaluate("&task1");
while (next) {
    var xt = Debug.evaluate("*(struct task*)" + next);
    Threads.add(xt.name, xt.priority, xt.state, xt.time, xtregisters);
    next = xt.next;
}

Note that, if the threads script goes into an endless loop, the debugger and consequently SEGGER Embedded Studiowill become unresponsive and you will need to kill SEGGER Embedded Studio using a task manager. Therefore, the above loop is better coded as follows:

var next = Debug.evaluate("&task1");
var count = 0;
while (next && count < 10) {
    var xt = Debug.evaluate("*(struct task*)" + next);
    Threads.add(xt.name, xt.priority, xt.state, xt.time, xtregisters);
    next = xt.next;
    count++;
}

You can speed up the Threads window update by not supplying the registers of the thread to the Threads.add() function. To do this, you should supply a handle/pointer to the thread as the last argument to the Threads.add() function. For example:

var next = Debug.evaluate("&task1");
var count = 0;
while (next && count < 10) {
    var xt = Debug.evaluate("*(struct task*)" + next);
    Threads.add(xt.name, xt.priority, xt.state, xt.time, next);
    next = xt.next;
    count++;
}

When the thread is selected, the Threads window will call getregs(x) in the threads script. That function should return the array of registers, for example:

function getregs(x)
{
    return Debug.evaluate("(*(struct task*)" + x + ")-registers");
}
If you use thread local storage, implementing the `gettls(x)` function enables you to return the base address of the thread local storage, for example:

```c
function gettls(x)
{
    return Debug.evaluate("((struct task*)" + x + "+")\->thread_local_storage");
}
```

The `gettls(x)` function can also be called with null as a parameter. In this case you will have to evaluate an expression that returns the current thread local storage, for example:

```c
function gettls(x)
{
    if (x==null)
        x = Debug.evaluate("&currentTask");
    return Debug.evaluate("((struct task*)" + x + "+")\->thread_local_storage");
}
```

The debugger may require the name of a thread which you can provide by implementing the `getname(x)` function, for example:

```c
function getname(x)
{
    return Debug.evaluate("((struct task*)" + x + "+")\->name");
}
```

Adding extra queues to the threads window

You can add extra information to the threads window to display other RTOS queues. In the function `init()` you can use `Threads.setColumns2` to create an additional display in the threads window, for example:

```c
function init()
{
    ... 
    Threads.setColumns2("Timers", "Id(Timers)", "Name", "Hook", "Timeout", "Period", "Active");
}
```

The first argument is identifier of the queue which is also supplied to `Threads.add2` in the function `update()` as follows

```c
function update()
{
    ... 
    Threads.add2("Timers", "0x1FF0A30", "MyTimer", "0x46C8 (Timer50)", "50(550)", "50", "1");
}
```
Execution Profile window

The Execution Profile window shows a list of source locations and the number of times those source locations have been executed. This window is only available for targets that support the collection of jump trace information.

To activate the Execution Profile window:

Choose View > More Debug Windows > Execution Profile or press Ctrl+T, P.

The count value displayed is the number of times the first instruction of the source code location has been executed. The source locations displayed are target dependent: they could represent each statement of the program or each jump target of the program. If however the debugger is in intermixed or disassembly mode then the count values will be displayed on a per instruction basis.

The execution counts window is updated each time your program stops and the window is visible so if you have this window displayed then single stepping may be slower than usual.
Execution Trace window

The trace window displays historical information on the instructions executed by the target.

To activate the Trace window:

Choose View > More Debug Windows > Execution Trace or press Ctrl+T, T.

The type and number of the trace entries depends upon the target that is connected when gathering trace information. Some targets may trace all instructions, others may trace jump instructions, and some may trace modifications to variables. You'll find the trace capabilities of your target on the shortcut menu.

Each entry in the trace window has a unique number, and the lower the number the earlier the trace. You can click on the header to show earliest to latest or the latest to earliest trace entries. If a trace entry can have source code located to it then double-clicking the trace entry will show the appropriate source display.

Some targets may provide timing information which will be displayed in the ticks column.

The trace window is updated each time the debugger stops when it is visible so single stepping is likely to be slower if you have this window displayed.
Debug file search editor

When a program is built with debugging enabled, the debugging information contains the paths and filenames of all the source files for the program in order to allow the debugger to find them. If a program or library linked into the program is on a different machine than the one on which it was compiled, or if the source files were moved after the program was compiled, the debugger will not be able to find the source files.

In this situation, the simplest way to help SEGGER Embedded Studio find the source files is to add the directory containing the source files to one of its source-file search paths. Alternatively, if SEGGER Embedded Studio cannot find a source file, it will prompt you for its location and will record its new location in the source-file map.

Debug source-file search paths

Debug’s source-file search paths can be used to help the debugger locate source files that are no longer located where they were at compile time. When a source file cannot be found, the search-path directories will be checked, in turn, to see if they contain the source file. SEGGER Embedded Studio maintains two debug source-file search paths:

- **Project-session search path:** This path is for the current project session and does not apply to all projects.
- **The global search path:** This system-wide path applies to all projects.

The project-session search path is checked before the global search path.

**To edit the debug search paths:**

Choose **Debug > Options > Search Paths**.

Debug source file map

If a source file cannot be found while debugging and the debugger has to prompt the user for its location, the results are stored in the debug source file map. The debug source file map simply correlates, or maps, the original pathnames to the new locations. When a file cannot be found at its original location or in the debug search paths, the debug source file map is checked to see if a new location has been recorded for the file or if the user has specified that the file does not exist. Each project session maintains its own source file map, the map is not shared by all projects.

**To view the debug source file map:**

Choose **Debug > Options > Search Paths**.

**To remove individual entries from the debug source file map:**

Choose **Debug > Options > Search Paths**.
Right-click the mapping to delete.
Choose Delete Mapping from the shortcut menu.

To remove all entries from the debug source file map:

Choose Debug > Options > Search Paths.
Right-click any mapping.
Choose Delete All Mappings from the shortcut menu.
Debug Terminal window

The **Debug Terminal** window displays debug output from the target application and can also be used to provide debug input to the target application.

**To activate the Debug Terminal window:**

Choose **View > Debug Terminal** or press **Ctrl+Alt+D**.
Breakpoint expressions

The debugger can set breakpoints by evaluating simple C-like expressions. Note that the exact capabilities offered by the hardware to assist in data breakpointing will vary from target to target; please refer to the particular target interface you are using and the capabilities of your target silicon for exact details. The simplest expression supported is a symbol name. If the symbol name is a function, a breakpoint occurs when the first instruction of the symbol is about to be executed. If the symbol name is a variable, a breakpoint occurs when the symbol has been accessed; this is termed a data breakpoint. For example, the expression \( x \) will breakpoint when \( x \) is accessed. You can use a debug expression (see Debug expressions) as a breakpoint expression. For example, \( x[4] \) will breakpoint when element 4 of array \( x \) is accessed, and \( \text{@sp} \) will breakpoint when the sp register is accessed.

Data breakpoints can be specified, using the \( == \) operator, to occur when a symbol is accessed with a specific value. The expression \( x == 4 \) will breakpoint when \( x \) is accessed and its value is 4. The operators \( <, >=, >, <=, ==, \) and \( != \) can be used similarly. For example, \( \text{@sp} <= 0x1000 \) will breakpoint when register \( sp \) is accessed and its value is less than or equal to 0x1000.

You can use the operator \& to mask the value you wish to break on. For example, \( (x & 1) == 1 \) will breakpoint when \( x \) is accessed and has an odd value.

You can use the operator \&\& to combine comparisons. For example

\[
(x >= 2) \&\& (x <= 14)
\]

will breakpoint when \( x \) is accessed and its value is between 2 and 14.

You can specify an arbitrary memory range using an array cast expression. For example, \( (\text{char}[256])(0x1000) \) will breakpoint when the memory region 0x10000x10FF is accessed.

You can specify an inverse memory range using the \( ! \) operator. For example \( !(\text{char}[256])(0x1000) \) will breakpoint when memory outside the range 0x10000x10FF is accessed.
Debug expressions

The debugger can evaluate simple expressions that can be displayed in the Watch window or as a tool-tip in the code editor.

The simplest expression is an identifier the debugger tries to interpret in the following order:

- an identifier that exists in the scope of the current context.
- the name of a global identifier in the program of the current context.

Numbers can be used in expressions. Hexadecimal numbers must be prefixed with 0x.

Registers can be referenced by prefixing the register name with @.

The standard C and C++ operators !, ~, *, /, %, +, -, >>, <<, <, <, >, >=, ==, |, & & | & ^ && | | are supported on numeric types.

The standard assignment operators =, +=, -=, *=, /=, %=, >>=, >>>=, <<=, &=, |=, ^= are supported on numeric types.

The array subscript operator [] is supported on array and pointer types.

The structure access operator . is supported on structured types (this also works on pointers to structures), and - > works similarly.

The dereference operator (prefix *) is supported on pointers, the address-of (prefix &) and sizeof operators are supported.

The addressof (filename, linenumber) operator will return the address of the specified source code line number.

Function calling with parameters and return results.

Casting to basic pointer types is supported. For example, (unsigned char *)0x300 can be used to display the memory at a given location.

Casting to basic array types is supported. For example, (unsigned char[256])0x100 can be used to reference a memory region.

Operators have the precedence and associativity one expects of a C-like programming language.
Terminal emulator window

The Terminal Emulator window contains a basic serial-terminal emulator that allows you to receive and transmit data over a serial interface.

To activate the Terminal Emulator window:

Choose Tools > Terminal Emulator > Terminal Emulator or press Ctrl+Alt+M.

To use the terminal emulator:

1. Set the required terminal emulator properties.
2. Connect the terminal emulator to the communications port by clicking the button on the toolbar or by selecting Connect from the shortcut menu.

Once connected, any input in the Terminal Emulator window is sent to the communications port and any data received from the communications port is displayed on the terminal.

Connection may be refused if the communication port is in use by another application or if the port doesn’t exist.

To disconnect the terminal emulator:

1. Disconnect the communications port by clicking the Disconnect icon on the toolbar or by right-clicking to select Disconnect from the shortcut menu.

This will release the communications port for use in other applications.

Supported control codes

The terminal supports a limited set of control codes:

<table>
<thead>
<tr>
<th>Control code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;BS&gt;</td>
<td>Backspace</td>
</tr>
<tr>
<td>&lt;CR&gt;</td>
<td>Carriage return</td>
</tr>
<tr>
<td>&lt;LF&gt;</td>
<td>Linefeed</td>
</tr>
<tr>
<td>&lt;ESC&gt;[(attr1);...;(attrn)m</td>
<td>Set display attributes. The attributes 2-Dim, 5-Blink, 7-Reverse, and 8-Hidden are not supported.</td>
</tr>
</tbody>
</table>
Command-line options

This section describes the command-line options accepted by SEGGER Embedded Studio.

Usage

`emStudio [options] [files]`
-D (Define macro)

Syntax

-D macro=value

Description

Define a SEGGER Embedded Studio for ARM macro value.
-noclang (Disable Clang support)

Syntax

-noclang

Description

Disable Clang support.
-packagesdir (Specify packages directory)

Syntax

-packagesdir dir

Description

Override the default value of the $(PackagesDir) macro.
-permit-multiple-studio-instances (Permit multiple studio instances)

Syntax

-permit-multiple-studio-instances

Description

Allow multiple instances of SEGGER Embedded Studio to run at the same time. This behaviour can also be enabled using the Environment > Startup Options > Allow Multiple SEGGER Embedded Studios environment option.
-rootuserdir (Set the root user data directory)

Syntax

-rootuserdir dir

Description

Set the SEGGER Embedded Studio for ARM root user data directory.
-save-settings-off (Disable saving of environment settings)

Syntax

-save-settings-off

Description

Disable the saving of modified environment settings.
-set-setting (Set environment setting)

Syntax

-`set-setting environment_setting=value`

Description

Sets an environment setting to a specified value. For example:

```
-set-setting "Environment/Build/Show Command Lines=Yes"
```
-templatesfile (Set project templates path)

Syntax

-templatesfile path

Description

Sets the search path for finding project template files.
Uninstalling SEGGER Embedded Studio for ARM

This section describes how to completely uninstall SEGGER Embedded Studio for ARM for each supported operating system:

- Uninstalling SEGGER Embedded Studio for ARM from Windows
- Uninstalling SEGGER Embedded Studio for ARM from Mac OS X
- Uninstalling SEGGER Embedded Studio for ARM from Linux

Uninstalling SEGGER Embedded Studio for ARM from Windows

Removing user data and settings

The uninstaller does not remove any user data such as settings or installed packages. To completely remove the user data you will need to carry out the following operations for each user that has used SEGGER Embedded Studio for ARM on your system.

To remove user data using SEGGER Embedded Studio:

1. Start SEGGER Embedded Studio.
2. Click Tools > Admin > Remove All User Data...
Alternatively, if SEGGER Embedded Studio for ARM has already been uninstalled you can manually remove the user data as follows:

1. Click the Windows Start button.
2. Type %LOCALAPPDATA% in the search field and press enter to open the local application data folder.
3. Open the SEGGER folder.
4. Open the SEGGER Embedded Studio for ARM folder.
5. Delete the v4 folder.
6. If you want to delete user data for all versions of the software, delete the SEGGER Embedded Studio for ARM folder as well.

Uninstalling SEGGER Embedded Studio for ARM

To uninstall SEGGER Embedded Studio for ARM:

1. If SEGGER Embedded Studio is running, click File > Exit to shut it down.
2. Click the Start Menu and select Control Panel. The Control Panel window will open.
3. In the Control Panel window, click the Uninstall a program link under the Programs section.
4. From the list of currently installed programs, select SEGGER Embedded Studio for ARM 4.52.
5. To begin the uninstall, click the Uninstall button at the top of the list.

Uninstalling SEGGER Embedded Studio for ARM from Mac OS X

Removing user data and settings

Uninstalling does not remove any user data such as settings or installed packages. To completely remove the user data you will need to carry out the following operations for each user that has used SEGGER Embedded Studio for ARM on your system.

To remove user data using SEGGER Embedded Studio:

1. Start SEGGER Embedded Studio.
2. Click Tools > Admin > Remove All User Data...

Alternatively, if SEGGER Embedded Studio for ARM has already been uninstalled you can manually remove the user data as follows:

1. Open Finder.
2. Go to the $HOME/Library/SEGGER/SEGGER Embedded Studio for ARM directory.
3. Drag the v4 folder to the Trash.
4. If you want to delete user data for all versions of the software, drag the SEGGER Embedded Studio for ARM folder to the Trash as well.

Uninstalling SEGGER Embedded Studio for ARM

To uninstall SEGGER Embedded Studio for ARM:

1. If SEGGER Embedded Studio is running, shut it down.
2. Open the Applications folder in Finder.
3. Drag the SEGGER Embedded Studio for ARM 4.52 folder to the Trash.

Uninstalling SEGGER Embedded Studio for ARM from Linux

Removing user data and settings

The uninstaller does not remove any user data such as settings or installed packages. To completely remove the user data you will need to carry out the following operations for each user that has used SEGGER Embedded Studio for ARM on your system.

To remove user data using SEGGER Embedded Studio:

1. Start SEGGER Embedded Studio.
2. Click Tools > Admin > Remove All User Data...

Alternatively, if SEGGER Embedded Studio for ARM has already been uninstalled you can manually remove the user data as follows:

1. Open a terminal window or file browser.
2. Go to the $HOME/.segger/SEGGER Embedded Studio for ARM directory.
3. Delete the v4 directory.
4. If you want to delete user data for all versions of the software, delete the SEGGER Embedded Studio for ARM directory as well.

Uninstalling SEGGER Embedded Studio for ARM

To uninstall SEGGER Embedded Studio for ARM:

1. If SEGGER Embedded Studio is running, click File > Exit to shut it down.
2. Open a terminal window.
3. Go to the SEGGER Embedded Studio for ARM bin directory (this is /usr/share/segger_embedded_studio_for_arm_4.52/bin by default).
4. Run sudo ./uninstall to start the uninstaller.
ARM target support

When a target-specific executable project is created using the New Project Wizard, the following default files are added to the project:

* Target_Startup.s* The target-specific startup code. See Target startup code.
* cr0.s/thumb_crt0.s* The SEGGER Embedded Studio for ARM standard C runtime. See Startup code.
* Target_MemoryMap.xml* The target-specific memory map file for the board. See Section Placement. Note that, for some targets, a general linker placement file may not be suitable. In these cases, there will be two memory-map files: one for a flash build and one for a RAM build.
* flash_placement.xml* The linker placement file for a flash build.
* sram_placement.xml* The linker placement file for a RAM build.

Initially, shared versions of these files are added to the project. If you want to modify any these shared files, select the file in the Project Explorer and then click the Import option from the shortcut menu. This will copy a writable version of the file into your project directory and change the path in the Project Explorer to that of the local version. You can then make changes to the local file without affecting the shared copy of it.

The following list describes the typical flow of a C program created with SEGGER Embedded Studio’s project templates:

The processor jumps to the reset_handler label in the target-specific startup code, which configures the target (see Target startup code).

When the target is configured, the target-specific startup code jumps to the _start entry point in the C runtime code, which sets up the C runtime environment (see Startup code).

When the C runtime environment has been set up, the C runtime code jumps to the C entry-point function, main.
When the program returns from main, it re-enters the C runtime code, executes the destructors and enters an endless loop.
Target startup code

The following section describes the role of the target-specific startup code.

When you create a new project to produce an executable file using a target-specific project template, a file containing the default startup code for the target will be added to the project. Initially, a shared version of this file will be added to the project; if you want to modify this file, select the file in the Project Explorer and select Import to copy the file to your project directory.

ARM and Cortex-A/Cortex-R startup code

The target startup file typically consists of the exception vector table and the default set of exception handlers.

_vectors This is the exception vector table. It is put into its own .vectors section in order to ensure that it is can be placed at a specific address which is usually 0x00000000 or the start of Flash memory. The vector table contains jump instructions to the particular exception handlers. It is recommended that absolute jump instructions are used ldr pc, =handler_address rather than relative branch instructions b handler_address since many devices shadow the memory at address zero to start execution but the program will be linked to run at a different address.

reset_handler The reset handler will usually carry out any target-specific initialization and then will jump to the _start entry point. In a C system, the _start entry point is in the crt0.s file. During development it is usual to replace the reset handler with an endless loop which will stop the device running potentially dangerous in-development code directly out of reset. In development the debugger will start the device from the specified debug entry point.

undef_handler This is the default, undefined-instruction exception handler.*

swi_handler This is the default, software-interrupt exception handler.*

pabort_handler This is the default, prefetch-abort exception handler.*

dabort_handler This is the default, data-abort exception handler.*

irq_handler This is the default, IRQ-exception handler.*

fiq_handler This is the default, FIQ-exception handler.*

* Declared as a weak symbol to allow the user to override the implementation.

Note that ARM and Cortex-A/Cortex-R exception handlers must be written in ARM assembly code. The CPU or board support package of the project you have created will typically supply an ARM assembly-coded irq_handler implementation that will enable you to write interrupt service routines as C functions.

Cortex-M startup code

The target startup file typically consists of the exception vector table and the default set of exception handlers.
_vectors  This is the exception vector table. It is put into its own .vectors section in order to ensure that it can be placed at a specific address which is usually 0x00000000 or the start of Flash memory.

The vector table is structured as follows:

The first entry is the initial value of the stack pointer.
The second entry is the address of the reset handler function. The reset handler will usually carry out any target-specific initialization and then jump to the _start entry point. In a C system, the _start entry point is in the thumb_crt0.s file. During development it is usual to replace this jump with an endless loop which will stop the device running potentially dangerous in-development code directly out of reset. In development the debugger will start the device from the specified debug entry point.
The following 15 entries are the addresses of the standard Cortex-M exception handlers ending with the SysTick_ISR entry.
Subsequent entries are addresses of device-specific interrupt sources and their associated handlers.

For each exception handler, a weak symbol is declared that will implement an endless loop. You can implement your own exception handler as a regular C function. Note that the name of the C function must match the name in the startup code e.g. void SysTick_ISR(void). You can use the C preprocessor to rename the symbol in the startup code if you have existing code with different exception handler names e.g. SysTick_ISR=SysTick_Handler.
Startup code

The following section describes the role of the C runtime-startup code, `crt0.s` (and the Cortex-M/Thumb equivalent `thumb_crt0.s`).

When you create a new project to produce an executable file using a target-specific project template, the `crt0.s/thumb_crt0.s` file is added to the project. Initially, a shared version of this file is added to the project. If you want to modify this file, right-click it in the Project Explorer and then select Import from the shortcut menu to copy the file to your project directory.

The entry point of the C runtime-startup code is `_start`. In a typical system, this will be called by the target-specific startup code after it has initialized the target.

The C runtime carries out the following actions:

- Initialize the stacks.
- If required, copy the contents of the `.data` (initialized data) section from non-volatile memory.
- If required, copy the contents of the `.fast` section from non-volatile memory to SRAM.
- Initialize the `.bss` section to zero.
- Initialize the heap.
- Call constructors.
- If compiled with FULL_LIBRARY, get the command line from the host using `debug_getargs` and set registers to supply `argc` and `argv` to `main`.
- Call the `main` entry point.

On return from `main` or when `exit` is called

- If compiled with FULL_LIBRARY, call destructors.
- If compiled with FULL_LIBRARY, call `atexit` functions.
- If compiled with FULL_LIBRARY, call `debug_exit` while supplying the return result from `main`.
- Wait in exit loop.

Program sections

The following program sections are used for the C runtime in section-placement files:

<table>
<thead>
<tr>
<th>Section name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.vectors</code></td>
<td>The exception vector table.</td>
</tr>
<tr>
<td><code>.init</code></td>
<td>Startup code that runs before the call to the application's <code>main</code> function.</td>
</tr>
<tr>
<td><code>.ctors</code></td>
<td>Static constructor function table.</td>
</tr>
<tr>
<td><code>.dtors</code></td>
<td>Static destructor function table.</td>
</tr>
<tr>
<td><code>.text</code></td>
<td>The program code.</td>
</tr>
</tbody>
</table>
```
<table>
<thead>
<tr>
<th>Section name</th>
<th>Linker size symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.stack</td>
<td><strong>STACKSIZE</strong></td>
<td>System and User mode stack.</td>
</tr>
<tr>
<td>.stack_svc</td>
<td><strong>STACKSIZE_SVC</strong></td>
<td>Supervisor mode stack</td>
</tr>
<tr>
<td>.stack_irq</td>
<td><strong>STACKSIZE_IRQ</strong></td>
<td>IRQ mode stack</td>
</tr>
<tr>
<td>.stack_fiq</td>
<td><strong>STACKSIZE_FIQ</strong></td>
<td>FIQ mode stack</td>
</tr>
<tr>
<td>.stack_abt</td>
<td><strong>STACKSIZE_ABT</strong></td>
<td>Abort mode stack</td>
</tr>
<tr>
<td>.stack_und</td>
<td><strong>STACKSIZE_UND</strong></td>
<td>Undefined mode stack</td>
</tr>
</tbody>
</table>
```

Cortex-M devices have the following stacks and linker symbol stack sizes are defined:

```
<table>
<thead>
<tr>
<th>Section name</th>
<th>Linker size symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.stack</td>
<td><strong>STACKSIZE</strong></td>
<td>Main stack.</td>
</tr>
<tr>
<td>.stack_process</td>
<td><strong>STACKSIZE_PROCESS</strong></td>
<td>Process stack.</td>
</tr>
</tbody>
</table>
```

The `crt0.s/thumb_crt0.s` startup code references these sections and initializes each of the stack-pointer registers to point to the appropriate location. To change the location in memory of a particular stack, the section should be moved to the required position in the section-placement or memory-map file.

Should your application not require one or more of these stacks, you can remove those sections from the memory-map file or set the size to 0 and remove the initialization code from the `crt0.s/thumb_crt0.s` file.

### The `.data` section

The `.data` section contains the initialized data. If the run address is different from the load address, as it would be in a flash-based application in order to allow the program to run from reset, the `crt0.s/thumb_crt0.s` startup code will copy the `.data` section from the load address to the run address before calling the `main` entry point.
The .fast section

For performance reasons, it is a common requirement for embedded systems to run critical code from fast memory; the .fast section can be used to simplify this. If the .fast section’s run address is different from the load address, the crt0.s/thumb_crt0.s startup code will copy the .fast section from the load address to the run address before calling the main entry point.

The .bss Section

The .bss section contains the zero-initialized data. The startup code in crt0.s/thumb_crt0.s references the .bss section and sets its contents to zero.

The heap

The position and size of the heap is specified in the project’s section-placement or memory-map file by the .heap program section.

The startup code in crt0.s/thumb_crt0.s references this section and initializes the heap. To change the position of the heap, the section should be moved to the required position in the section-placement or memory-map file.

There is a Heap Size linker project option you can modify in order to alter the heap size. For compatibility with earlier versions of SEGGER Embedded Studio, you can also specify the heap size using the heap section’s Size attribute in the section-placement or memory-map file.

Should your application not require the heap functions, you can remove the heap section from the memory-map file or set the size to zero and remove the heap-initialization code from the crt0.s/thumb_crt0.s file.
Section Placement

Section placement files map program sections used in your program into the memory spaces defined in the memory map or in the Memory Segments project option. For instance, it's common for code and read-only data to be programmed into non-volatile flash memory, whereas read-write data needs to be mapped onto either internal or external RAM.

Memory map files are provided in the CPU support package you are using and are referenced in executable projects by the Memory Map File project option. Section-placement files are provided in the base SEGGER Embedded Studio for ARM distribution.

The memory segments defined in the section placement files have macro-expandable names which can be defined using the Section Placement Macros project option.

Some of the section placement files have a macro-expandable start attribute in the first program section. You can use this to reserve space at the beginning of the memory segment.

ARM section placement

The following placement files are supplied for ARM targets:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flash_placement.xml</td>
<td>Single FLASH segment with internal RAM segment and optional external RAM segment.</td>
</tr>
<tr>
<td>flash_run_text_from_ram_placement.xml</td>
<td>Single FLASH segment with internal RAM segment and optional external RAM segments. Text section is copied from FLASH to RAM.</td>
</tr>
<tr>
<td>internal_sram_placement.xml</td>
<td>Single internal RAM segment.</td>
</tr>
<tr>
<td>multi_flash_placement.xml</td>
<td>Two FLASH segments with internal RAM segment and optional external RAM segment.</td>
</tr>
<tr>
<td>sram_placement.xml</td>
<td>Internal RAM segment and optional external RAM segment.</td>
</tr>
<tr>
<td>tcm_placement.xml</td>
<td>Data and Instruction tightly coupled memory segments.</td>
</tr>
</tbody>
</table>

Cortex-M section placement

The following placement files are supplied for Cortex-M targets:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flash_placement.xml</td>
<td>Two FLASH segments and two RAM segments.</td>
</tr>
<tr>
<td>flash_placement_tcm.xml</td>
<td>One FLASH segments, two RAM segments, Data and Instruction tightly coupled memory segments.</td>
</tr>
<tr>
<td>File Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>flash_placement2.xml</td>
<td>One FLASH segment and two RAM segments.</td>
</tr>
<tr>
<td>flash_to_ram_placement.xml</td>
<td>One FLASH segment and one RAM segment. Text section is copied from FLASH to RAM.</td>
</tr>
<tr>
<td>flash_to_ram_placement_tcm.xml</td>
<td>One FLASH segment, two RAM segments, Data and Instruction tightly coupled memory segments. Text section is copied from FLASH to RAM.</td>
</tr>
<tr>
<td>flash_to_ram_placement2.xml</td>
<td>One FLASH segment and two RAM segments. Text section is copied from FLASH to RAM.</td>
</tr>
<tr>
<td>flash_to_tcm_placement.xml</td>
<td>Two FLASH segments, two RAM segments, Data and Instruction tightly coupled memory segments.</td>
</tr>
<tr>
<td>ram_placement.xml</td>
<td>Two RAM segments.</td>
</tr>
<tr>
<td>tcm_placement.xml</td>
<td>Data and Instruction tightly coupled memory segments.</td>
</tr>
</tbody>
</table>
Using the SEGGER Assembler

You can use the SEGGER assembler by setting the project option `Assembler` to `SEGGER`. 
Using the SEGGER Linker

You can create a new project that builds using the SEGGER linker by choosing the project template

A C executable for a Cortex-M processor

from the New Project wizard. Alternatively you can modify an existing project as follows:

Set the project option **Linker** to **SEGGER**.
Set the project option **Linker Script File** to `$(StudioDir)/samples/SEGGER_Flash.icf`.
Set the project option **Supply Memory Segments To Linker** to **Yes**.
Set the project option **Exclude From Build** on the `thumb_crt0.s` file in your project.
Add the startup file `$(StudioDir)/samples/SEGGER_THUMB_Startup.s` to your project.
C Library User Guide

This section describes the library and how to use and customize it.

The libraries supplied with SEGGER Embedded Studio for ARM have all the support necessary for input and output using the standard C functions `printf` and `scanf`, support for the `assert` function, both 32-bit and 64-bit floating point, and are capable of being used in a multi-threaded environment. However, to use these facilities effectively you will need to customize the low-level details of how to input and output characters, what to do when an assertion fails, how to provide protection in a multithreaded environment, and how to use the available hardware to the best of its ability.
Floating point

The SEGGER Embedded Studio for ARM C library uses IEEE floating point format as specified by the ISO 60559 standard with restrictions.

This library favors code size and execution speed above absolute precision. It is suitable for applications that need to run quickly and not consume precious resources in limited environments. The library does not implement features rarely used by simple applications: floating point exceptions, rounding modes, and subnormals.

NaNs and infinities are supported and correctly generated. The only rounding mode supported is round-to-nearest. Subnormals are always flushed to a correctly-signed zero. The mathematical functions use stable approximations and do their best to cater ill-conditioned inputs.
Multithreading

The SEGGER Embedded Studio for ARM libraries support multithreading, for example, where you are using CTL or a third-party real-time operating system (RTOS).

Where you have single-threaded processes, there is a single flow of control. However, in multithreaded applications there may be several flows of control which access the same functions, or the same resources, concurrently. To protect the integrity of resources, any code you write for multithreaded applications must be reentrant and thread-safe.

Reentrancy and thread safety are both related to the way functions in a multithreaded application handle resources.

Reentrant functions

A reentrant function does not hold static data over successive calls and does not return a pointer to static data. For this type of function, the caller provides all the data that the function requires, such as pointers to any workspace. This means that multiple concurrent calls to the function do not interfere with each other, that the function can be called in mainline code, and that the function can be called from an interrupt service routine.

Thread-safe functions

A thread-safe function protects shared resources from concurrent access using locks. In C, local variables are held in processor registers or are on the stack. Any function that does not use static data, or other shared resources, is thread-safe. In general, thread-safe functions are safe to call from any thread but cannot be called directly, or indirectly, from an interrupt service routine.
Thread safety in the SEGGER Embedded Studio for ARM library

In the SEGGER Embedded Studio for ARM C library:

- some functions are inherently thread-safe, for example `strcmp`.
- some functions, such as `malloc`, are not thread-safe by default but can be made thread-safe by implementing appropriate lock functions.
- other functions are only thread-safe if passed appropriate arguments, for example `tmpnam`.
- some functions are never thread-safe, for example `setlocale`.

We define how the functions in the C library can be made thread-safe if needed. If you use a third-party library in a multi-threaded system and combine it with the SEGGER Embedded Studio for ARM C library, you will need to ensure that the third-party library can be made thread-safe in just the same way that the SEGGER Embedded Studio for ARM C library can be made thread-safe.
Implementing mutual exclusion in the C library

The SEGGER Embedded Studio for ARM C library ships as standard with callouts to functions that provide thread-safety in a multithreaded application. If your application has a single thread of execution, the default implementation of these functions does nothing and your application will run without modification.

If your application is intended for a multithreaded environment and you wish to use the SEGGER Embedded Studio for ARM C library, you must implement the following locking functions:

- *__heap_lock* and *__heap_unlock* to provide thread-safety for all heap operations such as *malloc*, *free*, and *realloc*.
- *__printf_lock* and *__printf_unlock* to provide thread-safety for *printf* and relatives.
- *__scanf_lock* and *__scanf_unlock* to provide thread-safety for *scanf* and relatives.
- *__debug_io_lock* and *__debug_io_unlock* to provide thread-safety for semi-hosting support in the SEGGER Embedded Studio I/O function.

If you use a third-party RTOS with the SEGGER Embedded Studio for ARM C library, you will need to use whatever your RTOS provides for mutual exclusion, typically a semaphore, a mutex, or an event set.
Input and output

The C library provides all the standard C functions for input and output except for the essential items of where to output characters printed to `stdout` and where to read characters from `stdin`.

If you want to output to a UART, to an LCD, or input from a keyboard using the standard library print and scan functions, you need to customize the low-level input and output functions.
Customizing putchar

To use the standard output functions putchar, puts, and printf, you need to customize the way that characters are written to the standard output device. These output functions rely on a function __putchar that outputs a character and returns an indication of whether it was successfully written.

The prototype for __putchar is

```c
int __putchar(int ch);
```

Sending all output to the SEGGER Embedded Studio virtual terminal

You can send all output to the SEGGER Embedded Studio virtual terminal by supplying the following implementation of the __putchar function in your code:

```c
#include <debugio.h>
int __putchar(int ch)
{
    return debug_putchar(ch);
}
```

This hands off output of the character ch to the low-level debug output routine, debug_putchar.

Whilst this is an adequate implementation of __putchar, it does consume stack space for an unnecessary nested call and associated register saving. A better way of achieving the same result is to define the low-level symbol for __putchar to be equivalent to the low-level symbol for debug_putchar. To do this, we need to instruct the linker to make the symbols equivalent.

Select the project node in the Project Explorer.
Display the Properties Window.
Enter the text __putchar=debug_putchar into the Linker > Linker Symbol Definitions property of the Linker Options group.

Sending all output to another device

If you need to output to a physical device, such as a UART, the following notes will help you:

- If the character cannot be written for any reason, putchar must return EOF. Just because a character can't be written immediately is not a reason to return EOF: you can busy-wait or tasking (if applicable) to wait until the character is ready to be written.
- The higher layers of the library do not translate C's end of line character \n before passing it to putchar.
- If you are directing output to a serial line connected to a terminal, for instance, you will most likely need to output a carriage return and line feed when given the character \n (ASCII code 10).
The standard functions that perform input and output are the `printf` and `scanf` functions. These functions convert between internal binary and external printable data. In some cases, though, you need to read and write formatted data on other channels, such as other RS232 ports. This section shows how you can extend the I/O library to best implement these functions.

**Classic custom printf-style output**

Assume that we need to output formatted data to two UARTs, numbered 0 and 1, and we have a functions `uart0_putchar` and `uart1_putchar` that do just that and whose prototypes are:

```c
int uart0_putchar(int ch, __printf_t *ctx);
int uart1_putchar(int ch, __printf_t *ctx);
```

These functions return a positive value if there is no error outputting the character and EOF if there was an error. The second parameter, `ctx`, is the context that the high-level formatting routines use to implement the C standard library functions.

Using a classic implementation, you would use `sprintf` to format the string for output and then output it:

```c
void uart0_printf(const char *fmt, ...) {
    char buf[80], *p;
    va_list ap;
    va_start(ap, fmt);
    vsnprintf(buf, sizeof(buf), fmt, ap);
    for (p = buf; *p; ++p)
        uart0_putchar(*p, 0); // null context
    va_end(ap);
}
```

We would, of course, need an identical routine for outputting to the other UART. This code is portable, but it requires an intermediate buffer of 80 characters. On small systems, this is quite an overhead, so we could reduce the buffer size to compensate. Of course, the trouble with that means that the maximum number of characters that can be output by a single call to `uart0_printf` is also reduced. What would be good is a way to output characters to one of the UARTs without requiring an intermediate buffer.

**SEGGER Embedded Studio for ARM printf-style output**

SEGGER Embedded Studio for ARM provides a solution for just this case by using some internal functions and data types in the SEGGER Embedded Studio for ARM library. These functions and types are define in the header file `<__vfprintf.h>`.

The first thing to introduce is the `__printf_tag` type which captures the current state and parameters of the format conversion:

```c
typedef struct __printf_tag {
    size_t charcount;
    size_t maxchars;
    char *string;
} __printf_t;
```
int (*output_fn)(int, struct __printf_tag *ctx);
} __printf_t;

This type is used by the library functions to direct what the formatting routines do with each character they need to output. If string is non-zero, the character is appended is appended to the string pointed to by string; if output_fn is non-zero, the character is output through the function output_fn with the context passed as the second parameter.

The member charcount counts the number of characters currently output, and maxchars defines the maximum number of characters output by the formatting routine __vfprintf.

We can use this type and function to rewrite uart0_printf:

```c
int uart0_printf(const char *fmt, ...) {
    int n;
    va_list ap;
    __printf_t iod;
    va_start(ap, fmt);
    iod.string = 0;
    iod.maxchars = INT_MAX;
    iod.output_fn = uart0_putchar;
    n = __vfprintf(&iod, fmt, ap);
    va_end(ap);
    return n;
}
```

This function has no intermediate buffer: when a character is ready to be output by the formatting routine, it calls the output_fn function in the descriptor iod to output it immediately. The maximum number of characters isn't limited as the maxchars member is set to INT_MAX. If you wanted to limit the number of characters output you can simply set the maxchars member to the appropriate value before calling __vfprintf.

We can adapt this function to take a UART number as a parameter:

```c
int uart_printf(int uart, const char *fmt, ...) {
    int n;
    va_list ap;
    __printf_t iod;
    va_start(ap, fmt);
    iod.is_string = 0;
    iod.maxchars = INT_MAX;
    iod.output_fn = uart ? uart1_putchar : uart0_putchar;
    n = __vfprintf(&iod, fmt, ap);
    va_end(ap);
    return n;
}
```

Now we can use:

```c
uart_printf(0, "This is uart %d\n...", 0);
uart_printf(1, "..and this is uart %d\n", 1);
```

__vfprintf returns the actual number of characters printed, which you may wish to dispense with and make the uart_printf routine return void.
Extending input functions

The formatted input functions would be implemented in the same manner as the output functions: read a string into an intermediate buffer and parse using `sscanf`. However, we can use the low-level routines in the SEGGER Embedded Studio for ARM library for formatted input without requiring the intermediate buffer.

The type `__stream_scanf_t` is:

```c
typedef struct {
    char is_string;
    int (* getc_fn)(void);
    int (* ungetc_fn)(int);
} __stream_scanf_t;
```

The function `getc_fn` reads a single character from the UART, and `ungetc_fn` pushes back a character to the UART. You can push at most one character back onto the stream.

Here's an implementation of functions to read and write from a single UART:

```c
static int uart0_ungot = EOF;

int uart0_getc(void)
{
    if (uart0_ungot)
    {
        int c = uart0_ungot;
        uart0_ungot = EOF;
        return c;
    }
    else
        return read_char_from_uart(0);
}

int uart0_ungetc(int c)
{
    uart0_ungot = c;
}
```

You can use these two functions to perform formatted input using the UART:

```c
int uart0_scanf(const char *fmt, ...)
{
    __stream_scanf_t iod;
    va_list a;
    int n;
    va_start(a, fmt);
    iod.is_string = 0;
    iod.getc_fn = uart0_getc;
    iod.ungetc_fn = uart0_ungetc;
    n = __vfscanf((__scanf_t *)&iod, (const unsigned char *)fmt, a);
    va_end(a);
    return n;
}
```

Using this template, we can add functions to do additional formatted input from other UARTs or devices, just as we did for formatted output.
Locales

The SEGGER Embedded Studio for ARM C library supports wide characters, multi-byte characters and locales. However, as not all programs require full localization, you can tailor the exact support provided by the SEGGER Embedded Studio for ARM C library to suit your application. These sections describe how to add new locales to your application and customize the runtime footprint of the C library.
Unicode, ISO 10646, and wide characters

The ISO standard 10646 is identical to the published Unicode standard and the SEGGER Embedded Studio for ARM C library uses the Unicode 6.2 definition as a base. Hence, whenever you see the term Unicode in this document, it is equivalent to Unicode 6.2 and ISO/IEC 10646:2011.

The SEGGER Embedded Studio for ARM C library supports both 16-bit and 32-bit wide characters, depending upon the setting of wide character width in the project.

When compiling with 16-bit wide characters, all characters in the Basic Multilingual Plane are representable in a single `wchar_t` (values 0 through 0xFFFF). When compiling with 32-bit wide characters, all characters in the Basic Multilingual Plane and planes 1 through 16 are representable in a single `wchar_t` (values 0 through 0x10FFFF).

The wide character type will hold Unicode code points in a locale that is defined to use Unicode and character type functions such as `iswalpha` will work correctly on all Unicode code points.
Multi-byte characters

SEGGER Embedded Studio for ARM supports multi-byte encoding and decoding of characters. Most new software on the desktop uses Unicode internally and UTF-8 as the external, on-disk encoding for files and for transport over 8-bit mediums such as network connections.

However, in embedded software there is still a case to use code pages, such as ISO-Latin1, to reduce the footprint of an application whilst also providing extra characters that do not form part of the ASCII character set.

The SEGGER Embedded Studio for ARM C library can support both models and you can choose a combination of models, dependent upon locale, or construct a custom locale.
The standard C and POSIX locales

The standard C locale is called simply C. In order to provide POSIX compatibility, the name POSIX is a synonym for C.

The C locale is fixed and supports only the ASCII character set with character codes 0 through 127. There is no multi-byte character support, so the character encoding between wide and narrow characters is simply one-to-one: a narrow character is converted to a wide character by zero extension. Thus, ASCII encoding of narrow characters is compatible with the ISO 10646 (Unicode) encoding of wide characters in this locale.
Additional locales in source form

The SEGGER Embedded Studio for ARM C library provides only the C locale; if you need other locales, you must provide those by linking them into your application. We have constructed a number of locales from the Unicode Common Locale Data Repository (CLDR) and provided them in source form in the $(StudioDir)/source folder for you to include in your application.

A C library locale is divided into two parts:

- the locale's date, time, numeric, and monetary formatting information
- how to convert between multi-byte characters and wide characters by the functions in the C library.

The first, the locale data, is independent of how characters are represented. The second, the code set in use, defines how to map between narrow, multi-byte, and wide characters.
Installing a locale

If the locale you request using `setlocale` is neither C nor POSIX, the C library calls the function `__user_find_locale` to find a user-supplied locale. The standard implementation of this function is to return a null pointer which indicates that no additional locales are installed and, hence, no locale matches the request.

The prototype for `__user_find_locale` is:

```c
const __RAL_locale_t *__user_find_locale(const char *locale);
```

The parameter `locale` is the locale to find; the locale name is terminated either by a zero character or by a semicolon. The locale name, up to the semicolon or zero, is identical to the name passed to `setlocale` when you select a locale.

Now let’s install the Hungarian locale using both UTF-8 and ISO 8859-2 encodings. The UTF-8 codecs are included in the SEGGER Embedded Studio for ARM C library, but the Hungarian locale and the ISO 8859-2 codec are not.

You will find the file `locale_hu_HU.c` in the source directory as described in the previous section. Add this file to your project.

Although this adds the data needed for the locale, it does not make the locale available for the C library: we need to write some code for `__user_find_locale` to return the appropriate locales.

To create the locales, we need to add the following code and data to tie everything together:

```c
#include <__crossworks.h>

static const __RAL_locale_t hu_HU_utf8 = {
   "hu_HU.utf8",
   &__RAL_hu_HU_locale,
   &__RAL_codeset_utf8
};

static const __RAL_locale_t hu_HU_iso_8859_2 = {
   "hu_HU.iso_8859_2",
   &__RAL_hu_HU_locale,
   &codeset_iso_8859_2
};

const __RAL_locale_t *__user_find_locale(const char *locale)
{
    if (__RAL_compare_locale_name(locale, hu_HU_utf8.name) == 0)
        return &hu_HU_utf8;
    else if (__RAL_compare_locale_name(locale, hu_HU_iso_8859_2.name) == 0)
        return &hu_HU_iso_8859_2;
    else
        return 0;
}
```
The function `__RAL_compareLocaleName` matches locale names up to a terminating null character, or a semicolon (which is required by the implementation of `setlocale` in the C library when setting multiple locales using `LC_ALL`).

In addition to this, you must provide a buffer, `__userLocaleNameBuffer`, for locale names encoded by `setlocale`. The buffer must be large enough to contain five locale names, one for each category. In the above example, the longest locale name is `hu_HU.iso_8859_2` which is 16 characters in length. Using this information, buffer must be at least \((16+1)5 = 85\) characters in size:

```
const char __userLocaleNameBuffer[85];
```
Setting a locale directly

Although we support `setlocale` in its full generality, most likely you’ll want to set a locale once and forget about it. You can do that by including the locale in your application and writing to the instance variables that hold the underlying locale data for the SEGGER Embedded Studio for ARM C library.

For instance, you might wish to use Czech locale with a UTF codeset:

```c
static __RAL_locale_t cz_locale = {
    "cz_CZ.utf8",
    __RAL_cs_CZ_locale,
    __RAL_codeset_utf8
};
```

You can install this directly into the locale without using `setlocale`:

```c
__RAL_global_locale.__category[LC_COLLATE] = &cz_locale;
__RAL_global_locale.__category[LC_CTYPE]     = &cz_locale;
__RAL_global_locale.__category[LC_MONETARY]  = &cz_locale;
__RAL_global_locale.__category[LC_NUMERIC]   = &cz_locale;
__RAL_global_locale.__category[LC_TIME]      = &cz_locale;
```
# Complete API reference

This section contains a complete reference to the SEGGER Embedded Studio for ARM C library API.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;assert.h&gt;</code></td>
<td>Describes the diagnostic facilities which you can build into your application.</td>
</tr>
<tr>
<td><code>&lt;debugio.h&gt;</code></td>
<td>Describes the virtual console services and semi-hosting support that SEGGER Embedded Studio provides to help you when developing your applications.</td>
</tr>
<tr>
<td><code>&lt;ctype.h&gt;</code></td>
<td>Describes the character classification and manipulation functions.</td>
</tr>
<tr>
<td><code>&lt;errno.h&gt;</code></td>
<td>Describes the macros and error values returned by the C library.</td>
</tr>
<tr>
<td><code>&lt;float.h&gt;</code></td>
<td>Defines macros that expand to various limits and parameters of the standard floating point types.</td>
</tr>
<tr>
<td><code>&lt;limits.h&gt;</code></td>
<td>Describes the macros that define the extreme values of underlying C types.</td>
</tr>
<tr>
<td><code>&lt;locale.h&gt;</code></td>
<td>Describes support for localization specific settings.</td>
</tr>
<tr>
<td><code>&lt;math.h&gt;</code></td>
<td>Describes the mathematical functions provided by the C library.</td>
</tr>
<tr>
<td><code>&lt;setjmp.h&gt;</code></td>
<td>Describes the non-local goto capabilities of the C library.</td>
</tr>
<tr>
<td><code>&lt;stdarg.h&gt;</code></td>
<td>Describes the way in which variable parameter lists are accessed.</td>
</tr>
<tr>
<td>Header</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>&lt;stdio.h&gt;</code></td>
<td>Describes the formatted input and output functions.</td>
</tr>
<tr>
<td><code>&lt;stdlib.h&gt;</code></td>
<td>Describes the general utility functions provided by the C library.</td>
</tr>
<tr>
<td><code>&lt;string.h&gt;</code></td>
<td>Describes the string handling functions provided by the C library.</td>
</tr>
<tr>
<td><code>&lt;time.h&gt;</code></td>
<td>Describes the functions to get and manipulate date and time information provided by the C library.</td>
</tr>
<tr>
<td><code>&lt;wchar.h&gt;</code></td>
<td>Describes the facilities you can use to manipulate wide characters.</td>
</tr>
</tbody>
</table>
<assert.h>

API Summary

<table>
<thead>
<tr>
<th>Macros</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>assert</td>
<td>__assert</td>
</tr>
<tr>
<td></td>
<td><strong>assert</strong></td>
</tr>
<tr>
<td></td>
<td>Allows you to place assertions and diagnostic tests into programs</td>
</tr>
<tr>
<td></td>
<td>User defined behaviour for the assert macro</td>
</tr>
</tbody>
</table>
__assert

Synopsis

```
void __assert(const char *expression,
              const char *filename,
              int line);
```

Description

There is no default implementation of __assert. Keeping __assert out of the library means that you can customize its behaviour without rebuilding the library. You must implement this function where `expression` is the stringized expression, `filename` is the filename of the source file and `line` is the linenumber of the failed assertion.
assert

Synopsis

```c
#define assert(e) ...
```

Description

If `NDEBUG` is defined as a macro name at the point in the source file where `<assert.h>` is included, the `assert` macro is defined as:

```c
#define assert(ignore) ((void)0)
```

If `NDEBUG` is not defined as a macro name at the point in the source file where `<assert.h>` is included, the `assert` macro expands to a `void` expression that calls `__assert`.

```c
#define assert(e) ((e) ? (void)0 : __assert(#e, __FILE__, __LINE__))
```

When such an `assert` is executed and `e` is false, `assert` calls the `__assert` function with information about the particular call that failed: the text of the argument, the name of the source file, and the source line number. These are the stringized expression and the values of the preprocessing macros `__FILE__` and `__LINE__`.

Note

The `assert` macro is redefined according to the current state of `NDEBUG` each time that `<assert.h>` is included.
### <complex.h>

#### API Summary

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Trigonometric functions</strong></td>
<td></td>
</tr>
<tr>
<td>cacos</td>
<td>Compute inverse cosine of a complex float</td>
</tr>
<tr>
<td>cacosf</td>
<td>Compute inverse cosine of a complex float</td>
</tr>
<tr>
<td>casin</td>
<td>Compute inverse sine of a complex float</td>
</tr>
<tr>
<td>casinf</td>
<td>Compute inverse sine of a complex float</td>
</tr>
<tr>
<td>catan</td>
<td>Compute inverse tangent of a complex float</td>
</tr>
<tr>
<td>catanf</td>
<td>Compute inverse tangent of a complex float</td>
</tr>
<tr>
<td>ccos</td>
<td>Compute cosine of a complex float</td>
</tr>
<tr>
<td>ccosf</td>
<td>Compute cosine of a complex float</td>
</tr>
<tr>
<td>csin</td>
<td>Compute sine of a complex float</td>
</tr>
<tr>
<td>csinf</td>
<td>Compute sine of a complex float</td>
</tr>
<tr>
<td>ctan</td>
<td>Compute tangent of a complex float</td>
</tr>
<tr>
<td>ctanf</td>
<td>Compute tangent of a complex float</td>
</tr>
<tr>
<td><strong>Hyperbolic trigonometric functions</strong></td>
<td></td>
</tr>
<tr>
<td>cacosh</td>
<td>Compute inverse hyperbolic cosine of a complex float</td>
</tr>
<tr>
<td>cacoshf</td>
<td>Compute inverse hyperbolic cosine of a complex float</td>
</tr>
<tr>
<td>casinh</td>
<td>Compute inverse hyperbolic sine of a complex float</td>
</tr>
<tr>
<td>casinhf</td>
<td>Compute inverse hyperbolic sine of a complex float</td>
</tr>
<tr>
<td>catanh</td>
<td>Compute inverse hyperbolic tangent of a complex float</td>
</tr>
<tr>
<td>catanhf</td>
<td>Compute inverse hyperbolic tangent of a complex float</td>
</tr>
<tr>
<td>ccosh</td>
<td>Compute hyperbolic cosine of a complex float</td>
</tr>
<tr>
<td>ccoshf</td>
<td>Compute hyperbolic cosine of a complex float</td>
</tr>
<tr>
<td>csinh</td>
<td>Compute hyperbolic sine of a complex float</td>
</tr>
<tr>
<td>csinhf</td>
<td>Compute hyperbolic sine of a complex float</td>
</tr>
<tr>
<td>ctanh</td>
<td>Compute hyperbolic tangent of a complex float</td>
</tr>
<tr>
<td>ctanhf</td>
<td>Compute hyperbolic tangent of a complex float</td>
</tr>
<tr>
<td><strong>Exponential and logarithmic functions</strong></td>
<td></td>
</tr>
<tr>
<td>cexp</td>
<td>Computes the base-e exponential of a complex float</td>
</tr>
<tr>
<td>cexpf</td>
<td>Computes the base-e exponential of a complex float</td>
</tr>
<tr>
<td>clog</td>
<td>Computes the base-e logarithm of a complex float</td>
</tr>
</tbody>
</table>
**clogf**
Computes the base-e logarithm of a complex float

### Power and absolute value functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cabs</strong></td>
<td>Computes the absolute value of a complex float</td>
</tr>
<tr>
<td><strong>cabsf</strong></td>
<td>Computes the absolute value of a complex float</td>
</tr>
<tr>
<td><strong>cpow</strong></td>
<td>Compute a complex float raised to a power</td>
</tr>
<tr>
<td><strong>cpowf</strong></td>
<td>Compute a complex float raised to a power</td>
</tr>
<tr>
<td><strong>csqrt</strong></td>
<td>Compute square root of a complex float</td>
</tr>
<tr>
<td><strong>csqrtf</strong></td>
<td>Compute square root of a complex float</td>
</tr>
</tbody>
</table>

### Manipulation functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>carg</strong></td>
<td>Compute argument of a complex float</td>
</tr>
<tr>
<td><strong>cargf</strong></td>
<td>Compute argument of a complex float</td>
</tr>
<tr>
<td><strong>cimag</strong></td>
<td>Compute imaginary part of a complex float</td>
</tr>
<tr>
<td><strong>cimagf</strong></td>
<td>Compute imaginary part of a complex float</td>
</tr>
<tr>
<td><strong>conj</strong></td>
<td>Compute conjugate of a complex float</td>
</tr>
<tr>
<td><strong>conjf</strong></td>
<td>Compute conjugate of a complex float</td>
</tr>
<tr>
<td><strong>cproj</strong></td>
<td>Compute projection on the Riemann sphere</td>
</tr>
<tr>
<td><strong>cprojf</strong></td>
<td>Compute projection on the Riemann sphere</td>
</tr>
<tr>
<td><strong>creal</strong></td>
<td>Compute real part of a complex float</td>
</tr>
<tr>
<td><strong>crealf</strong></td>
<td>Compute real part of a complex float</td>
</tr>
</tbody>
</table>
cabs

Synopsis

```c
double cabs(double complex z);
```

Description

cabs returns the absolute value of z.
cabsf

Synopsis

```c
float cabsf(float complex z);
```

Description

cabsf returns the absolute value of z.
**cacos**

**Synopsis**

```c
double complex cacos(double complex z);
```

**Description**

`cacos` returns the principal value the inverse cosine of \( z \) with branch cuts outside the interval \([-1, 1]\) on the real axis. The principal value lies in the interval \([0, \pi]\) on the real axis and in the range of a strip mathematically unbounded on the imaginary axis.
cacosf

Synopsis

```c
float complex cacosf(float complex z);
```

Description

cacosf returns the principal value the inverse cosine of z with branch cuts outside the interval [-1,+1] on the real axis. The principal value lies in the interval [0, ] on the real axis and in the range of a strip mathematically unbounded on the imaginary axis.
cacosh

Synopsis

```c
double complex cacosh(double complex z);
```

Description

`cacosh` returns the principal value the inverse hyperbolic cosine of `z` with branch cuts of values less than 1 on the real axis. The principal value lies in the range of a half-strip of non-negative values on the real axis and in the interval [-i,+i] on the imaginary axis.
cacoshf

Synopsis

```c
float complex cacoshf(float complex _z);
```

Description

cacoshf returns the principal value the inverse hyperbolic cosine of z with branch cuts of values less than 1 on the real axis. The principal value lies in the range of a half-strip of non-negative values on the real axis and in the interval [-i,+i] on the imaginary axis.
carg

Synopsis

```c
double carg(double complex z);
```

Description

carg computes the argument of \( z \) with a branch cut along the negative real axis.
cargf

Synopsis

```c
float cargf(float complex z);
```

Description

cargf computes the argument of z with a branch cut along the negative real axis.
casin

Synopsis

```c
double complex casin(double complex z);
```

Description

casin returns the principal value the inverse sine of z with branch cuts outside the interval [-1,+1] on the real axis. The principal value lies in the interval [, ] on the real axis and in the range of a strip mathematically unbounded on the imaginary axis.
casinf

Synopsis

```c
float complex casinf(float complex z);
```

Description

casinf returns the principal value the inverse sine of \( z \) with branch cuts outside the interval \([-1,+1]\) on the real axis. The principal value lies in the interval \([-\frac{\pi}{2}, +\frac{\pi}{2}]\) on the real axis and in the range of a strip mathematically unbounded on the imaginary axis.
casinh

Synopsis

```c
double complex casinh(double complex z);
```

Description

casinh returns the principal value the inverse hyperbolic sine of z with branch cuts outside the interval [-i,+i] on the imaginary axis. The principal value lies in the range of a strip mathematically unbounded on the real axis and in the interval [-i,+i] on the imaginary axis.
casinhf

Synopsis

```c
float complex casinhf(float complex z);
```

Description

casinhf returns the principal value the inverse hyperbolic sine of \( z \) with branch cuts outside the interval \([-i, +i]\) on the imaginary axis. The principal value lies in the range of a strip mathematically unbounded on the real axis and in the interval \([-i, +i]\) on the imaginary axis.
catan

Synopsis

```c
double complex catan(double complex z);
```

Description

`catan` returns the principal value the inverse sine of `z` with branch cuts outside the interval [-1,+1] on the real axis. The principal value lies in the interval [-\pi/2, \pi/2] on the real axis and in the range of a strip mathematically unbounded on the imaginary axis.
catanf

Synopsis

```c
float complex catanf(float complex z);
```

Description

catanf returns the principal value the inverse sine of z with branch cuts outside the interval [-1,+1] on the real axis. The principal value lies in the interval [, ] on the real axis and in the range of a strip mathematically unbounded on the imaginary axis.
**catanh**

**Synopsis**

```c
double complex catanh(double complex z);
```

**Description**

`catanh` returns the principal value the inverse hyperbolic sine of `z` with branch cuts outside the interval [-1,+1] on the real axis. The principal value lies in the range of a strip mathematically unbounded on the real axis and in the interval [-i,+i] on the imaginary axis.
catanhf

Synopsis

```c
float complex catanhf(float complex z);
```

Description

`catanhf` returns the principal value the inverse hyperbolic sine of `z` with branch cuts outside the interval [-1,+1] on the real axis. The principal value lies in the range of a strip mathematically unbounded on the real axis and in the interval [-i,+i] on the imaginary axis.
**CCOS**

**Synopsis**

```c
double complex ccos(double complex z);
```

**Description**

cos returns the complex cosine of z.
ccosf

Synopsis

```c
float complex ccosf(float complex z);
```

Description

ccosf returns the complex cosine of z.
ccosh

Synopsis

```c
double complex ccosh(double complex z);
```

Description

The `ccosh` function returns the complex hyperbolic cosine of `z`. 
ccoshf

Synopsis

```c
float complex ccoshf(float complex z);
```

Description

`ccoshf` returns the complex hyperbolic cosine of `z`. 
cexp

Synopsis

```c
double complex cexp(double complex z);
```

Description

cexp returns the complex base-e exponential value of z.
cexpf

Synopsis

float complex cexpf(float complex z);

Description

cexpf returns the complex base-e exponential value of z.
cimag

Synopsis

```c
double cimag(double complex);
```

Description

cimag computes the imaginary part of z.
cimagf

Synopsis

```c
float cimagf(float complex);
```

Description

cimagf computes the imaginary part of z.
**clog**

**Synopsis**

```c
double complex clog(double complex z);
```

**Description**

clog returns the complex base-e logarithm value of z.
**clogf**

**Synopsis**

```c
float complex clogf(float complex z);
```

**Description**

clogf returns the complex base-e logarithm value of z.
conj

Synopsis

```c
double complex conj(double complex);
```

Description

`conj` computes the conjugate of \( z \) by reversing the sign of the imaginary part.
conjf

Synopsis

```c
float complex conjf(float complex);
```

Description

conjf computes the conjugate of \( z \) by reversing the sign of the imaginary part.
cPow

Synopsis

```c
double complex cpow(double complex x,
        double complex y);
```

Description

cpow computes \( x \) raised to the power \( y \) with a branch cut for the \( x \) along the negative real axis.
**cpowf**

**Synopsis**

```c
float complex cpowf(float complex x, float complex y);
```

**Description**

`cpowf` computes `x` raised to the power `y` with a branch cut for the `x` along the negative real axis.
cproj

Synopsis

```
double complex cproj(double complex);
```

Description

cproj computes the projection of z on the Riemann sphere.
cprojf

Synopsis

```c
float complex cprojf(float complex);
```

Description

cprojf computes the projection of \( z \) on the Riemann sphere.
creal

Synopsis

```
double creal(double complex);
```

Description

`creal` computes the real part of `z`. 
crealf

Synopsis

```c
float crealf(float complex);
```

Description

crealf computes the real part of z.
csin

Synopsis

double complex csin(double complex z);

Description

csin returns the complex sine of z.
csinf

Synopsis

```c
float complex csinf(float complex z);
```

Description

csinf returns the complex sine of z.
csinh

Synopsis

double complex csinh(double complex z);

Description

csinh returns the complex hyperbolic sine of z.
csinhf

Synopsis

```c
float complex csinhf(float complex z);
```

Description

csinhf returns the complex hyperbolic sine of z.
csqrt

Synopsis

```c
double complex csqrt(double complex z);
```

Description

csqrt computes the complex square root of z with a branch cut along the negative real axis.
csqrtf

Synopsis

```c
float complex csqrtf(float complex z);
```

Description

csqrtf computes the complex square root of z with a branch cut along the negative real axis.
ctan

Synopsis

double complex ctan(double complex z);

Description

ctan returns the complex tangent of z.
ctanf

Synopsis

```c
float complex ctanf(float complex z);
```

Description

`ctanf` returns the complex tangent of `z`. 
ctanh

Synopsis

```c
double complex ctanh(double complex z);
```

Description

`ctanh` returns the complex hyperbolic tangent of `z`. 
ctanhf

Synopsis

```c
float complex ctanhf(float complex z);
```

Description

canhf returns the complex hyperbolic tangent of z.
## API Summary

### Classification functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isalnum</code></td>
<td>Is character alphanumeric?</td>
</tr>
<tr>
<td><code>isalpha</code></td>
<td>Is character alphabetic?</td>
</tr>
<tr>
<td><code>isblank</code></td>
<td>Is character a space or horizontal tab?</td>
</tr>
<tr>
<td><code>iscntrl</code></td>
<td>Is character a control?</td>
</tr>
<tr>
<td><code>isdigit</code></td>
<td>Is character a decimal digit?</td>
</tr>
<tr>
<td><code>isgraph</code></td>
<td>Is character any printing character except space?</td>
</tr>
<tr>
<td><code>islower</code></td>
<td>Is character a lowercase letter?</td>
</tr>
<tr>
<td><code>isprint</code></td>
<td>Is character printable?</td>
</tr>
<tr>
<td><code>ispunct</code></td>
<td>Is character a punctuation mark?</td>
</tr>
<tr>
<td><code>isspace</code></td>
<td>Is character a whitespace character?</td>
</tr>
<tr>
<td><code>isupper</code></td>
<td>Is character an uppercase letter?</td>
</tr>
<tr>
<td><code>isxdigit</code></td>
<td>Is character a hexadecimal digit?</td>
</tr>
</tbody>
</table>

### Conversion functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tolower</code></td>
<td>Convert uppercase character to lowercase</td>
</tr>
<tr>
<td><code>toupper</code></td>
<td>Convert lowercase character to uppercase</td>
</tr>
</tbody>
</table>

### Classification functions (extended)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isalnum_l</code></td>
<td>Is character alphanumeric?</td>
</tr>
<tr>
<td><code>isalpha_l</code></td>
<td>Is character alphabetic?</td>
</tr>
<tr>
<td><code>isblank_l</code></td>
<td>Is character a space or horizontal tab?</td>
</tr>
<tr>
<td><code>iscntrl_l</code></td>
<td>Is character a control character?</td>
</tr>
<tr>
<td><code>isdigit_l</code></td>
<td>Is character a decimal digit?</td>
</tr>
<tr>
<td><code>isgraph_l</code></td>
<td>Is character any printing character except space?</td>
</tr>
<tr>
<td><code>islower_l</code></td>
<td>Is character a lowercase letter?</td>
</tr>
<tr>
<td><code>isprint_l</code></td>
<td>Is character printable?</td>
</tr>
<tr>
<td><code>ispunct_l</code></td>
<td>Is character a punctuation mark?</td>
</tr>
<tr>
<td><code>isspace_l</code></td>
<td>Is character a whitespace character?</td>
</tr>
<tr>
<td><code>isupper_l</code></td>
<td>Is character an uppercase letter?</td>
</tr>
<tr>
<td><code>isxdigit_l</code></td>
<td>Is character a hexadecimal digit?</td>
</tr>
</tbody>
</table>

### Conversion functions (extended)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tolower_l</code></td>
<td>Convert uppercase character to lowercase</td>
</tr>
<tr>
<td>toupper_l</td>
<td>Convert lowercase character to uppercase</td>
</tr>
</tbody>
</table>
isalnum

Synopsis

```c
int isalnum(int c);
```

Description

`isalnum` returns nonzero (true) if and only if the value of the argument `c` is an alphabetic or numeric character.
isalnum_l

Synopsis

```c
int isalnum_l(int c,
locale_t loc);
```

Description

`isalnum_l` returns nonzero (true) if and only if the value of the argument `c` is a alphabetic or numeric character in locale `loc`. 
isalpha

Synopsis

```c
int isalpha(int c);
```

Description

isalpha returns true if the character c is alphabetic. That is, any character for which isupper or islower returns true is considered alphabetic in addition to any of the locale-specific set of alphabetic characters for which none of iscntrl, isdigit, ispunct, or isspace is true.

In the C locale, isalpha returns nonzero (true) if and only if isupper or islower return true for value of the argument c.
isalpha_l

Synopsis

```c
int isalpha_l(int c,
              locale_t loc);
```

Description

isalpha_l returns nonzero (true) if and only if isupper or islower return true for value of the argument c in locale loc.
isblank

Synopsis

```c
int isblank(int c);
```

Description

isblank returns nonzero (true) if and only if the value of the argument c is either a space character (' ') or the horizontal tab character ('\t').
isblank_l

Synopsis

```c
int isblank_l(int c, locale_t loc);
```

Description

isblank_l returns nonzero (true) if and only if the value of the argument c is either a space character (' ') or the horizontal tab character ('\t') in locale loc.
iscntrl

Synopsis

```c
int iscntrl(int c);
```

Description

iscntrl returns nonzero (true) if and only if the value of the argument `c` is a control character. Control characters have values 0 through 31 and the single value 127.
iscntrl_l

Synopsis

```c
int iscntrl_l(int c,
    locale_t loc);
```

Description

iscntrl_l returns nonzero (true) if and only if the value of the argument c is a control character in locale loc.
is digit

Synopsis

```c
int isdigit(int c);
```

Description

isdigit returns nonzero (true) if and only if the value of the argument `c` is a digit.
isdigit_l

Synopsis

```c
int isdigit_l(int c,
              locale_t loc);
```

Description

isdigit_l returns nonzero (true) if and only if the value of the argument c is a decimal digit in locale loc.
isgraph

Synopsis

```c
int isgraph(int c);
```

Description

`isgraph` returns nonzero (true) if and only if the value of the argument `c` is any printing character except space (').
isgraph_l

Synopsis

```c
int isgraph_l(int c,
locale_t loc);
```

Description

isgraph_l returns nonzero (true) if and only if the value of the argument `c` is any printing character except space (’ ’) in locale `loc`. 
islower

Synopsis

```c
int islower(int c);
```

Description

`islower` returns nonzero (true) if and only if the value of the argument `c` is an lowercase letter.
islower_l

Synopsis

```c
int islower_l(int c,
    locale_t loc);
```

Description

`islower_l` returns nonzero (true) if and only if the value of the argument `c` is an lowercase letter in locale `loc`. 
isprint

Synopsis

```c
int isprint(int c);
```

Description

isprint returns nonzero (true) if and only if the value of the argument `c` is any printing character including space (`' '`).
isprint_l

Synopsis

```
int isprint_l(int c,
    locale_t loc);
```

Description

`isprint_l` returns nonzero (true) if and only if the value of the argument `c` is any printing character including space (" ") in locale `loc`. 
ispunct

Synopsis

```c
int ispunct(int c);
```

Description

ispunct returns nonzero (true) for every printing character for which neither isspace nor isalnum is true.
ispunct_l

Synopsis

```c
int ispunct_l(int c, locale_t loc);
```

Description

`ispunct_l` returns nonzero (true) for every printing character for which neither `isspace` nor `isalnum` is true in in locale `loc`. 
**isspace**

**Synopsis**

```c
int isspace(int c);
```

**Description**

`isspace` returns nonzero (true) if and only if the value of the argument `c` is a standard white-space character. The standard white-space characters are space ( ' ' ), form feed ( '\f' ), new-line ( '\n' ), carriage return ( '\r' ), horizontal tab ( '\t' ), and vertical tab ( '\v' ).
isspace_l

Synopsis

```c
int isspace_l(int c,
    locale_t loc);
```

Description

isspace_l returns nonzero (true) if and only if the value of the argument c is a standard white-space character in locale loc.
isupper

Synopsis

```
int isupper(int c);
```

Description

isupper returns nonzero (true) if and only if the value of the argument c is an uppercase letter.
isupper_l

Synopsis

```c
int isupper_l(int c,
    locale_t loc);
```

Description

isupper_l returns nonzero (true) if and only if the value of the argument c is an uppercase letter in locale loc.
isxdigit

Synopsis

```c
int isxdigit(int c);
```

Description

`isxdigit` returns nonzero (true) if and only if the value of the argument `c` is a hexadecimal digit.
isxdigit_l

Synopsis

```c
int isxdigit_l(int c,
    locale_t loc);
```

Description

isxdigit_l returns nonzero (true) if and only if the value of the argument c is a hexadecimal digit in locale loc.
tolower

Synopsis

```c
int tolower(int c);
```

Description

tolower converts an uppercase letter to a corresponding lowercase letter. If the argument `c` is a character for which `isupper` is true and there are one or more corresponding characters, as specified by the current locale, for which `islower` is true, the `tolower` function returns one of the corresponding characters (always the same one for any given locale); otherwise, the argument is returned unchanged.

Note that even though `isupper` can return true for some characters, `tolower` may return that uppercase character unchanged as there are no corresponding lowercase characters in the locale.
tolower_l

Synopsis

```c
int tolower_l(int c, locale_t loc);
```

Description

towler_l converts an uppercase letter to a corresponding lowercase letter in locale loc. If the argument c is a character for which isupper is true in locale loc, tolower_l returns the corresponding lowercase letter; otherwise, the argument is returned unchanged.
Synopsis

```c
int toupper(int c);
```

Description

toupper converts a lowercase letter to a corresponding uppercase letter. If the argument is a character for which islower is true and there are one or more corresponding characters, as specified by the current locale, for which isupper is true, toupper returns one of the corresponding characters (always the same one for any given locale); otherwise, the argument is returned unchanged. Note that even though islower can return true for some characters, toupper may return that lowercase character unchanged as there are no corresponding uppercase characters in the locale.
toupper_l

Synopsis

```c
int toupper_l(int c,
               locale_t loc);
```

Description

toupper_l converts a lowercase letter to a corresponding uppercase letter in locale loc. If the argument c is a character for which islower is true in locale loc, toupper_l returns the corresponding uppercase letter; otherwise, the argument is returned unchanged.
## API Summary

### File Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>debug_clearerr</code></td>
<td>Clear error indicator</td>
</tr>
<tr>
<td><code>debug_fclose</code></td>
<td>Closes an open stream</td>
</tr>
<tr>
<td><code>debug_feof</code></td>
<td>Check end of file condition</td>
</tr>
<tr>
<td><code>debug_ferror</code></td>
<td>Check error indicator</td>
</tr>
<tr>
<td><code>debug_fflush</code></td>
<td>Flushes buffered output</td>
</tr>
<tr>
<td><code>debug_fgetc</code></td>
<td>Read a character from a stream</td>
</tr>
<tr>
<td><code>debug_fgetpos</code></td>
<td>Return file position</td>
</tr>
<tr>
<td><code>debug_fgets</code></td>
<td>Read a string</td>
</tr>
<tr>
<td><code>debug_filesize</code></td>
<td>Return the size of a file</td>
</tr>
<tr>
<td><code>debug_fopen</code></td>
<td>Opens a file on the host PC</td>
</tr>
<tr>
<td><code>debug_fprintf</code></td>
<td>Formatted write</td>
</tr>
<tr>
<td><code>debug_fprintf_c</code></td>
<td>Formatted write</td>
</tr>
<tr>
<td><code>debug_fputc</code></td>
<td>Write a character</td>
</tr>
<tr>
<td><code>debug_fputs</code></td>
<td>Write a string</td>
</tr>
<tr>
<td><code>debug_fread</code></td>
<td>Read data</td>
</tr>
<tr>
<td><code>debug_freopen</code></td>
<td>Reopens a file on the host PC</td>
</tr>
<tr>
<td><code>debug_fscanf</code></td>
<td>Formatted read</td>
</tr>
<tr>
<td><code>debug_fscanf_c</code></td>
<td>Formatted read</td>
</tr>
<tr>
<td><code>debug_fseek</code></td>
<td>Set file position</td>
</tr>
<tr>
<td><code>debug_fsetpos</code></td>
<td>Return file position</td>
</tr>
<tr>
<td><code>debug_ftell</code></td>
<td>Return file position</td>
</tr>
<tr>
<td><code>debug_fwrite</code></td>
<td>Write data</td>
</tr>
<tr>
<td><code>debug_remove</code></td>
<td>Deletes a file on the host PC</td>
</tr>
<tr>
<td><code>debug_rename</code></td>
<td>Renames a file on the host PC</td>
</tr>
<tr>
<td><code>debug_rewind</code></td>
<td>Set file position to the beginning</td>
</tr>
<tr>
<td><code>debug_tmpfile</code></td>
<td>Open a temporary file</td>
</tr>
<tr>
<td><code>debug_tmpnam</code></td>
<td>Generate temporary filename</td>
</tr>
<tr>
<td><code>debug_ungetc</code></td>
<td>Push a character</td>
</tr>
<tr>
<td><code>debug_vfprintf</code></td>
<td>Formatted write</td>
</tr>
<tr>
<td><code>debug_vfscanf</code></td>
<td>Formatted read</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td><strong>Debug Terminal Output Functions</strong></td>
<td></td>
</tr>
<tr>
<td>debug_printf</td>
<td>Formatted write</td>
</tr>
<tr>
<td>debug_printf_c</td>
<td>Formatted write</td>
</tr>
<tr>
<td>debug_putchar</td>
<td>Write a character</td>
</tr>
<tr>
<td>debug_puts</td>
<td>Write a string</td>
</tr>
<tr>
<td>debug_vlog</td>
<td>Formatted write</td>
</tr>
<tr>
<td><strong>Debug Terminal Input Functions</strong></td>
<td></td>
</tr>
<tr>
<td>debug_getch</td>
<td>Blocking character read</td>
</tr>
<tr>
<td>debug_getchar</td>
<td>Line-buffered character read</td>
</tr>
<tr>
<td>debug_getd</td>
<td>Line-buffered double read</td>
</tr>
<tr>
<td>debug_getf</td>
<td>Line-buffered float read</td>
</tr>
<tr>
<td>debug_geti</td>
<td>Line-buffered integer read</td>
</tr>
<tr>
<td>debug_getl</td>
<td>Line-buffered long read</td>
</tr>
<tr>
<td>debug_getll</td>
<td>Line-buffered long long read</td>
</tr>
<tr>
<td>debug_gets</td>
<td>String read</td>
</tr>
<tr>
<td>debug_getu</td>
<td>Line-buffered unsigned integer</td>
</tr>
<tr>
<td>debug_getul</td>
<td>Line-buffered unsigned long read</td>
</tr>
<tr>
<td>debug_getull</td>
<td>Line-buffered unsigned long long read</td>
</tr>
<tr>
<td>debug_kbhit</td>
<td>Polled character read</td>
</tr>
<tr>
<td>debug_scanf</td>
<td>Formatted read</td>
</tr>
<tr>
<td>debug_scanf_c</td>
<td>Formatted read</td>
</tr>
<tr>
<td>debug_vscanf</td>
<td>Formatted read</td>
</tr>
<tr>
<td><strong>Debugger Functions</strong></td>
<td></td>
</tr>
<tr>
<td>debug_abort</td>
<td>Stop debugging</td>
</tr>
<tr>
<td>debug_break</td>
<td>Stop target</td>
</tr>
<tr>
<td>debug_enabled</td>
<td>Test if debug input/output is enabled</td>
</tr>
<tr>
<td>debug_exit</td>
<td>Stop debugging</td>
</tr>
<tr>
<td>debug_getargs</td>
<td>Get arguments</td>
</tr>
<tr>
<td>debug_loadsymbols</td>
<td>Load debugging symbols</td>
</tr>
<tr>
<td>debug_runtime_error</td>
<td>Stop and report error</td>
</tr>
<tr>
<td>debug_unloadsymbols</td>
<td>Unload debugging symbols</td>
</tr>
<tr>
<td><strong>Misc Functions</strong></td>
<td></td>
</tr>
<tr>
<td>debug_getenv</td>
<td>Get environment variable value</td>
</tr>
<tr>
<td>debug_perror</td>
<td>Display error</td>
</tr>
<tr>
<td>debug_system</td>
<td>Execute command</td>
</tr>
<tr>
<td>debug_time</td>
<td>get time</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
</tr>
</tbody>
</table>

debug_abort

Synopsis

```c
void debug_abort(void);
```

Description

debug_abort causes the debugger to exit and a failure result is returned to the user.
debug_break

Synopsis

```
void debug_break(void);
```

Description

`debug_break` causes the debugger to stop the target and position the cursor at the line that called `debug_break`. 
debug_clearerr

Synopsis

```c
void debug_clearerr(DEBUG_FILE *stream);
```

Description

depbug_clearerr clears any error indicator or end of file condition for the stream.
debug_enabled

Synopsis

```c
int debug_enabled(void);
```

Description

d debug_enabled returns non-zero if the debugger is connected - you can use this to test if a debug input/output functions will work. For this to work correctly, the Startup Completion Breakpoint project property needs to be set to a point in the program where the startup code has finished initialising, this is typically main.
debug_exit

Synopsis

```c
void debug_exit(int result);
```

Description

`debug_exit` causes the debugger to exit and `result` is returned to the user.
debug_fclose

Synopsis

```c
int debug_fclose(DEBUG_FILE *stream);
```

Description

depth_fclose flushes any buffered output of the stream and then closes the stream.

depth_fclose returns 0 on success or -1 if there was an error.
debug_feof

Synopsis

```c
int debug_feof(DEBUG_FILE *stream);
```

Description

d debug_feof returns non-zero if the end of file condition is set for the stream.
debug_ferror

Synopsis

```c
int debug_ferror(DEBUG_FILE *stream);
```

Description

depth_ferror returns non-zero if the error indicator is set for the stream.
debug_fflush

Synopsis

```c
int debug_fflush(DEBUG_FILE *stream);
```

Description

ddebug_fflush flushes any buffered output of the stream.
ddebug_fflush returns 0 on success or -1 if there was an error.
debug_fgetc

Synopsis

```c
int debug_fgetc(DEBUG_FILE *stream);
```

Description

ddebug_fgetc reads and returns the next character on stream or -1 if no character is available.
debug_fgetpos

Synopsis

```c
int debug_fgetpos(DEBUG_FILE *stream,
                    long *pos);
```

Description

`debug_fgetpos` is equivalent to `debug_fseek`.

**debug_fgets**

**Synopsis**

```c
char *debug_fgets(char *s,
                   int n,
                   DEBUG_FILE *stream);
```

**Description**

**debug_fgets** reads at most \(n-1\) characters or the characters up to (and including) a newline from the input stream into the array pointed to by \(s\). A null character is written to the array after the input characters.

**debug_fgets** returns \(s\) on success, or 0 on error or end of file.
**debug_filesize**

**Synopsis**

```c
int debug_filesize(DEBUG_FILE *stream);
```

**Description**

`debug_filesize` returns the size of the file associated with the `stream` in bytes.

`debug_filesize` returns -1 on error.
**debug_fopen**

**Synopsis**

```c
DEBUG_FILE *debug_fopen(const char *filename,
                         const char *mode);
```

**Description**

`debug_fopen` opens the `filename` on the host PC and returns a stream or 0 if the open fails. The `filename` is a host PC filename which is opened relative to the debugger working directory. The `mode` is a string containing one of:

- r open file for reading.
- w create file for writing.
- a open or create file for writing and position at the end of the file.
- r+ open file for reading and writing.
- w+ create file for reading and writing.
- a+ open or create text file for reading and writing and position at the end of the file.

followed by one of:

- t for a text file.
- b for a binary file.

`debug_fopen` returns a stream that can be used to access the file or 0 if the open fails.
debug_fprintf

Synopsis

```c
int debug_fprintf(DEBUG_FILE *stream,
                   const char *format,
                   ...);
```

Description

debug_fprintf writes to stream, under control of the string pointed to by format that specifies how subsequent arguments are converted for output. The format string is a standard C printf format string. The actual formatting is performed on the host by the debugger and therefore debug_fprintf consumes only a very small amount of code and data space, only the overhead to call the function.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

debug_fprintf returns the number of characters transmitted, or a negative value if an output or encoding error occurred.
debug_fprintf_c

Synopsis

```c
int debug_fprintf_c (DEBUG_FILE *stream,
    __code const char *format,
    ...);
```

Description

debug_fprintf_c is equivalent to debug_fprintf with the format string in code memory.
debug_fputc

Synopsis

```c
int debug_fputc(int c,
                DEBUG_FILE *stream);
```

Description

debug_fputc writes the character \(c\) to the output \(stream\).

debug_fputc returns the character written or -1 if an error occurred.
debug_fputs

Synopsis

```
int debug_fputs(const char *s,
               DEBUG_FILE *stream);
```

Description

`debug_fputs` writes the string pointed to by `s` to the output `stream` and appends a new-line character. The terminating null character is not written.

`debug_fputs` returns -1 if a write error occurs; otherwise it returns a nonnegative value.
debug_fread

Synopsis

```c
int debug_fread(void *ptr,
    int size,
    int nobj,
    DEBUG_FILE *stream);
```

Description

default_fread reads from the input stream into the array ptr at most nobj objects of size size.

default_fread returns the number of objects read. If this number is different from nobj then debug_feof and debug_ferror can be used to determine status.
debug_freopen

Synopsis

```c
DEBUG_FILE *debug_freopen(const char *filename,
                           const char *mode,
                           DEBUG_FILE *stream);
```

Description

debug_freopen is the same as debug_open except the file associated with the stream is closed and the opened file is then associated with the stream.
debug_fscanf

Synopsis

```c
int debug_fscanf(DEBUG_FILE *stream,
    const char *format,
    ...);
```

Description

dump_fscanf reads from the input `stream`, under control of the string pointed to by `format`, that specifies how subsequent arguments are converted for input. The `format` string is a standard C scanf format string. The actual formatting is performed on the host by the debugger and therefore `debug_fscanf` consumes only a very small amount of code and data space, only the overhead to call the function.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

dump_fscanf returns number of characters read, or a negative value if an output or encoding error occurred.
debug_fscanf_c

Synopsis

```c
int debug_fscanf_c (DEBUG_FILE *stream,
                    __code const char *format,
                    ...);
```

Description

debug_fscanf_c is equivalent to debug_fscanf with the format string in code memory.
**debug_fseek**

**Synopsis**

```c
int debug_fseek(DEBUG_FILE *stream,
                 long offset,
                 int origin);
```

**Description**

`debug_fseek` sets the file position for the `stream`. A subsequent read or write will access data at that position. The `origin` can be one of:

- 0 sets the position to `offset` bytes from the beginning of the file.
- 1 sets the position to `offset` bytes relative to the current position.
- 2 sets the position to `offset` bytes from the end of the file.

Note that for text files `offset` must be zero. `debug_fseek` returns zero on success, non-zero on error.
**debug_fsetpos**

**Synopsis**

```c
int debug_fsetpos(DEBUG_FILE *stream,
                   const long *pos);
```

**Description**

`debug_fsetpos` is equivalent to `debug_fseek` with 0 as the origin.
debug_ftell

Synopsis

```c
long debug_ftell(DBG_FILE *stream);
```

Description

d debug_ftell returns the current file position of the stream.

d debug_ftell returns -1 on error.
debug_fwrite

Synopsis

```c
int debug_fwrite(const void *ptr,
                 int size,
                 int nobj,
                 DEBUG_FILE *stream);
```

Description

ddebug_fwrite write to the output stream from the array ptr at most nobj objects of size size.

ddebug_fwrite returns the number of objects written. If this number is different from nobj then debug_feof and debug_ferror can be used to determine status.
debug_getargs

Synopsis

```c
int debug_getargs(unsigned bufsize,
                  unsigned char *buf);
```

Description

depug_getargs stores the debugger command line arguments into the memory pointed at by buf up to a maximum of bufsize bytes. The command line is stored as a C argc array of null terminated string and the number of entries is returned as the result.
debug_getch

Synopsis

```c
int debug_getch(void);
```

Description

depug_getch reads one character from the Debug Terminal. This function will block until a character is available.
**debug_getchar**

**Synopsis**

```c
int debug_getchar(void);
```

**Description**

`debug_getchar` reads one character from the **Debug Terminal**. This function uses line input and will therefore block until characters are available and ENTER has been pressed.

`debug_getchar` returns the character that has been read.
debug_getd

Synopsis

```
int debug_getd(double *);
```

Description

d debug_getd reads a double from the Debug Terminal. The number is written to the double object pointed to by d.

d debug_getd returns zero on success or -1 on error.
debug_getenv

Synopsis

```c
char *debug_getenv(char *name);
```

Description

d debug_getenv returns the value of the environment variable name or 0 if the environment variable cannot be found.
debug_getf

Synopsis

```c
int debug_getf(float *f);
```

Description

ddebug_getf reads an float from the Debug Terminal. The number is written to the float object pointed to by f.

ddebug_getf returns zero on success or -1 on error.
debug_geti

Synopsis

```c
int debug_geti(int *i);
```

Description

depbug_geti reads an integer from the Debug Terminal. If the number starts with `0x` it is interpreted as a hexadecimal number, if it starts with `0` it is interpreted as an octal number, if it starts with `0b` it is interpreted as a binary number, otherwise it is interpreted as a decimal number. The number is written to the integer object pointed to by `i`.

depbug_geti returns zero on success or -1 on error.
debug_getl

Synopsis

```c
int debug_getl(long *l);
```

Description

depug_getl reads a long from the Debug Terminal. If the number starts with `0x` it is interpreted as a hexadecimal number, if it starts with `0` it is interpreted as an octal number, if it starts with it is interpreted as a binary number, otherwise it is interpreted as a decimal number. The number is written to the long object pointed to by `l`.

depug_getl returns zero on success or -1 on error.
debug_getll

Synopsis

```c
int debug_getll(long long *ll);
```

Description

depug_getll reads a long long from the Debug Terminal. If the number starts with `0x` it is interpreted as a hexadecimal number, if it starts with `0` it is interpreted as an octal number, if it starts with `0b` it is interpreted as a binary number, otherwise it is interpreted as a decimal number. The number is written to the long long object pointed to by `ll`.

depug_getll returns zero on success or -1 on error.
debug_gets

Synopsis

```c
char *debug_gets(char *s);
```

Description

depbug_gets reads a string from the Debug Terminal in memory pointed at by s. This function will block until ENTER has been pressed.

depbug_gets returns the value of s.
debug_getu

Synopsis

```c
int debug_getu(unsigned *u);
```

Description

d debug_getu reads an unsigned integer from the Debug Terminal. If the number starts with 0x it is interpreted as a hexadecimal number, if it starts with 0 it is interpreted as an octal number, if it starts with 0b it is interpreted as a binary number, otherwise it is interpreted as a decimal number. The number is written to the unsigned integer object pointed to by u.

d debug_getu returns zero on success or -1 on error.
debug_getul

Synopsis

```c
int debug_getul(unsigned long *ul);
```

Description

`debug_getul` reads an unsigned long from the Debug Terminal. If the number starts with `0x` it is interpreted as a hexadecimal number, if it starts with `0` it is interpreted as an octal number, if it starts with `0b` it is interpreted as a binary number, otherwise it is interpreted as a decimal number. The number is written to the long object pointed to by `ul`.

`debug_getul` returns zero on success or -1 on error.
debug_getull

Synopsis

```c
int debug_getull(unsigned long long *ull);
```

Description

download reads an unsigned long long from the Debug Terminal. If the number starts with 0x it is interpreted as a hexadecimal number, if it starts with 0 it is interpreted as an octal number, if it starts with 0b it is interpreted as a binary number, otherwise it is interpreted as a decimal number. The number is written to the long long object pointed to by ull.

download returns zero on success or -1 on error.
debug_kbhit

Synopsis

```c
int debug_kbhit(void);
```

Description

debug_kbhit polls the Debug Terminal for a character and returns a non-zero value if a character is available or 0 if not.
debug_loadsymbols

Synopsis

```c
void debug_loadsymbols(const char *filename,
                        const void *address,
                        const char *breaksymbol);
```

Description

debug_loadsymbols instructs the debugger to load the debugging symbols in the file denoted by filename. The filename is a (macro expanded) host PC filename which is relative to the debugger working directory. The address is the load address which is required for debugging position independent executables, supply NULL for regular executables. The breaksymbol is the name of a symbol in the filename to set a temporary breakpoint on or NULL.
debug_perror

Synopsis

```c
void debug_perror(const char *s);
```

Description

d debug_perror displays the optional string s on the Debug Terminal together with a string corresponding to the errno value of the last Debug IO operation.
debug_printf

Synopsis

```c
int debug_printf(const char *format, ...
```

Description

ddebug_printf writes to the Debug Terminal, under control of the string pointed to by format that specifies how subsequent arguments are converted for output. The format string is a standard C printf format string. The actual formatting is performed on the host by the debugger and therefore debug_printf consumes only a very small amount of code and data space, only the overhead to call the function.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

ddebug_printf returns the number of characters transmitted, or a negative value if an output or encoding error occurred.
debug_printf_c

Synopsis

```c
int debug_printf_c(__code const char *format,
                   ...);
```

Description

dumpintf_c is equivalent to debug_printf with the format string in code memory.
debug_putchar

Synopsis

```c
int debug_putchar(int c);
```

Description

deput_putchar write the character c to the Debug Terminal.

deput_putchar returns the character written or -1 if a write error occurs.
debug_puts

Synopsis

```c
int debug_puts(const char *);
```

Description

d debug_puts writes the string s to the Debug Terminal followed by a new-line character.
d debug_puts returns -1 if a write error occurs, otherwise it returns a nonnegative value.
debug_remove

Synopsis

```c
int debug_remove(const char *filename);
```

Description

ddebug_remove removes the filename denoted by filename and returns 0 on success or -1 on error. The filename is a host PC filename which is relative to the debugger working directory.
debug_rename

Synopsis

```c
int debug_rename(const char *oldfilename,
                 const char *newfilename);
```

Description

depuginame renames the file denoted by oldpath to newpath and returns zero on success or non-zero on error. The oldpath and newpath are host PC filenames which are relative to the debugger working directory.
debug_rewind

Synopsis

void debug_rewind(DEBUG_FILE *stream);

Description

ddebug_rewind sets the current file position of the stream to the beginning of the file and clears any error and end of file conditions.
**debug_runtime_error**

**Synopsis**

```c
void debug_runtime_error(const char *error);
```

**Description**

depbug_runtime_error causes the debugger to stop the target, position the cursor at the line that called debug_runtime_error, and display the null-terminated string pointed to by error.
**debug_scanf**

**Synopsis**

```c
int debug_scanf(const char *format,
               ...);
```

**Description**

`debug_scanf` reads from the Debug Terminal, under control of the string pointed to by `format` that specifies how subsequent arguments are converted for input. The `format` string is a standard C `scanf` format string. The actual formatting is performed on the host by the debugger and therefore `debug_scanf` consumes only a very small amount of code and data space, only the overhead to call the function.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

`debug_scanf` returns number of characters read, or a negative value if an output or encoding error occurred.
debug_scanf_c

Synopsis

```c
int debug_scanf_c(__code const char *format,
...);
```

Description

debug_scanf_c is equivalent to debug_scanf with the format string in code memory.
**debug_system**

**Synopsis**

```c
int debug_system(char *command);
```

**Description**

`debug_system` executes the **command** with the host command line interpreter and returns the commands exit status.
debug_time

Synopsis

```c
long debug_time(long *ptr);
```

Description

depug_time returns the number of seconds elapsed since midnight (00:00:00), January 1, 1970, coordinated universal time (UTC), according to the system clock of the host computer. The return value is stored in *ptr if ptr is not NULL.
debug_tmpfile

Synopsis

```c
DEBUG_FILE *debug_tmpfile(void);
```

Description

depag_tmpfile creates a temporary file on the host PC which is deleted when the stream is closed.
debug_tmpnam

Synopsis

```c
char *debug_tmpnam(char *str);
```

Description

debug_tmpnam returns a unique temporary filename. If `str` is NULL then a static buffer is used to store the filename, otherwise the filename is stored in `str`. On success a pointer to the string is returned, on failure 0 is returned.
debug_ungetc

Synopsis

```c
int debug_ungetc(int c,
                 DEBUG_FILE *stream);
```

Description

`debug_ungetc` pushes the character `c` onto the input `stream`. If successful `c` is returned, otherwise -1 is returned.
debug_unloadsymbols

Synopsis

```c
void debug_unloadsymbols(const char *filename);
```

Description

depend_unloadsymbols instructs the debugger to unload the debugging symbols (previously loaded by a call to debug_loadsymbols) in the file denoted by filename. The filename is a host PC filename which is relative to the debugger working directory.
debug_vfprintf

Synopsis

```c
int debug_vfprintf(DEBUG_FILE *stream,
                    const char *format,
                    __va_list);
```

Description

depug_vfprintf is equivalent to debug_fprintf with arguments passed using stdarg.h rather than a variable number of arguments.
debug_vfscanf

Synopsis

```c
int debug_vfscanf(DEBUG_FILE *stream,
                   const char *format,
                   __va_list);
```

Description

ddebug_vfscanf is equivalent to debug_fscanf with arguments passed using stdarg.h rather than a variable number of arguments.
debug_vprintf

Synopsis

```c
int debug_vprintf(const char *format, __va_list);
```

Description

depbug_vprintf is equivalent to debug_printf with arguments passed using stdarg.h rather than a variable number of arguments.
debug_vscanf

Synopsis

```c
int debug_vscanf(const char *format, __va_list);
```

Description

`debug_vscanf` is equivalent to `debug_scanf` with arguments passed using `stdarg.h` rather than a variable number of arguments.
<errno.h>

## API Summary

### Error numbers

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDOM</td>
<td>Domain error</td>
</tr>
<tr>
<td>EILSEQ</td>
<td>Illegal byte sequence</td>
</tr>
<tr>
<td>EINVAL</td>
<td>Invalid argument</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>No memory available</td>
</tr>
<tr>
<td>ERANGE</td>
<td>Result too large or too small</td>
</tr>
</tbody>
</table>

### Macros

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errno</td>
<td>Last-set error condition</td>
</tr>
</tbody>
</table>
EDOM

Synopsis

```c
#define EDOM ...
```

Description

EDOM - an input argument is outside the defined domain of a mathematical function.
EILSEQ

Synopsis

#define EILSEQ ...

Description

EILSEQ - A wide-character code has been detected that does not correspond to a valid character, or a byte sequence does not form a valid wide-character code.
EINVAL

Synopsis

#define EINVAL 0x06

Description

EINVAL - An argument was invalid, or a combination of arguments was invalid.
ENOMEM

Synopsis

#define ENOMEM 0x05

Description

ENOMEM - no memory can be allocated by a function in the library. Note that malloc, calloc, and realloc do not set errno to ENOMEM on failure, but other library routines (such as duplocale) may set errno to ENOMEM when memory allocation fails.
ERANGE

Synopsis

#define ERANGE ...

Description

ERANGE - the result of the function is too large (overflow) or too small (underflow) to be represented in the available space.
errno

Synopsis

```
int errno;
```

Description

errno is treated as an writable l-value, but the implementation of how the l-value is read an written is hidden from the user.

The value of errno is zero at program startup, but is never set to zero by any library function. The value of errno may be set to a nonzero value by a library function, and this effect is documented in each function that does so.

Note

The ISO standard does not specify whether errno is a macro or an identifier declared with external linkage. Portable programs must not make assumptions about the implementation of errno.

In this implementation, errno expands to a function call to __errno (MSP430, AVR, MAXQ) or __aeabi_errno_addr (ARM) that returns a pointer to a volatile int. This function can be implemented by the application to provide a thread-specific errno.
### <float.h>

#### API Summary

<table>
<thead>
<tr>
<th>Double exponent minimum and maximum values</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DBL_MAX_10_EXP</strong></td>
<td>The maximum exponent value in base 10 of a double</td>
</tr>
<tr>
<td><strong>DBL_MAX_EXP</strong></td>
<td>The maximum exponent value of a double</td>
</tr>
<tr>
<td><strong>DBL_MIN_10_EXP</strong></td>
<td>The minimal exponent value in base 10 of a double</td>
</tr>
<tr>
<td><strong>DBL_MIN_EXP</strong></td>
<td>The minimal exponent value of a double</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Implementation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DBL_DIG</strong></td>
<td>The number of digits of precision of a double</td>
</tr>
<tr>
<td><strong>DBL_MANT_DIG</strong></td>
<td>The number of digits in a double</td>
</tr>
<tr>
<td><strong>DECIMAL_DIG</strong></td>
<td>The number of decimal digits that can be rounded without change</td>
</tr>
<tr>
<td><strong>FLT_DIG</strong></td>
<td>The number of digits of precision of a float</td>
</tr>
<tr>
<td><strong>FLT_EVAL_METHOD</strong></td>
<td>The evaluation format</td>
</tr>
<tr>
<td><strong>FLT_MANT_DIG</strong></td>
<td>The number of digits in a float</td>
</tr>
<tr>
<td><strong>FLT_RADIX</strong></td>
<td>The radix of the exponent representation</td>
</tr>
<tr>
<td><strong>FLT_ROUNDS</strong></td>
<td>The rounding mode</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Float exponent minimum and maximum values</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FLT_MAX_10_EXP</strong></td>
<td>The maximum exponent value in base 10 of a float</td>
</tr>
<tr>
<td><strong>FLT_MAX_EXP</strong></td>
<td>The maximum exponent value of a float</td>
</tr>
<tr>
<td><strong>FLT_MIN_10_EXP</strong></td>
<td>The minimal exponent value in base 10 of a float</td>
</tr>
<tr>
<td><strong>FLT_MIN_EXP</strong></td>
<td>The minimal exponent value of a float</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Double minimum and maximum values</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DBL_EPSILON</strong></td>
<td>The difference between 1 and the least value greater than 1 of a double</td>
</tr>
<tr>
<td><strong>DBL_MAX</strong></td>
<td>The maximum value of a double</td>
</tr>
<tr>
<td><strong>DBL_MIN</strong></td>
<td>The minimal value of a double</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Float minimum and maximum values</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FLT_EPSILON</strong></td>
<td>The difference between 1 and the least value greater than 1 of a float</td>
</tr>
<tr>
<td><strong>FLT_MAX</strong></td>
<td>The maximum value of a float</td>
</tr>
<tr>
<td><strong>FLT_MIN</strong></td>
<td>The minimal value of a float</td>
</tr>
</tbody>
</table>
DBL_DIG

Synopsis

```c
#define DBL_DIG 15
```

Description

DBL_DIG specifies the number of digits of precision of a `double`. 
**DBL_EPSILON**

**Synopsis**

```c
#define DBL_EPSILON 2.2204460492503131E-16
```

**Description**

DBL_EPSILON the minimum positive number such that 1.0 + DBL_EPSILON != 1.0.
DBL_MANT_DIG

Synopsis

```c
#define DBL_MANT_DIG 53
```

Description

DBL_MANT_DIG specifies the number of base FLT_RADIX digits in the mantissa part of a double.
DBL_MAX

Synopsis

```c
#define  DBL_MAX    1.7976931348623157E+308
```

Description

DBL_MAX is the maximum value of a double.
DBL_MAX_10_EXP

Synopsis

```c
#define DBL_MAX_10_EXP +308
```

Description

DBL_MAX_10_EXP is the maximum value in base 10 of the exponent part of a `double`. 
DBL_MAX_EXP

Synopsis

```c
#define DBL_MAX_EXP +1024
```

Description

DBL_MAX_EXP is the maximum value of base `FLT_RADIX` in the exponent part of a `double`. 
DBL_MIN

Synopsis

#define  DBL_MIN  2.2250738585072014E-308

Description

DBL_MIN is the minimum value of a double.
DBL_MIN_10_EXP

Synopsis

#define DBL_MIN_10_EXP -307

Description

DBL_MIN_10_EXP is the minimum value in base 10 of the exponent part of a double.
**DBL_MIN_EXP**

**Synopsis**

```c
#define DBL_MIN_EXP -1021
```

**Description**

DBL_MIN_EXP is the minimum value of base FLT_RADIX in the exponent part of a double.
DECIMAL_DIG

Synopsis

#define DECIMAL_DIG 17

Description

DECIMAL_DIG specifies the number of decimal digits that can be rounded to a floating-point number without change to the value.
FLT_DIG

Synopsis

#define FLT_DIG 6

Description

FLT_DIG specifies the number of digits of precision of a float.
FLT_EPSILON

Synopsis

#define FLT_EPSILON 1.19209290E-07F // decimal constant

Description

FLT_EPSILON the minimum positive number such that 1.0 + FLT_EPSILON != 1.0.
FLT_EVAL_METHOD

Synopsis

#define FLT_EVAL_METHOD 0

Description

FLT_EVAL_METHOD specifies that all operations and constants are evaluated to the range and precision of the type.
FLT_MANT_DIG

Synopsis

#define FLT_MANT_DIG 24

Description

FLT_MANT_DIG specifies the number of base FLT_RADIX digits in the mantissa part of a float.
FLT_MAX

Synopsis

```c
#define FLT_MAX 3.40282347E+38F
```

Description

FLT_MAX is the maximum value of a float.
FLT_MAX_10_EXP

Synopsis

#define FLT_MAX_10_EXP +38

Description

FLT_MAX_10_EXP is the maximum value in base 10 of the exponent part of a float.
**FLT_MAX_EXP**

**Synopsis**

```c
#define FLT_MAX_EXP +128
```

**Description**

FLT_MAX_EXP is the maximum value of base `FLT_RADIX` in the exponent part of a `float`. 
FLT_MIN

Synopsis

#define  FLT_MIN     1.17549435E-38F

Description

FLT_MIN is the minimum value of a float.
FLT_MIN_10_EXP

Synopsis

#define FLT_MIN_10_EXP -37

Description

FLT_MIN_10_EXP is the minimum value in base 10 of the exponent part of a float.
FLT_MIN_EXP

Synopsis

#define FLT_MIN_EXP -125

Description

FLT_MIN_EXP is the minimum value of base FLT_RADIX in the exponent part of a float.
FLT_RADIX

Synopsis

```c
#define FLT_RADIX 2
```

Description

FLT_RADIX specifies the radix of the exponent representation.
FLT_ROUNDS

Synopsis

```c
#define FLT_ROUNDS 1
```

Description

FLT_ROUNDS specifies the rounding mode of floating-point addition is round to nearest.
<iso646.h>

Overview

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.

API Summary

<table>
<thead>
<tr>
<th>Macros</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>Alternative spelling for logical and operator</td>
</tr>
<tr>
<td>and_eq</td>
<td>Alternative spelling for logical and-equals operator</td>
</tr>
<tr>
<td>bitand</td>
<td>Alternative spelling for bitwise and operator</td>
</tr>
<tr>
<td>bitor</td>
<td>Alternative spelling for bitwise or operator</td>
</tr>
<tr>
<td>compl</td>
<td>Alternative spelling for bitwise complement operator</td>
</tr>
<tr>
<td>not</td>
<td>Alternative spelling for logical not operator</td>
</tr>
<tr>
<td>not_eq</td>
<td>Alternative spelling for not-equal operator</td>
</tr>
<tr>
<td>or</td>
<td>Alternative spelling for logical or operator</td>
</tr>
<tr>
<td>or_eq</td>
<td>Alternative spelling for bitwise or-equals operator</td>
</tr>
<tr>
<td>xor</td>
<td>Alternative spelling for bitwise exclusive or operator</td>
</tr>
<tr>
<td>xor_eq</td>
<td>Alternative spelling for bitwise exclusive-or-equals operator</td>
</tr>
</tbody>
</table>
and

Synopsis

#define and &&

Description

and defines the alternative spelling for &&.
and_eq

Synopsis

#define and_eq &=

Description

and_eq defines the alternative spelling for &=.
**bitand**

**Synopsis**

```
#define bitand &
```

**Description**

`bitand` defines the alternative spelling for `&`. 
bitor

Synopsis

#define bitor |

Description

bitor defines the alternative spelling for |.
compl

Synopsis

```c
#define compl ~
```

Description

`compl` defines the alternative spelling for `~`. 
not

Synopsis

#define not !

Description

not defines the alternative spelling for !.
not_eq

Synopsis

#define not_eq !=

Description

not_eq defines the alternative spelling for !=
or

Synopsis

```
#define or ||
```

Description

`or` defines the alternative spelling for `||`. 
or_eq

Synopsis

```c
#define or_eq |=
```

Description

`or_eq` defines the alternative spelling for `|=`. 
xor

Synopsis

```c
#define xor ^
```

Description

`xor` defines the alternative spelling for `^`. 
xor_eq

Synopsis

#define xor_eq ^=

Description

xor_eq defines the alternative spelling for ^=.
### API Summary

<table>
<thead>
<tr>
<th>Category</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long integer minimum and maximum values</td>
<td>LONG_MAX</td>
<td>Maximum value of a long integer</td>
</tr>
<tr>
<td></td>
<td>LONG_MIN</td>
<td>Minimum value of a long integer</td>
</tr>
<tr>
<td></td>
<td>ULONG_MAX</td>
<td>Maximum value of an unsigned long integer</td>
</tr>
<tr>
<td>Character minimum and maximum values</td>
<td>CHAR_MAX</td>
<td>Maximum value of a plain character</td>
</tr>
<tr>
<td></td>
<td>CHAR_MIN</td>
<td>Minimum value of a plain character</td>
</tr>
<tr>
<td></td>
<td>SCHAR_MAX</td>
<td>Maximum value of a signed character</td>
</tr>
<tr>
<td></td>
<td>SCHAR_MIN</td>
<td>Minimum value of a signed character</td>
</tr>
<tr>
<td></td>
<td>UCHAR_MAX</td>
<td>Maximum value of an unsigned char</td>
</tr>
<tr>
<td>Long long integer minimum and maximum values</td>
<td>LLONG_MAX</td>
<td>Maximum value of a long long integer</td>
</tr>
<tr>
<td></td>
<td>LLONG_MIN</td>
<td>Minimum value of a long long integer</td>
</tr>
<tr>
<td></td>
<td>ULLONG_MAX</td>
<td>Maximum value of an unsigned long long integer</td>
</tr>
<tr>
<td>Short integer minimum and maximum values</td>
<td>SHRT_MAX</td>
<td>Maximum value of a short integer</td>
</tr>
<tr>
<td></td>
<td>SHRT_MIN</td>
<td>Minimum value of a short integer</td>
</tr>
<tr>
<td></td>
<td>USHRT_MAX</td>
<td>Maximum value of an unsigned short integer</td>
</tr>
<tr>
<td>Integer minimum and maximum values</td>
<td>INT_MAX</td>
<td>Maximum value of an integer</td>
</tr>
<tr>
<td></td>
<td>INT_MIN</td>
<td>Minimum value of an integer</td>
</tr>
<tr>
<td></td>
<td>UINT_MAX</td>
<td>Maximum value of an unsigned integer</td>
</tr>
<tr>
<td>Type sizes</td>
<td>CHAR_BIT</td>
<td>Number of bits in a character</td>
</tr>
<tr>
<td>Multi-byte values</td>
<td>MB_LEN_MAX</td>
<td>Maximum number of bytes in a multi-byte character</td>
</tr>
</tbody>
</table>
CHAR_BIT

Synopsis

#define CHAR_BIT 8

Description

CHAR_BIT is the number of bits for smallest object that is not a bit-field (byte).
CHAR_MAX

Synopsis

#define CHAR_MAX 255

Description

CHAR_MAX is the maximum value for an object of type char.
CHAR_MIN

Synopsis

```c
#define CHAR_MIN 0
```

Description

CHAR_MIN is the minimum value for an object of type char.
INT_MAX

Synopsis

```
#define INT_MAX 2147483647
```

Description

INT_MAX is the maximum value for an object of type int.
**INT_MIN**

**Synopsis**

```c
#define INT_MIN (-2147483647 - 1)
```

**Description**

`INT_MIN` is the minimum value for an object of type `int`. 
LLONG_MAX

Synopsis

```
#define LLONG_MAX 9223372036854775807LL
```

Description

LLONG_MAX is the maximum value for an object of type long long int.
LLONG_MIN

Synopsis

#define LLONG_MIN (-9223372036854775807LL - 1)

Description

LLONG_MIN is the minimum value for an object of type long long int.
LONG_MAX

Synopsis

#define LONG_MAX 2147483647L

Description

LONG_MAX is the maximum value for an object of type long int.
LONG_MIN

Synopsis

#define LONG_MIN (-2147483647L - 1)

Description

LONG_MIN is the minimum value for an object of type long int.
## MB_LEN_MAX

### Synopsis

```
#define MB_LEN_MAX  4
```

### Description

`MB_LEN_MAX` is the maximum number of bytes in a multi-byte character for any supported locale. Unicode (ISO 10646) characters between 0 and 10FFFF inclusive are supported which convert to a maximum of four bytes in the UTF-8 encoding.
SCHAR_MAX

Synopsis

```
#define SCHAR_MAX 127
```  

Description

SCHAR_MAX is the maximum value for an object of type signed char.
SCHAR_MIN

Synopsis

```c
#define SCHAR_MIN (-128)
```

Description

SCHAR_MIN is the minimum value for an object of type `signed char`. 
**SHRT_MAX**

**Synopsis**

```c
#define SHRT_MAX 32767
```

**Description**

`SHRT_MAX` is the maximum value for an object of type `short int`. 
**SHRT_MIN**

**Synopsis**

```
#define SHRT_MIN (-32767 - 1)
```

**Description**

`SHRT_MIN` is the minimum value for an object of type `short int`. 
UCHAR_MAX

Synopsis

#define UCHAR_MAX 255

Description

UCHAR_MAX is the maximum value for an object of type unsigned char.
UINT_MAX

Synopsis

```c
#define UINT_MAX 4294967295U
```

Description

`UINT_MAX` is the maximum value for an object of type `unsigned int`. 
ULLONG_MAX

Synopsis

```c
#define ULLONG_MAX 18446744073709551615ULL
```

Description

ULLONG_MAX is the maximum value for an object of type unsigned long long int.
ULONG_MAX

Synopsis

#define  ULONG_MAX  4294967295UL

Description

ULONG_MAX is the maximum value for an object of type unsigned long int.
**USHRT_MAX**

**Synopsis**

```c
#define USHRT_MAX 65535
```

**Description**

`USHRT_MAX` is the maximum value for an object of type `unsigned short int`. 
<locale.h>

API Summary

<table>
<thead>
<tr>
<th>Structures</th>
<th>Formatting info for numeric values</th>
</tr>
</thead>
<tbody>
<tr>
<td>lconv</td>
<td>Formatting info for numeric values</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>localeconv</td>
<td>Get current locale data</td>
</tr>
<tr>
<td>setlocale</td>
<td>Set Locale</td>
</tr>
</tbody>
</table>
**lconv**

**Synopsis**

```c
typedef struct {
    char *decimal_point;
    char *thousands_sep;
    char *grouping;
    char *int_curr_symbol;
    char *currency_symbol;
    char *mon_decimal_point;
    char *mon_thousands_sep;
    char *mon_grouping;
    char *positive_sign;
    char *negative_sign;
    char int_frac_digits;
    char frac_digits;
    char p_cs_precedes;
    char p_sep_by_space;
    char n_cs_precedes;
    char n_sep_by_space;
    char p_sign_posn;
    char n_sign_posn;
    char int_p_cs_precedes;
    char int_n_cs_precedes;
    char int_p_sep_by_space;
    char int_n_sep_by_space;
    char int_p_sign_posn;
    char int_n_sign_posn;
} lconv;
```

**Description**

The `lconv` structure holds formatting information on how numeric values are to be written. Note that the order of fields in this structure is not consistent between implementations, nor is it consistent between C89 and C99 standards.

The members `decimal_point`, `grouping`, and `thousands_sep` are controlled by `LC_NUMERIC`, the remainder by `LC_MONETARY`.

The members `int_n_cs_precedes`, `int_n_sep_by_space`, `int_n_sign_posn`, `int_p_cs_precedes`, `int_p_sep_by_space`, and `int_p_sign_posn` are added by the C99 standard.

We have standardized on the ordering specified by the ARM EABI for the base of this structure. This ordering is neither that of C89 nor C99.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>currency_symbol</td>
<td>Local currency symbol.</td>
</tr>
<tr>
<td>decimal_point</td>
<td>Decimal point separator.</td>
</tr>
<tr>
<td>frac_digits</td>
<td>Amount of fractional digits to the right of the decimal point for monetary quantities in the local format.</td>
</tr>
<tr>
<td><strong>grouping</strong></td>
<td>Specifies the amount of digits that form each of the groups to be separated by thousands_sep separator for non-monetary quantities.</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>int_curr_symbol</strong></td>
<td>International currency symbol.</td>
</tr>
<tr>
<td><strong>int_frac_digits</strong></td>
<td>Amount of fractional digits to the right of the decimal point for monetary quantities in the international format.</td>
</tr>
<tr>
<td><strong>mon_decimal_point</strong></td>
<td>Decimal-point separator used for monetary quantities.</td>
</tr>
<tr>
<td><strong>mon_grouping</strong></td>
<td>Specifies the amount of digits that form each of the groups to be separated by mon_thousands_sep separator for monetary quantities.</td>
</tr>
<tr>
<td><strong>mon_thousands_sep</strong></td>
<td>Separators used to delimit groups of digits to the left of the decimal point for monetary quantities.</td>
</tr>
<tr>
<td><strong>negative_sign</strong></td>
<td>Sign to be used for negative monetary quantities.</td>
</tr>
<tr>
<td><strong>n_cs_precedes</strong></td>
<td>Whether the currency symbol should precede negative monetary quantities.</td>
</tr>
<tr>
<td><strong>n_sep_by_space</strong></td>
<td>Whether a space should appear between the currency symbol and negative monetary quantities.</td>
</tr>
<tr>
<td><strong>n_sign_posn</strong></td>
<td>Position of the sign for negative monetary quantities.</td>
</tr>
<tr>
<td><strong>positive_sign</strong></td>
<td>Sign to be used for nonnegative (positive or zero) monetary quantities.</td>
</tr>
<tr>
<td><strong>p_cs_precedes</strong></td>
<td>Whether the currency symbol should precede nonnegative (positive or zero) monetary quantities.</td>
</tr>
<tr>
<td><strong>p_sep_by_space</strong></td>
<td>Whether a space should appear between the currency symbol and nonnegative (positive or zero) monetary quantities.</td>
</tr>
<tr>
<td><strong>p_sign_posn</strong></td>
<td>Position of the sign for nonnegative (positive or zero) monetary quantities.</td>
</tr>
<tr>
<td><strong>thousands_sep</strong></td>
<td>Separators used to delimit groups of digits to the left of the decimal point for non-monetary quantities.</td>
</tr>
</tbody>
</table>
localeconv

Synopsis

```c
localeconv(void);
```

Description

`localeconv` returns a pointer to a structure of type `lconv` with the corresponding values for the current locale filled in.
setlocale

Synopsis

```c
char *setlocale(int category,
                const char *locale);
```

Description

`setlocale` sets the current locale. The `category` parameter can have the following values:

<table>
<thead>
<tr>
<th>Name</th>
<th>Locale affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>LC_ALL</td>
<td>Entire locale</td>
</tr>
<tr>
<td>LC_CCollATE</td>
<td>Affects <code>strcoll</code> and <code>strxfrm</code></td>
</tr>
<tr>
<td>LC_CTYPE</td>
<td>Affects character handling</td>
</tr>
<tr>
<td>LC_MONETARY</td>
<td>Affects monetary formatting information</td>
</tr>
<tr>
<td>LC_NUMERIC</td>
<td>Affects decimal-point character in I/O and string formatting operations</td>
</tr>
<tr>
<td>LC_TIME</td>
<td>Affects <code>strftime</code></td>
</tr>
</tbody>
</table>

The `locale` parameter contains the name of a C locale to set or if `NULL` is passed the current locale is not changed.

Return Value

`setlocale` returns the name of the current locale.
### API Summary

#### Comparison Macros

- `isgreater`  
  Is greater

- `isgreaterequal`  
  Is greater or equal

- `isless`  
  Is less

- `islessequal`  
  Is less or equal

- `islessgreater`  
  Is less or greater

- `isunordered`  
  Is unordered

#### Classification Macros

- `fpclassify`  
  Classify floating type

- `isfinite`  
  Test for a finite value

- `isinf`  
  Test for infinity

- `isnan`  
  Test for NaN

- `isnormal`  
  Test for a normal value

- `signbit`  
  Test sign

#### Trigonometric functions

- `cos`  
  Compute cosine of a double

- `cosf`  
  Compute cosine of a float

- `sin`  
  Compute sine of a double

- `sinf`  
  Compute sine of a float

- `tan`  
  Compute tangent of a double

- `tanf`  
  Compute tangent of a double

#### Inverse trigonometric functions

- `acos`  
  Compute inverse cosine of a double

- `acosf`  
  Compute inverse cosine of a float

- `asin`  
  Compute inverse sine of a double

- `asinf`  
  Compute inverse sine of a float

- `atan`  
  Compute inverse tangent of a double

- `atan2`  
  Compute inverse tangent of a ratio of doubles

- `atan2f`  
  Compute inverse tangent of a ratio of floats

- `atanf`  
  Compute inverse tangent of a float

#### Exponential and logarithmic functions
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>exp</code></td>
<td>Compute exponential of a double</td>
</tr>
<tr>
<td><code>exp2</code></td>
<td>Compute binary exponential of a double</td>
</tr>
<tr>
<td><code>exp2f</code></td>
<td>Compute binary exponential of a float</td>
</tr>
<tr>
<td><code>expf</code></td>
<td>Compute exponential of a float</td>
</tr>
<tr>
<td><code>expm1</code></td>
<td>Compute exponential minus one of a double</td>
</tr>
<tr>
<td><code>expm1f</code></td>
<td>Compute exponential minus one of a float</td>
</tr>
<tr>
<td><code>frexp</code></td>
<td>Set exponent of a double</td>
</tr>
<tr>
<td><code>frexpf</code></td>
<td>Set exponent of a float</td>
</tr>
<tr>
<td><code>ilogb</code></td>
<td>Compute integer binary logarithm of a double</td>
</tr>
<tr>
<td><code>ilogbf</code></td>
<td>Compute integer binary logarithm of a float</td>
</tr>
<tr>
<td><code>ldexp</code></td>
<td>Adjust exponent of a double</td>
</tr>
<tr>
<td><code>ldexpf</code></td>
<td>Adjust exponent of a float</td>
</tr>
<tr>
<td><code>log</code></td>
<td>Compute natural logarithm of a double</td>
</tr>
<tr>
<td><code>log10</code></td>
<td>Compute common logarithm of a double</td>
</tr>
<tr>
<td><code>log10f</code></td>
<td>Compute common logarithm of a float</td>
</tr>
<tr>
<td><code>log1p</code></td>
<td>Compute natural logarithm plus one of a double</td>
</tr>
<tr>
<td><code>log1pf</code></td>
<td>Compute natural logarithm plus one of a float</td>
</tr>
<tr>
<td><code>log2</code></td>
<td>Compute binary logarithm of a double</td>
</tr>
<tr>
<td><code>log2f</code></td>
<td>Compute binary logarithm of a float</td>
</tr>
<tr>
<td><code>logb</code></td>
<td>Compute floating-point base logarithm of a double</td>
</tr>
<tr>
<td><code>logbf</code></td>
<td>Compute floating-point base logarithm of a float</td>
</tr>
<tr>
<td><code>logf</code></td>
<td>Compute natural logarithm of a float</td>
</tr>
<tr>
<td><code>scalbln</code></td>
<td>Scale a double</td>
</tr>
<tr>
<td><code>scalblnf</code></td>
<td>Scale a float</td>
</tr>
<tr>
<td><code>scalbn</code></td>
<td>Scale a double</td>
</tr>
<tr>
<td><code>scalbnf</code></td>
<td>Scale a float</td>
</tr>
</tbody>
</table>

**Rounding and remainder functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ceil</code></td>
<td>Compute smallest integer not greater than a double</td>
</tr>
<tr>
<td><code>ceilf</code></td>
<td>Compute smallest integer not greater than a float</td>
</tr>
<tr>
<td><code>floor</code></td>
<td>Compute largest integer not greater than a double</td>
</tr>
<tr>
<td><code>floorf</code></td>
<td>Compute largest integer not greater than a float</td>
</tr>
<tr>
<td><code>fmod</code></td>
<td>Compute remainder after division of two doubles</td>
</tr>
<tr>
<td><code>fmodf</code></td>
<td>Compute remainder after division of two floats</td>
</tr>
<tr>
<td><code>llrint</code></td>
<td>Round and cast double to long long</td>
</tr>
<tr>
<td><code>llrintf</code></td>
<td>Round and cast float to long long</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>llround</td>
<td>Round and cast double to long long</td>
</tr>
<tr>
<td>llroundf</td>
<td>Round and cast float to long long</td>
</tr>
<tr>
<td>lrint</td>
<td>Round and cast double to long</td>
</tr>
<tr>
<td>lrintf</td>
<td>Round and cast float to long</td>
</tr>
<tr>
<td>lround</td>
<td>Round and cast double to long</td>
</tr>
<tr>
<td>lroundf</td>
<td>Round and cast float to long</td>
</tr>
<tr>
<td>modf</td>
<td>Break a double into integer and fractional parts</td>
</tr>
<tr>
<td>modff</td>
<td>Break a float into integer and fractional parts</td>
</tr>
<tr>
<td>nearbyint</td>
<td>Round double to nearby integral value</td>
</tr>
<tr>
<td>nearbyintf</td>
<td>Round float to nearby integral value</td>
</tr>
<tr>
<td>remainder</td>
<td>Compute remainder of a double</td>
</tr>
<tr>
<td>remainderf</td>
<td>Compute remainder of a float</td>
</tr>
<tr>
<td>remquo</td>
<td>Compute remainder and quotient of a double</td>
</tr>
<tr>
<td>remquof</td>
<td>Compute remainder and quotient of a float</td>
</tr>
<tr>
<td>rint</td>
<td>Round a double to an integral value</td>
</tr>
<tr>
<td>rintf</td>
<td>Round a float to an integral value</td>
</tr>
<tr>
<td>round</td>
<td>Round a double to the nearest integral value</td>
</tr>
<tr>
<td>roundf</td>
<td>Round a float to the nearest integral value</td>
</tr>
<tr>
<td>trunc</td>
<td>Truncate a double value</td>
</tr>
<tr>
<td>truncf</td>
<td>Truncate a float value</td>
</tr>
</tbody>
</table>

**Power functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cbrt</td>
<td>Compute cube root of a double</td>
</tr>
<tr>
<td>cbrtf</td>
<td>Compute cube root of a float</td>
</tr>
<tr>
<td>hypot</td>
<td>Compute complex magnitude of two doubles</td>
</tr>
<tr>
<td>hypotf</td>
<td>Compute complex magnitude of two floats</td>
</tr>
<tr>
<td>pow</td>
<td>Raise a double to a power</td>
</tr>
<tr>
<td>powf</td>
<td>Raise a float to a power</td>
</tr>
<tr>
<td>sqrt</td>
<td>Compute square root of a double</td>
</tr>
<tr>
<td>sqrtf</td>
<td>Compute square root of a float</td>
</tr>
</tbody>
</table>

**Absolute value functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fabs</td>
<td>Compute absolute value of a double</td>
</tr>
<tr>
<td>fabsf</td>
<td>Compute absolute value of a float</td>
</tr>
</tbody>
</table>

**Maximum, minimum, and positive difference functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fdim</td>
<td>Compute positive difference of two doubles</td>
</tr>
<tr>
<td>fdimf</td>
<td>Compute positive difference of two floats</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td><strong>fmax</strong></td>
<td>Compute maximum of two doubles</td>
</tr>
<tr>
<td><strong>fmaxf</strong></td>
<td>Compute maximum of two floats</td>
</tr>
<tr>
<td><strong>fmin</strong></td>
<td>Compute minimum of two doubles</td>
</tr>
<tr>
<td><strong>fminf</strong></td>
<td>Compute minimum of two floats</td>
</tr>
<tr>
<td><strong>Hyperbolic functions</strong></td>
<td></td>
</tr>
<tr>
<td><strong>cosh</strong></td>
<td>Compute hyperbolic cosine of a double</td>
</tr>
<tr>
<td><strong>coshf</strong></td>
<td>Compute hyperbolic cosine of a float</td>
</tr>
<tr>
<td><strong>sinh</strong></td>
<td>Compute hyperbolic sine of a double</td>
</tr>
<tr>
<td><strong>sinhf</strong></td>
<td>Compute hyperbolic sine of a float</td>
</tr>
<tr>
<td><strong>tanh</strong></td>
<td>Compute hyperbolic tangent of a double</td>
</tr>
<tr>
<td><strong>tanhf</strong></td>
<td>Compute hyperbolic tangent of a float</td>
</tr>
<tr>
<td><strong>Inverse hyperbolic functions</strong></td>
<td></td>
</tr>
<tr>
<td><strong>acosh</strong></td>
<td>Compute inverse hyperbolic cosine of a double</td>
</tr>
<tr>
<td><strong>acoshf</strong></td>
<td>Compute inverse hyperbolic cosine of a float</td>
</tr>
<tr>
<td><strong>asinh</strong></td>
<td>Compute inverse hyperbolic sine of a double</td>
</tr>
<tr>
<td><strong>asinhf</strong></td>
<td>Compute inverse hyperbolic sine of a float</td>
</tr>
<tr>
<td><strong>atanh</strong></td>
<td>Compute inverse hyperbolic tangent of a double</td>
</tr>
<tr>
<td><strong>atanhf</strong></td>
<td>Compute inverse hyperbolic tangent of a float</td>
</tr>
<tr>
<td><strong>Fused multiply functions</strong></td>
<td></td>
</tr>
<tr>
<td><strong>fma</strong></td>
<td>Compute fused multiply-add of doubles</td>
</tr>
<tr>
<td><strong>fmaf</strong></td>
<td>Compute fused multiply-add of floats</td>
</tr>
<tr>
<td><strong>Floating-point manipulation functions</strong></td>
<td></td>
</tr>
<tr>
<td><strong>copysign</strong></td>
<td>Copy magnitude and sign of a double</td>
</tr>
<tr>
<td><strong>copysignf</strong></td>
<td>Copy magnitude and sign of a float</td>
</tr>
<tr>
<td><strong>nextafter</strong></td>
<td>Next representable double value</td>
</tr>
<tr>
<td><strong>nextafterf</strong></td>
<td>Next representable float value</td>
</tr>
<tr>
<td><strong>Error and Gamma functions</strong></td>
<td></td>
</tr>
<tr>
<td><strong>erf</strong></td>
<td>Compute error function of a double</td>
</tr>
<tr>
<td><strong>erfc</strong></td>
<td>Compute complementary error function of a double</td>
</tr>
<tr>
<td><strong>erfcf</strong></td>
<td>Compute complementary error function of a float</td>
</tr>
<tr>
<td><strong>erff</strong></td>
<td>Compute error function of a float</td>
</tr>
<tr>
<td><strong>lgamma</strong></td>
<td>Compute log-gamma function of a double</td>
</tr>
<tr>
<td><strong>lgammaf</strong></td>
<td>Compute log-gamma function of a float</td>
</tr>
<tr>
<td><strong>tgamma</strong></td>
<td>Compute gamma function of a double</td>
</tr>
<tr>
<td><strong>tgammaf</strong></td>
<td>Compute gamma function of a float</td>
</tr>
</tbody>
</table>
acos

Synopsis

double acos(double x);

Description

acos returns the principal value, in radians, of the inverse circular cosine of x. The principal value lies in the interval \([0, \pi]\) radians.

If \(|x| > 1\), errno is set to EDOM and acos returns HUGE_VAL.

If x is NaN, acos returns x. If \(|x| > 1\), acos returns NaN.
acosf

Synopsis

```c
float acosf(float x);
```

Description

acosf returns the principal value, in radians, of the inverse circular cosine of x. The principal value lies in the interval [0, PI] radians.

If |x| > 1, errno is set to EDOM and acosf returns HUGE_VAL.

If x is NaN, acosf returns x. If |x| > 1, acosf returns NaN.
acosh

Synopsis

```c
double acosh(double x);
```

Description

acosh returns the non-negative inverse hyperbolic cosine of \( x \).

acosh(\( x \)) is defined as \( \log(x + \sqrt{x^2 - 1}) \), assuming completely accurate computation.

If \( x < 1 \), errno is set to EDOM and acosh returns HUGE_VAL.

If \( x < 1 \), acosh returns NaN.
If \( x \) is NaN, acosh returns NaN.
acoshf

Synopsis

```c
float acoshf(float x);
```

Description

acoshf returns the non-negative inverse hyperbolic cosine of x.

acosh(x) is defined as \( \log(x + \sqrt{x^2 - 1}) \), assuming completely accurate computation.

If \( x < 1 \), errno is set to EDOM and acoshf returns HUGE_VALF.

If \( x < 1 \), acoshf returns NaN.
If \( x \) is NaN, acoshf returns that NaN.
**asin**

**Synopsis**

```c
double asin(double x);
```

**Description**

asin returns the principal value, in radians, of the inverse circular sine of x. The principal value lies in the interval $[-\pi, +\pi]$ radians.

If $|x| > 1$, *errno* is set to **EDOM** and asin returns **HUGE_VAL**.

If x is NaN, asin returns x. If $|x| > 1$, asin returns NaN.
asinf

Synopsis

```c
float asinf(float x);
```

Description

asinf returns the principal value, in radians, of the inverse circular sine of val. The principal value lies in the
interval \([-\pi, +\pi]\) radians.

If |x| > 1, \texttt{errno} is set to \texttt{EDOM} and asinf returns \texttt{HUGE_VALF}.

If x is NaN, asinf returns x. If |x| > 1, asinf returns NaN.
asinh

Synopsis

```c
double asinh(double x);
```

Description

asinh calculates the hyperbolic sine of x.

If |x| > ~709.782, errno is set to EDOM and asinh returns HUGE_VAL.

If x is +, , or NaN, asinh returns |x|. If |x| > ~709.782, asinh returns + or depending upon the sign of x.
asinhf

Synopsis

```c
float asinhf(float x);
```

Description

`asinhf` calculates the hyperbolic sine of `x`.

If \(|x| \approx 88.7228\), `errno` set to `EDOM` and `asinhf` returns `HUGE_VALF`.

If `x` is +, - or NaN, `asinhf` returns \(|x|\). If \(|x| \approx 88.7228\), `asinhf` returns + or depending upon the sign of `x`. 
atan

Synopsis

double atan(double x);

Description

atan returns the principal value, in radians, of the inverse circular tangent of x. The principal value lies in the interval \([-\frac{\pi}{2}, +\frac{\pi}{2}\]) radians.
atan2

Synopsis

```c
double atan2(double y, double x);
```

Description

atan2 returns the value, in radians, of the inverse circular tangent of y divided by x using the signs of x and y to compute the quadrant of the return value. The principal value lies in the interval [-\pi, +\pi] radians. If x = y = 0, errno is set to EDOM and atan2 returns HUGE_VAL.

atan2(x, NaN) is NaN.
atan2(NaN, x) is NaN.
atan2(0, +(anything but NaN)) is 0.
atan2(0, (anything but NaN)) is +\pi.
atan2((anything but 0 and NaN), 0) is -\pi.
atan2((anything but and NaN), +) is 0.
atan2((anything but and NaN), ) is +\pi.
atan2(, +) is -\pi.
atan2(, ) is 0.
atan2(, (anything but 0, NaN, and )) is .
atan2f

Synopsis

```c
float atan2f(float y,
            float x);
```

Description

atan2f returns the value, in radians, of the inverse circular tangent of y divided by x using the signs of x and y to compute the quadrant of the return value. The principal value lies in the interval [−π, +π] radians.

If x = y = 0, errno is set to EDOM and atan2f returns HUGE_VALF.

atan2f(x, NaN) is NaN.
atan2f(NaN, x) is NaN.
atan2f(0, +(anything but NaN)) is 0.
atan2f(0, (anything but NaN)) is ±π.
atan2f((anything but 0 and NaN), 0) is ±π.
atan2f((anything but 0 and NaN), +) is 0.
atan2f((anything but 0 and NaN), −) is ±π.
atan2f(, +) is +π.
atan2f(, −) is −π.
atan2f(, (anything but 0, NaN, and )) is ±π.
atanf

Synopsis

```c
float atanf(float x);
```

Description

atanf returns the principal value, in radians, of the inverse circular tangent of x. The principal value lies in the interval \([-\frac{\pi}{2}, \frac{\pi}{2}]\) radians.
atanh

Synopsis

```c
double atanh(double x);
```

Description

atanh returns the inverse hyperbolic tangent of x.

If |x| > 1, errno is set to EDOM and atanh returns HUGE_VAL.

If |x| > 1 atanh returns NaN.
If x is NaN, atanh returns that NaN.
If x is 1, atanh returns -
If x is 1, atanh returns +
atanhf

Synopsis

```c
float atanhf(float x);
```

Description

atanhf returns the inverse hyperbolic tangent of x.

If |x| > 1 atanhf returns NaN. If x is NaN, atanhf returns that NaN. If x is 1, atanhf returns -Inf. If x is 1, atanhf returns Inf.
cbrt

Synopsis

double cbrt(double x);

Description

cbrt computes the cube root of x.
cbrtf

Synopsis

```c
float cbrtf(float x);
```

Description

`cbrtf` computes the cube root of `x`. 
cei1

Synopsis

double ceil(double x);

Description

ceil computes the smallest integer value not less than x.

ceil (0) is 0. ceil ( ) is .
**ceilf**

**Synopsis**

```plaintext
float ceilf(float x);
```

**Description**

`ceilf` computes the smallest integer value not less than `x`.

`ceilf(0)` is 0. `ceilf()` is .
copysign

Synopsis

```c
double copysign(double x, double y);
```

Description

copysign returns a value with the magnitude of x and the sign of y.
copysignf

Synopsis

```c
float copysignf(float x,
float y);
```

Description

copysignf returns a value with the magnitude of x and the sign of y.
COS

Synopsis

```c
double cos(double x);
```

Description

`cos` returns the radian circular cosine of `x`.

If `|x| > 10^9`, `errno` is set to `EDOM` and `cos` returns `HUGE_VAL`.

If `x` is `NaN`, `cos` returns `x`. If `|x|` is ``, `cos` returns `NaN`.
cosf

Synopsis

```c
float cosf(float x);
```

Description

`cosf` returns the radian circular cosine of `x`.

If `|x| > 10^9`, `errno` is set to `EDOM` and `cosf` returns `HUGE_VALF`.

If `x` is NaN, `cosf` returns `x`. If `|x|` is NaN, `cosf` returns NaN.
cosh

Synopsis

double cosh(double x);

Description

cosh calculates the hyperbolic cosine of x.

If |x| > ~709.782, errno is set to EDOM and cosh returns HUGE_VAL.

If x is +, , or NaN, cosh returns |x|. If |x| > ~709.782, cosh returns + or depending upon the sign of x.
coshf

Synopsis

```c
float coshf(float x);
```

Description

coshf calculates the hyperbolic sine of x.

If \(|x| > \sim 88.7228\), errno is set to EDOM and coshf returns HUGE_VALF.

If x is +, , or NaN, coshf returns \(|x|\).

If \(|x| > \sim 88.7228\), coshf returns + or depending upon the sign of x.
erf

Synopsis

double erf(double x);

Description

erf returns the error function for x.
erfc

Synopsis

double erfc(double x);

Description

erfc returns the complementary error function for x.
erfcf

Synopsis

```c
float erfcf(float x);
```

Description

erfcf returns the complementary error function for x.

erff

Synopsis

```c
float erff(float x);
```

Description

erff returns the error function for x.
exp

Synopsis

```c
double exp(double x);
```

Description

exp computes the base-e exponential of x.

If |x| > ~709.782, errno is set to EDOM and exp returns HUGE_VAL.

If x is NaN, exp returns NaN.
If x is , exp returns .
If x is , exp returns 0.
exp2

Synopsis

```c
double exp2(double x);
```

Description

exp2 returns 2 raised to the power of x.
exp2f

Synopsis

```c
float exp2f(float x);
```

Description

`exp2f` returns 2 raised to the power of `x`. 
expf

Synopsis

```c
float expf(float x);
```

Description

`expf` computes the base-e exponential of `x`.

If \(|x| \geq 88.722\), `errno` is set to `EDOM` and `expf` returns `HUGE_VALF`. If `x` is NaN, `expf` returns NaN.

If `x` is -\(\infty\), `expf` returns 0.

If `x` is +\(\infty\), `expf` returns 0.
expm1

Synopsis

```c
double expm1(double x);
```

Description

expm1 returns $e$ raised to the power of $x$ minus one.
expm1f

Synopsis

```c
float expm1f(float x);
```

Description

`expm1f` returns e raised to the power of x minus one.
fabs

Synopsis

```c
double fabs(double x);
```
**fabsf**

**Synopsis**

```c
float fabsf(float x);
```

**Description**

fabsf computes the absolute value of the floating-point number x.
fdim

Synopsis

```c
double fdim(double x, double y);
```

Description

`fdim` returns the positive difference between `x` and `y`. 
**fdimf**

**Synopsis**

```c
float fdimf(float x, float y);
```

**Description**

*fdimf* returns the positive difference between *x* and *y*.
floor

Synopsis

```c
double floor(double);
```

floor computes the largest integer value not greater than x.

floor(0) is 0. floor() is .
floorf

Synopsis

```c
float floorf(float);
```

`floorf` computes the largest integer value not greater than x.

`floorf(0)` is 0. `floorf()` is 0.
fma

Synopsis

```c
double fma(double x,
            double y,
            double z);
```

Description

fma computes \(x y + z\) with a single rounding.
fmaf

Synopsis

```c
float fmaf(float x, float y, float z);
```

Description

fmaf computes \( x \cdot y + z \) with a single rounding.
fmax

Synopsis

```c
double fmax(double x,
            double y);
```

Description

fmax determines the maximum of x and y.

fmax (NaN, y) is y. fmax (x, NaN) is x.
fmaxf

Synopsis

```c
float fmaxf(float x,
            float y);
```

Description

fmaxf determines the maximum of x and y.

fmaxf (NaN, y) is y. fmaxf(x, NaN) is x.
fmin

Synopsis

```c
double fmin(double x,
            double y);
```

Description

fmin determines the minimum of x and y.

fmin (NaN, y) is y. fmin (x, NaN) is x.
fminf

Synopsis

```c
float fminf(float x,
            float y);
```

Description

`fminf` determines the minimum of `x` and `y`.

`fminf` (NaN, `y`) is `y`. `fminf` (x, NaN) is `x`. 
fmod

Synopsis

```c
double fmod(double x,
            double y);
```

Description

fmod computes the floating-point remainder of x divided by y. This returns the value $x \mod y$, for some integer $n$ such that, if $y$ is nonzero, the result has the same sign as $x$ and magnitude less than the magnitude of $y$.

fmod (NaN, $y$) is NaN. fmod ($x$, NaN) is NaN. fmod (0, $y$) is 0 for $y$ not zero.

fmod ($x$, 0) is NaN.

fmod ($x$, ) is $x$ for $x$ not infinite.
fmodf

Synopsis

```c
float fmodf(float x,
    float y);
```

Description

fmodf computes the floating-point remainder of x divided by y. fmodf returns the value x n y, for some integer n such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

fmodf (NaN, y) is NaN. fmodf (x, NaN) is NaN. fmodf (0, y) is 0 for y not zero.

fmodf (-y, y) is NaN.

fmodf (x, 0) is NaN.

fmodf (x, ) is x for x not infinite.
fpclassify

Synopsis

```
#define fpclassify(x) (__is_float32(x) ? __float32_classify(x) : __float64_classify(x))
```

Description

fpclassify classifies x as NaN, infinite, normal, subnormal, zero, or into another implementation-defined category. fpclassify returns one of:

- FP_ZERO
- FP_SUBNORMAL
- FP_NORMAL
- FP_INFINITY
- FP_NAN
frexp

Synopsis

double frexp(double x,
    int *exp);

Description

frexp breaks a floating-point number into a normalized fraction and an integral power of 2.

frexp stores power of two in the int object pointed to by exp and returns the value x, such that x has a
magnitude in the interval [1/2, 1) or zero, and value equals x * 2^exp.

If x is zero, both parts of the result are zero.

If x is or NaN, frexp returns x and stores zero into the int object pointed to by exp.
frexp

Synopsis

```c
float frexpf(float x,
             int *exp);
```

Description

`frexp` breaks a floating-point number into a normalized fraction and an integral power of 2.

`frexp` stores power of two in the `int` object pointed to by `frexp` and returns the value `x`, such that `x` has a magnitude in the interval `[1, 1)` or zero, and value equals `x * 2^exp`.

If `x` is zero, both parts of the result are zero.

If `x` is or NaN, `frexp` returns `x` and stores zero into the `int` object pointed to by `exp`. 
hypot

Synopsis

```c
double hypot(double x, double y);
```

Description

`hypot` computes the square root of the sum of the squares of `x` and `y`, `sqrt(x^2 + y^2)`, without undue overflow or underflow. If `x` and `y` are the lengths of the sides of a right-angled triangle, then `hypot` computes the length of the hypotenuse.

If `x` or `y` is ± or , `hypot` returns .
If `x` or `y` is NaN, `hypot` returns NaN.
**hypotf**

**Synopsis**

```c
float hypotf(float x, float y);
```

**Description**

`hypotf` computes the square root of the sum of the squares of `x` and `y`, `sqrtf(x*x + y*y)`, without undue overflow or underflow. If `x` and `y` are the lengths of the sides of a right-angled triangle, then `hypotf` computes the length of the hypotenuse.

If `x` or `y` is + or , `hypotf` returns . If `x` or `y` is NaN, `hypotf` returns NaN.
**ilogb**

**Synopsis**

```c
int ilogb(double x);
```

**Description**

`ilogb` returns the integral part of the logarithm of `x`, using `FLT_RADIX` as the base for the logarithm.
**ilogbf**

**Synopsis**

```c
int ilogbf(float x);
```

**Description**

`ilogbf` returns the integral part of the logarithm of `x`, using `FLT_RADIX` as the base for the logarithm.
isfinite

Synopsis

```c
#define isfinite(x) (sizeof(x) == sizeof(float) ? __float32_isfinite(x) : __float64_isfinite(x))
```

Description

`isfinite` determines whether `x` is a finite value (zero, subnormal, or normal, and not infinite or NaN). `isfinite` returns a non-zero value if and only if `x` has a finite value.
isgreater

Synopsis

```c
#define isgreater(x, y) (!isunordered(x, y) && (x > y))
```

Description

isgreater returns whether \( x \) is greater than \( y \).
isgreaterequal

Synopsis

```c
#define isgreaterequal(x, y) (!isunordered(x, y) && (x >= y))
```

Description

isgreaterequal returns whether x is greater than or equal to y.
isinf

Synopsis

```
#define isinf(x) (sizeof(x) == sizeof(float) ? __float32_isinf(x) : __float64_isinf(x))
```

Description

isinf determines whether x is an infinity (positive or negative). The determination is based on the type of the argument.
isless

Synopsis

```c
#define isless(x, y) (isunordered(x, y) && (x < y))
```

Description

isless returns whether x is less than y.
islessequal

Synopsis

```c
#define islessequal(x, y) (!isunordered(x, y) && (x <= y))
```

Description

islessequal returns whether x is less than or equal to y.
### islessgreater

**Synopsis**

```c
#define islessgreater(x, y) (!isunordered(x, y) && (x < y || x > y))
```

**Description**

`islessgreater` returns whether `x` is less than or greater than `y`. 
isnan

Synopsis

```c
#define isnan(x) (sizeof(x) == sizeof(float) ? __float32_isnan(x) : __float64_isnan(x))
```

Description

isnan determines whether x is a NaN. The determination is based on the type of the argument.
isnormal

Synopsis

#define isnormal(x) (sizeof(x) == sizeof(float) ? __float32_isnormal(x) : __float64_isnormal(x))

Description

isnormal determines whether x is a normal value (zero, subnormal, or normal, and not infinite or NaN). isnormal returns a non-zero value if and only if x has a normal value.
isunordered

Synopsis

```c
#define isunordered(a,b) (fpclassify(a) == FP_NAN || fpclassify(b) == FP_NAN)
```

Description

`isunordered` returns whether `x` or `y` are unordered values.
ldexp

Synopsis

```c
double ldexp(double x, int exp);
```

Description

`ldexp` multiplies a floating-point number by an integral power of 2.

`ldexp` returns \( x \times 2^{\exp} \).

If the result overflows, `errno` is set to `ERANGE` and `ldexp` returns `HUGE_VALF`.

If \( x \) is or NaN, `ldexp` returns \( x \). If the result overflows, `ldexp` returns .
ldexpf

Synopsis

```c
float ldexpf(float x,
             int exp);
```

Description

`ldexpf` multiplies a floating-point number by an integral power of 2.

`ldexpf` returns \( x \times 2^{\text{exp}} \). If the result overflows, `errno` is set to `ERANGE` and `ldexpf` returns `HUGE_VALF`.

If \( x \) is or NaN, `ldexpf` returns \( x \). If the result overflows, `ldexpf` returns .
Igamma

Synopsis

```c
double lgamma(double x);
```

Description

Igamma returns the natural logarithm of the gamma function for x.
Igammaf

Synopsis

float lgammaf(float x);

Description

Igammaf returns the natural logarithm of the gamma function for x.
llrint

Synopsis

```c
long long int llrint(double x);
```

Description

llrint rounds x to an integral value and returns it as a long long int.
**llrintf**

**Synopsis**

```c
long long int llrintf(float x);
```

**Description**

`llrintf` rounds `x` to an integral value and returns it as a long long int.
Ilround

Synopsis

```c
long long int llround(double x);
```

Description

Ilround rounds x to an integral value, with halfway cases rounded away from zero, and returns it as a long long int.
Ilroundf

Synopsis

```c
long long int llroundf(float x);
```

Description

Ilroundf rounds x to an integral value, with halfway cases rounded away from zero, and returns it as a long long int.
**log**

**Synopsis**

```c
double log(double x);
```

**Description**

`log` computes the base-e logarithm of `x`.

If `x = 0`, `errno` is set to `ERANGE` and `log` returns `HUGE_VAL`. If `x < 0`, `errno` is set to `EDOM` and `log` returns `HUGE_VAL`.

If `x < 0` or `x = -`, `log` returns `NaN`.
If `x = 0`, `log` returns `-`.
If `x = -`, `log` returns `-`.
If `x = NaN`, `log` returns `x`. 
log10

Synopsis

```c
double log10(double x);
```

Description

log10 computes the base-10 logarithm of x.

If x = 0, errno is set to ERANGE and log10 returns HUGE_VAL. If x < 0, errno is set to EDOM and log10 returns HUGE_VAL.

If x < 0 or x = , log10 returns NaN.
If x = 0, log10 returns .
If x = , log10 returns .
If x = NaN, log10 returns x.
log10f

Synopsis

```c
float log10f(float x);
```

Description

log10f computes the base-10 logarithm of x.

If x = 0, errno is set to ERANGE and log10f returns HUGE_VALF. If x < 0, errno is set to EDOM and log10f returns HUGE_VALF.

If x < 0 or x = , log10f returns NaN.
If x = 0, log10f returns .
If x = , log10f returns .
If x = NaN, log10f returns x.
**log1p**

**Synopsis**

```c
double log1p(double x);
```

**Description**

`log1p` computes the base-e logarithm of `x` plus one.
**log1pf**

**Synopsis**

```c
float log1pf(float x);
```

**Description**

`log1pf` computes the base-$e$ logarithm of $x$ plus one.
log2

Synopsis

```c
double log2(double x);
```

Description

`log2` computes the base-2 logarithm of `x`. 
**log2f**

**Synopsis**

```c
float log2f(float x);
```

**Description**

`log2f` computes the base-2 logarithm of `x`. 
logb

Synopsis

double logb(double x);

Description

logb computes the base-FLT_RADIX logarithm of x.
logbf

Synopsis

```c
float logbf(float x);
```

Description

`logbf` computes the base-`FLT_RADIX` logarithm of `x`. 
**logf**

**Synopsis**

```c
float logf(float x);
```

**Description**

`logf` computes the base-e logarithm of `x`.

If `x = 0`, `errno` is set to `ERANGE` and `logf` returns `HUGE_VALF`. If `x < 0`, `errno` is set to `EDOM` and `logf` returns `HUGE_VALF`.

If `x < 0` or `x = NaN`, `logf` returns NaN.
If `x = 0`, `logf` returns `-inf`.
If `x = NaN`, `logf` returns `x`.
lrint

Synopsis

```c
long int lrint(double x);
```

Description

lrint rounds x to an integral value and returns it as a long int.
lrintf

Synopsis

\[
\text{long int \ lrintf(float } x)\;
\]

Description

lrintf rounds \( x \) to an integral value and returns it as a long int.
lround

Synopsis

```c
long int lround(double x);
```

Description

`lround` rounds `x` to an integral value, with halfway cases rounded away from zero, and returns it as a long int.
lroundf

Synopsis

```c
long int lroundf(float x);
```

Description

`lroundf` rounds `x` to an integral value, with halfway cases rounded away from zero, and returns it as a long int.
modf

Synopsis

```c
double modf(double x, 
    double *iptr);
```

Description

`modf` breaks `x` into integral and fractional parts, each of which has the same type and sign as `x`.

The integral part (in floating-point format) is stored in the object pointed to by `iptr` and `modf` returns the signed fractional part of `x`. 
modff

Synopsis

```
float modff(float x, float *iptr);
```

Description

modff breaks x into integral and fractional parts, each of which has the same type and sign as x.

The integral part (in floating-point format) is stored in the object pointed to by iptr and modff returns the signed fractional part of x.
nearbyint

Synopsis

```c
double nearbyint (double);
```

Description

everyint Rounds x to an integral value.
nearbyintf

Synopsis

```c
float nearbyintf(float);
```

Description

`nearbyintf` Rounds x to an integral value.
nextafter

Synopsis

```c
double nextafter(double x, double y);
```

Description

nextafter Returns the next representable value after x in the direction of y.
nextafterf

Synopsis

```c
float nextafterf(float x,
               float y);
```

Description

`nextafterf` Returns the next representable value after `x` in the direction of `y`. 
pow

Synopsis

```c
double pow(double x,
            double y);
```

Description

`pow` computes `x` raised to the power `y`.

If `x < 0` and `y 0`, `errno` is set to `EDOM` and `pow` returns `HUGE_VAL`. If `x 0` and `y` is not an integer value, `errno` is set to `EDOM` and `pow` returns `HUGE_VAL`.

If `y = 0`, `pow` returns 1.
If `y = 1`, `pow` returns `x`.
If `y = NaN`, `pow` returns `NaN`.
If `x = NaN` and `y` is anything other than 0, `pow` returns `NaN`.
If `x < 1` or `1 < x`, and `y = +`, `pow` returns `+`.
If `x < 1` or `1 < x`, and `y = -`, `pow` returns 0.
If `1 < x < 1` and `y = +`, `pow` returns `+0`.
If `1 < x < 1` and `y = -`, `pow` returns `+`.
If `x = +1` or `x = 1` and `y = +` or `y = -`, `pow` returns `NaN`.
If `x = +0` and `y > 0` and `y NaN` or `y` not an odd integer, `pow` returns `+0`.
If `x = +0` and `y > 0` and `y NaN` or `y` not an odd integer, `pow` returns `+`.
If `x = 0` and `y` is an odd integer, `pow` returns 0.
If `x = +` and `y > 0` and `y NaN`, `pow` returns `+`.
If `x = +` and `y < 0` and `y NaN`, `pow` returns `+0`.
If `x = -`, `pow` returns `pow(0, y)`.
If `x < 0` and `x` and `y` is a non-integer, `pow` returns `NaN`. 
powf

Synopsis

```c
float powf(float x, float y);
```

Description

powf computes x raised to the power y.

If \( x < 0 \) and \( y < 0 \), errno is set to EDOM and powf returns HUGE_VALF. If \( x = 0 \) and \( y \) is not an integer value, errno is set to EDOM and pow returns HUGE_VALF.

If \( y = 0 \), powf returns 1.

If \( y = 1 \), powf returns \( x \).

If \( y = NaN \), powf returns NaN.

If \( x = NaN \) and \( y \) is anything other than 0, powf returns NaN.

If \( x < 1 \) or \( 1 < x \), and \( y = + \), powf returns +.

If \( x < 1 \) or \( 1 < x \), and \( y = - \), powf returns 0.

If \( 1 < x < 1 \) and \( y = + \), powf returns +0.

If \( 1 < x < 1 \) and \( y = - \), powf returns -. 

If \( x = +1 \) or \( x = 1 \) and \( y = + \) or \( y = - \), powf returns NaN.

If \( x = +0 \) and \( y > 0 \) and \( y \) NaN or \( y \) not an odd integer, powf returns +0.

If \( x = +0 \) and \( y < 0 \) and \( y \) NaN or \( y \) not an odd integer, powf returns +.

If \( x = 0 \) and \( y < 0 \) and \( y \) NaN or \( y \) not an odd integer, powf returns +.

If \( x = 0 \) and \( y \) is an odd integer, powf returns 0.

If \( x = + \) and \( y > 0 \) and \( y \) NaN, powf returns +.

If \( x = + \) and \( y < 0 \) and \( y \) NaN, powf returns +0.

If \( x = - \), powf returns powf(0, y)

If \( x < 0 \) and \( x \) and \( y \) is a non-integer, powf returns NaN.
remainder

Synopsis

```c
double remainder(double numer,
                  double denom);
```

Description

`remainder` computes the remainder of `numer` divided by `denom`. 
remainderf

Synopsis

```c
float remainderf(float numer,
                 float denom);
```

Description

`remainderf` computes the remainder of `numer` divided by `denom`. 
remquo

Synopsis

```c
double remquo(double numer,
               double denom,
               int *quot);
```

Description

`remquo` computes the remainder of `numer` divided by `denom` and the quotient pointed by `quot`. 
remquof

Synopsis

```c
float remquof(float numer,
        float denom,
        int *quot);
```

Description

remquof computes the remainder of numer divided by denom and the quotient pointed by quot.
rint

Synopsis

\texttt{double rint(double x);} 

Description

rint rounds x to an integral value.
rintf

Synopsis

```c
float rintf(float x);
```

Description

rintf rounds x to an integral value.
round

Synopsis

```c
double round(double x);
```

Description

`round` rounds `x` to an integral value, with halfway cases rounded away from zero.
roundf

Synopsis

```c
float roundf(float x);
```

Description

`roundf` rounds `x` to an integral value, with halfway cases rounded away from zero.
scalbln

Synopsis

```c
double scalbln(double x,
               long int exp);
```

Description

scalbln multiplies x by FLT_RADIX raised to the power exp.
scalblnf

Synopsis

```c
float scalblnf(float x, long int exp);
```

Description

scalblnf multiplies x by FLT_RADIX raised to the power exp.
scalbn

Synopsis

```c
double scalbn(double x, int exp);
```

Description

**scalbn** multiplies a floating-point number by an integral power of **DBL_RADIX**.

As floating-point arithmetic conforms to IEC 60559, **DBL_RADIX** is 2 and **scalbn** is (in this implementation) identical to **ldexp**.

**scalbn** returns $x \times \text{DBL\_RADIX}^\text{exp}$.

If the result overflows, **errno** is set to **ERANGE** and **scalbn** returns **HUGE\_VAL**.

If $x$ is or NaN, **scalbn** returns $x$.

If the result overflows, **scalbn** returns.

See Also

**ldexp**
**scalbnf**

**Synopsis**

```c
float scalbnf(float x, int exp);
```

**Description**

scalbnf multiplies a floating-point number by an integral power of FLT_RADIX.

As floating-point arithmetic conforms to IEC 60559, FLT_RADIX is 2 and scalbnf is (in this implementation) identical to ldexpf.

scalbnf returns \( x \times \text{FLT\_RADIX}^{\exp} \).

If the result overflows, \text{errno} is set to ERANGE and scalbnf returns HUGE\_VALF.

If \( x \) is or NaN, scalbnf returns \( x \). If the result overflows, scalbnf returns .

**See Also**

ldexpf
signbit

Synopsis

```c
#define signbit(x) (sizeof(x) == sizeof(float) ? __float32_signbit(x) : __float64_signbit(x))
```

Description

signbit macro determines whether the sign of `x` is negative. `signbit` returns a non-zero value if and only if `x` is negative.
**sin**

**Synopsis**

```c
double sin(double x);
```

**Description**

`sin` returns the radian circular sine of `x`.

If `|x| > 10^9`, `errno` is set to `EDOM` and `sin` returns `HUGE_VAL`.

`sin` returns `x` if `x` is NaN. `sin` returns NaN if `|x|` is .
sinf

Synopsis

```c
float,sinf(float,x);
```

Description

sinf returns the radian circular sine of x.

If |x| > 10^9, errno is set to EDOM and sinf returns HUGE_VALF.

sinf returns x if x is NaN. sinf returns NaN if |x| is . 
**sinh**

**Synopsis**

```c
double sinh(double x);
```

**Description**

`sinh` calculates the hyperbolic sine of `x`.

If `|x| > 0.782`, `errno` is set to `EDOM` and `sinh` returns `HUGE_VAL`.

If `x` is `+`, `-`, or `NaN`, `sinh` returns `|x|`. If `|x| > ~709.782`, `sinh` returns `+` or depending upon the sign of `x`. 
**sinhf**

**Synopsis**

```c
float sinhf(float x);
```

**Description**

`sinhf` calculates the hyperbolic sine of `x`.

If `|x| > ~88.7228`, `errno` is set to `EDOM` and `sinhf` returns `HUGE_VALF`.

If `x` is +, -, or NaN, `sinhf` returns `|x|`. If `|x| > ~88.7228`, `sinhf` returns + or depending upon the sign of `x`. 
**sqrt**

**Synopsis**

```c
double sqrt(double x);
```

**Description**

`sqrt` computes the nonnegative square root of `x`. C90 and C99 require that a domain error occurs if the argument is less than zero. `sqrt` deviates and always uses IEC 60559 semantics.

If `x` is +0, `sqrt` returns +0.
If `x` is 0, `sqrt` returns 0.
If `x` is , `sqrt` returns .
If `x` < 0, `sqrt` returns NaN.
If `x` is NaN, `sqrt` returns that NaN.
sqrtf

Synopsis

```c
float sqrtf(float x);
```

Description

sqrtf computes the nonnegative square root of x. C90 and C99 require that a domain error occurs if the argument is less than zero sqrtf deviates and always uses IEC 60559 semantics.

- If x is +0, sqrtf returns +0.
- If x is 0, sqrtf returns 0.
- If x is , sqrtf returns .
- If x < 0, sqrtf returns NaN.
- If x is NaN, sqrtf returns that NaN.
\textbf{tan}

\textbf{Synopsis}

\begin{verbatim}
  double tan(double x);
\end{verbatim}

\textbf{Description}

\texttt{tan} returns the radian circular tangent of \texttt{x}.

If $|x| > 10^9$, \texttt{errno} is set to \texttt{EDOM} and \texttt{tan} returns \texttt{HUGE_VAL}.

If \texttt{x} is \texttt{NaN}, \texttt{tan} returns \texttt{x}. If $|x|$ is \texttt{NaN}, \texttt{tan} returns \texttt{NaN}. 
tanf

Synopsis

```c
float tanf(float x);
```

Description

tanf returns the radian circular tangent of x.

If \( |x| > 10^9 \), errno is set to EDOM and tanf returns HUGE_VALF.

If x is NaN, tanf returns x. If \( |x| \) is NaN, tanf returns NaN.
tanh

Synopsis

```c
double tanh(double x);
```

Description

tanh calculates the hyperbolic tangent of x.

If x is NaN, tanh returns NaN.
tanhf

Synopsis

```c
float tanhf(float x);
```

Description

tanhf calculates the hyperbolic tangent of x.

If x is NaN, tanhf returns NaN.
**tgamma**

**Synopsis**

```c
double tgamma(double x);
```

**Description**

*tgamma* returns the gamma function for *x*. 
tgammaf

Synopsis

```c
float tgammaf(float x);
```

Description

tgammaf returns the gamma function for x.
trunc

Synopsis

double trunc(double x);

Description

double trunc(double x); trunc rounds x to an integral value that is not larger in magnitude than x.
truncf

Synopsis

```c
float truncf(float x);
```

Description

`truncf` rounds `x` to an integral value that is not larger in magnitude than `x`. 
<setjmp.h>

## API Summary

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>longjmp</td>
<td>Restores the saved environment</td>
</tr>
<tr>
<td>setjmp</td>
<td>Save calling environment for non-local jump</td>
</tr>
</tbody>
</table>
longjmp

Synopsis

```c
void longjmp(jmp_buf env,
             int val);
```

Description

`longjmp` restores the environment saved by `setjmp` in the corresponding `env` argument. If there has been no such invocation, or if the function containing the invocation of `setjmp` has terminated execution in the interim, the behavior of `longjmp` is undefined.

After `longjmp` is completed, program execution continues as if the corresponding invocation of `setjmp` had just returned the value specified by `val`.

Note

`longjmp` cannot cause `setjmp` to return the value 0; if `val` is 0, `setjmp` returns the value 1.

Objects of automatic storage allocation that are local to the function containing the invocation of the corresponding `setjmp` that do not have `volatile` qualified type and have been changed between the `setjmp` invocation and this call are indeterminate.
setjmp

Synopsis

```c
int setjmp(jmp_buf env);
```

Description

`setjmp` saves its calling environment in the `env` for later use by the `longjmp` function.

On return from a direct invocation `setjmp` returns the value zero. On return from a call to the `longjmp` function, the `setjmp` returns a nonzero value determined by the call to `longjmp`.

The environment saved by a call to `setjmp` consists of information sufficient for a call to the `longjmp` function to return execution to the correct block and invocation of that block, were it called recursively.
**<stdarg.h>**

### API Summary

<table>
<thead>
<tr>
<th>Macros</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>va_arg</td>
<td>Get variable argument value</td>
</tr>
<tr>
<td>va_copy</td>
<td>Copy var args</td>
</tr>
<tr>
<td>va_end</td>
<td>Finish access to variable arguments</td>
</tr>
<tr>
<td>va_start</td>
<td>Start access to variable arguments</td>
</tr>
</tbody>
</table>
va_arg

Synopsis

```c
type va_arg(va_list ap, type);
```

Description

va_arg expands to an expression that has the specified type and the value of the type argument. The ap parameter must have been initialized by va_start or va_copy, without an intervening invocation of va_end. You can create a pointer to a va_list and pass that pointer to another function, in which case the original function may make further use of the original list after the other function returns.

Each invocation of the va_arg macro modifies ap so that the values of successive arguments are returned in turn. The parameter type must be a type name such that the type of a pointer to an object that has the specified type can be obtained simply by postfixing a * to type.

If there is no actual next argument, or if type is not compatible with the type of the actual next argument (as promoted according to the default argument promotions), the behavior of va_arg is undefined, except for the following cases:

- one type is a signed integer type, the other type is the corresponding unsigned integer type, and the value is representable in both types;
- one type is pointer to void and the other is a pointer to a character type.

The first invocation of the va_arg macro after that of the va_start macro returns the value of the argument after that specified by parmN. Successive invocations return the values of the remaining arguments in succession.
**va_copy**

**Synopsis**

```c
void va_copy(va_list dest, val_list src);
```

**Description**

`va_copy` initializes `dest` as a copy of `src`, as if the `va_start` macro had been applied to `dest` followed by the same sequence of uses of the `va_arg` macro as had previously been used to reach the present state of `src`. Neither the `va_copy` nor `va_start` macro shall be invoked to reinitialize `dest` without an intervening invocation of the `va_end` macro for the same `dest`. 
va_end

Synopsis

```c
void va_end(va_list ap);
```

Description

`va_end` indicates a normal return from the function whose variable argument list `ap` was initialised by `va_start` or `va_copy`. The `va_end` macro may modify `ap` so that it is no longer usable without being reinitialized by `va_start` or `va_copy`. If there is no corresponding invocation of `va_start` or `va_copy`, or if `va_end` is not invoked before the return, the behavior is undefined.
va_start

Synopsis

```c
void va_start(va_list ap,
              paramN);
```

Description

`va_start` initializes `ap` for subsequent use by the `va_arg` and `va_end` macros.

The parameter `parmN` is the identifier of the last fixed parameter in the variable parameter list in the function definition (the one just before the `', ...'`).

The behaviour of `va_start` and `va_arg` is undefined if the parameter `parmN` is declared with the `register` storage class, with a function or array type, or with a type that is not compatible with the type that results after application of the default argument promotions.

`va_start` must be invoked before any access to the unnamed arguments.

`va_start` and `va_copy` must not be invoked to reinitialize `ap` without an intervening invocation of the `va_end` macro for the same `ap`. 
<stddef.h>

API Summary

<table>
<thead>
<tr>
<th>Macros</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL pointer</td>
</tr>
<tr>
<td>offsetof</td>
<td>offsetof</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Types</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ptrdiff_t</td>
<td>ptrdiff_t type</td>
</tr>
<tr>
<td>size_t</td>
<td>size_t type</td>
</tr>
</tbody>
</table>
NULL

Synopsis

#define NULL 0

Description

NULL is the null pointer constant.
offsetof

Synopsis

#define offsetof(type, member)

Description

offsetof returns the offset in bytes to the structure member, from the beginning of its structure type.
ptrdiff_t

Synopsis

typedef __RAL_PTRDIFF_T ptrdiff_t;

Description

ptrdiff_t is the signed integral type of the result of subtracting two pointers.
size_t

Synopsis

typedef __RAL_SIZE_T size_t;

Description

size_t is the unsigned integral type returned by the sizeof operator.
### API Summary

<table>
<thead>
<tr>
<th>Character and string I/O functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getchar</td>
<td>Read a character from standard input</td>
</tr>
<tr>
<td>gets</td>
<td>Read a string from standard input</td>
</tr>
<tr>
<td>putchar</td>
<td>Write a character to standard output</td>
</tr>
<tr>
<td>puts</td>
<td>Write a string to standard output</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Formatted output functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printf</td>
<td>Write formatted text to standard output</td>
</tr>
<tr>
<td>snprintf</td>
<td>Write formatted text to a string with truncation</td>
</tr>
<tr>
<td>sprintf</td>
<td>Write formatted text to a string</td>
</tr>
<tr>
<td>vprintf</td>
<td>Write formatted text to standard output using variable argument context</td>
</tr>
<tr>
<td>vsnprintf</td>
<td>Write formatted text to a string with truncation using variable argument context</td>
</tr>
<tr>
<td>vsprintf</td>
<td>Write formatted text to a string using variable argument context</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Formatted input functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scanf</td>
<td>Read formatted text from standard input</td>
</tr>
<tr>
<td>sscanf</td>
<td>Read formatted text from string</td>
</tr>
<tr>
<td>vscanf</td>
<td>Read formatted text from standard using variable argument context</td>
</tr>
<tr>
<td>vsscanf</td>
<td>Read formatted text from a string using variable argument context</td>
</tr>
</tbody>
</table>
getchar

Synopsis

```c
int getchar(void);
```

Description

`getchar` reads a single character from the standard input stream.

If the stream is at end-of-file or a read error occurs, `getchar` returns `EOF`. 
gets

Synopsis

```c
char *gets(char *s);
```

Description

gets reads characters from standard input into the array pointed to by s until end-of-file is encountered or a new-line character is read. Any new-line character is discarded, and a null character is written immediately after the last character read into the array.

gets returns s if successful. If end-of-file is encountered and no characters have been read into the array, the contents of the array remain unchanged and gets returns a null pointer. If a read error occurs during the operation, the array contents are indeterminate and gets returns a null pointer.
printf

Synopsis

```c
int printf(const char *format, ...
```

Description

`printf` writes to the standard output stream using `putchar`, under control of the string pointed to by `format` that specifies how subsequent arguments are converted for output.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

`printf` returns the number of characters transmitted, or a negative value if an output or encoding error occurred.

Formatted output control strings

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.
Some library variants do not support width and precision specifiers in order to reduce code and data space requirements; please ensure that you have selected the correct library in the Printf Width/Precision Support property of the project if you use these.

Flag characters

The flag characters and their meanings are:

- 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 character 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.

Length modifiers

The length modifiers and their meanings are:

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.
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.

I

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. Some library variants do not support the l length modifier in order to reduce code and data space requirements; please ensure that you have selected the correct library in the Printf Integer Support property of the project if you use this length modifier.

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. Some library variants do not support the ll length modifier in order to reduce code and data space requirements; please ensure that you have selected the correct library in the Printf Integer Support property of the project if you use this length modifier.

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, t, and L are not supported.

Conversion specifiers

The conversion specifiers and their meanings are:

d, i

The argument is converted to signed decimal in the style [-]ddddd. 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 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. Some library variants do not support the f and F conversion specifiers in order to reduce code and data space requirements; please ensure that you have selected the correct library in the Printf Floating Point Support property of the project if you use these conversion specifiers.

**e, E**

A double argument representing a floating-point number is converted in the style [-]d.ddd e 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. Some library variants do not support the f and F conversion specifiers in order to reduce code and data space requirements; please ensure that you have selected the correct library in the Printf Floating Point Support property of the project if you use these conversion specifiers.

**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. Some library variants do not support the f and F conversion specifiers in order to reduce code and data space requirements; please ensure that you have selected the correct library in the Printf Floating Point Support property of the project if you use these conversion specifiers.

**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 \texttt{void}. The value of the pointer is converted in the same format as the \texttt{x} conversion specifier with a fixed precision of \(2^n \texttt{sizeof(void *)}\).

n
The argument is a pointer to a signed integer into which is \textit{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 \texttt{l} used in conjunction with the \texttt{c} and \texttt{s} conversion specifiers is not supported and nor are the conversion specifiers \texttt{a} and \texttt{A}. 
putchar

Synopsis

```c
int putchar(int c);
```

Description

`putchar` writes the character `c` to the standard output stream.

`putchar` returns the character written. If a write error occurs, `putchar` returns `EOF`. 
puts

Synopsis

```c
int puts(const char *s);
```

Description

`puts` writes the string pointed to by `s` to the standard output stream using `putchar` and appends a new-line character to the output. The terminating null character is not written.

`puts` returns `EOF` if a write error occurs; otherwise it returns a nonnegative value.
**scanf**

**Synopsis**

```c
int scanf(const char *format, ...);
```

**Description**

`scanf` reads input from the standard input stream under control of the string pointed to by `format` that specifies the admissible input sequences and how they are to be converted for assignment, using subsequent arguments as pointers to the objects to receive the converted input.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

`scanf` returns the value of the macro `EOF` if an input failure occurs before any conversion. Otherwise, `scanf` returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

**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.

**Length modifiers**

The length modifiers and their meanings are:

**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`.

**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`. Some library variants do not support the l length modifier in order to reduce code and data space requirements; please ensure that you have selected the correct library in the Printf Integer Support property of the project if you use this length modifier.

**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`. Some library variants do not support the ll length modifier in order to reduce code and data space requirements; please ensure that you have selected the correct library in the Printf Integer Support property of the project if you use this length modifier.

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, t, and L are not supported.
Conversion specifiers

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 `strtol` function. The corresponding argument shall be a pointer to floating. Some library variants do not support the e, f and F conversion specifiers in order to reduce code and data space requirements; please ensure that you have selected the correct library in the Scanf Floating Point Support property of the project if you use these conversion specifiers.

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 \texttt{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 \texttt{scanlist}) compose the scanset, unless the character after the left bracket is a circumflex \texttt{^}, in which case the scanset contains all characters that do 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. Some library variants do not support the [ conversion specifier in order to reduce code and data space requirements; please ensure that you have selected the correct library in the \textbf{Scanf Classes Supported} property of the project if you use this conversion specifier.

\texttt{p} Reads a sequence output by the corresponding \texttt{\%p} formatted output conversion. The corresponding argument must be a pointer to a pointer to \texttt{void}.

\texttt{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 \texttt{\%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.

\texttt{\%} Matches a single \texttt{\%} 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.
snprintf

Synopsis

```c
int snprintf(char *s,
             size_t n,
             const char *format,
             ...);
```

Description

snprintf writes to the string pointed to by s under control of the string pointed to by format that specifies how subsequent arguments are converted for output.

If n is zero, nothing is written, and s can be a null pointer. Otherwise, output characters beyond the n1st are discarded rather than being written to the array, and a null character is written at the end of the characters actually written into the array. A null character is written at the end of the conversion; it is not counted as part of the returned value.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

If copying takes place between objects that overlap, the behavior is undefined.

snprintf returns the number of characters that would have been written had n been sufficiently large, not counting the terminating null character, or a negative value if an encoding error occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than n>.
sprintf

Synopsis

```c
int sprintf(char *s,  
    const char *format,  
    ...);
```

Description

`sprintf` writes to the string pointed to by `s` under control of the string pointed to by `format` that specifies how subsequent arguments are converted for output. A null character is written at the end of the characters written; it is not counted as part of the returned value.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

If copying takes place between objects that overlap, the behavior is undefined.

`sprintf` returns number of characters transmitted (not counting the terminating null), or a negative value if an output or encoding error occurred.
**sscanf**

**Synopsis**

```c
int sscanf(const char *s,
           const char *format,
           ...);
```

**Description**

`sscanf` reads input from the string `s` under control of the string pointed to by `format` that specifies the admissible input sequences and how they are to be converted for assignment, using subsequent arguments as pointers to the objects to receive the converted input.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

`sscanf` returns the value of the macro `EOF` if an input failure occurs before any conversion. Otherwise, `sscanf` returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.
vprintf

Synopsis

```c
int vprintf(const char *format, __va_list arg);
```

Description

vprintf writes to the standard output stream using putchar under control of the string pointed to by format that specifies how subsequent arguments are converted for output. Before calling vprintf, arg must be initialized by the va_start macro (and possibly subsequent va_arg calls). vprintf does not invoke the va_end macro.

vprintf returns the number of characters transmitted, or a negative value if an output or encoding error occurred.

Note

vprintf is equivalent to printf with the variable argument list replaced by arg.
vscanf

Synopsis

```c
int vscanf(const char *format, __va_list arg);
```

Description

vscanf reads input from the standard input stream under control of the string pointed to by format that specifies the admissible input sequences and how they are to be converted for assignment, using subsequent arguments as pointers to the objects to receive the converted input. Before calling vscanf, arg must be initialized by the va_start macro (and possibly subsequent va_arg calls). vscanf does not invoke the va_end macro.

If there are insufficient arguments for the format, the behavior is undefined.

vscanf returns the value of the macro EOF if an input failure occurs before any conversion. Otherwise, vscanf returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

Note

vscanf is equivalent to scanf with the variable argument list replaced arg.
vsnprintf

Synopsis

```c
int vsnprintf(char *s,
     size_t n,
     const char *format,
     __va_list arg);
```

Description

vsnprintf writes to the string pointed to by s under control of the string pointed to by format that specifies how subsequent arguments are converted for output. Before calling vsnprintf, arg must be initialized by the va_start macro (and possibly subsequent va_arg calls). vsnprintf does not invoke the va_end macro.

If n is zero, nothing is written, and s can be a null pointer. Otherwise, output characters beyond the n1st are discarded rather than being written to the array, and a null character is written at the end of the characters actually written into the array. A null character is written at the end of the conversion; it is not counted as part of the returned value.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

If copying takes place between objects that overlap, the behavior is undefined.

vsnprintf returns the number of characters that would have been written had n been sufficiently large, not counting the terminating null character, or a negative value if an encoding error occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than n.

Note

vsnprintf is equivalent to snprintf with the variable argument list replaced by arg.
vsprintf

Synopsis

```c
int vsprintf(char *s,
    const char *format,
    __va_list arg);
```

Description

vsprintf writes to the string pointed to by *s* under control of the string pointed to by *format* that specifies how subsequent arguments are converted for output. Before calling vsprintf, *arg* must be initialized by the va_start macro (and possibly subsequent va_arg calls). vsprintf does not invoke the va_end macro.

A null character is written at the end of the characters written; it is not counted as part of the returned value.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

If copying takes place between objects that overlap, the behavior is undefined.

vsprintf returns number of characters transmitted (not counting the terminating null), or a negative value if an output or encoding error occurred.

Note

vsprintf is equivalent to sprintf with the variable argument list replaced by *arg*. 
vsscanf

Synopsis

```c
int vsscanf(const char *s,
            const char *format,
            __va_list arg);
```

Description

`vsscanf` reads input from the string `s` under control of the string pointed to by `format` that specifies the admissible input sequences and how they are to be converted for assignment, using subsequent arguments as pointers to the objects to receive the converted input. Before calling `vsscanf`, `arg` must be initialized by the `va_start` macro (and possibly subsequent `va_arg` calls). `vsscanf` does not invoke the `va_end` macro.

If there are insufficient arguments for the format, the behavior is undefined.

`vsscanf` returns the value of the macro `EOF` if an input failure occurs before any conversion. Otherwise, `vsscanf` returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

Note

`vsscanf` is equivalent to `sscanf` with the variable argument list replaced by `arg`. 
# <stdlib.h>

## API Summary

### Macros

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXIT_FAILURE</td>
<td>EXIT_FAILURE</td>
</tr>
<tr>
<td>EXIT_SUCCESS</td>
<td>EXIT_SUCCESS</td>
</tr>
<tr>
<td>MB_CUR_MAX</td>
<td>Maximum number of bytes in a multi-byte character in the current locale</td>
</tr>
<tr>
<td>RAND_MAX</td>
<td>RAND_MAX</td>
</tr>
</tbody>
</table>

### Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>div_t</td>
<td>Structure containing quotient and remainder after division of an int</td>
</tr>
<tr>
<td>ldiv_t</td>
<td>Structure containing quotient and remainder after division of a long</td>
</tr>
<tr>
<td>lldiv_t</td>
<td>Structure containing quotient and remainder after division of a long long</td>
</tr>
</tbody>
</table>

### Integer arithmetic functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>Return an integer absolute value</td>
</tr>
<tr>
<td>div</td>
<td>Divide two ints returning quotient and remainder</td>
</tr>
<tr>
<td>labs</td>
<td>Return a long integer absolute value</td>
</tr>
<tr>
<td>ldiv</td>
<td>Divide two longs returning quotient and remainder</td>
</tr>
<tr>
<td>llabs</td>
<td>Return a long long integer absolute value</td>
</tr>
<tr>
<td>lldiv</td>
<td>Divide two long longs returning quotient and remainder</td>
</tr>
</tbody>
</table>

### Memory allocation functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calloc</td>
<td>Allocate space for an array of objects and initialize them to zero</td>
</tr>
<tr>
<td>free</td>
<td>Frees allocated memory for reuse</td>
</tr>
<tr>
<td>malloc</td>
<td>Allocate space for a single object</td>
</tr>
<tr>
<td>realloc</td>
<td>Resizes allocated memory space or allocates memory space</td>
</tr>
</tbody>
</table>

### String to number conversions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>atof</td>
<td>Convert string to double</td>
</tr>
<tr>
<td>atoi</td>
<td>Convert string to int</td>
</tr>
<tr>
<td>atol</td>
<td>Convert string to long</td>
</tr>
<tr>
<td>atoll</td>
<td>Convert string to long long</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>strtod</td>
<td>Convert string to double</td>
</tr>
<tr>
<td>strtof</td>
<td>Convert string to float</td>
</tr>
<tr>
<td>strtol</td>
<td>Convert string to long</td>
</tr>
<tr>
<td>strtoll</td>
<td>Convert string to long long</td>
</tr>
<tr>
<td>strtoul</td>
<td>Convert string to unsigned long</td>
</tr>
<tr>
<td>strtoull</td>
<td>Convert string to unsigned long long</td>
</tr>
</tbody>
</table>

**Pseudo-random sequence generation functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rand</td>
<td>Return next random number in sequence</td>
</tr>
<tr>
<td>srand</td>
<td>Set seed of random number sequence</td>
</tr>
</tbody>
</table>

**Search and sort functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bsearch</td>
<td>Search a sorted array</td>
</tr>
<tr>
<td>qsort</td>
<td>Sort an array</td>
</tr>
</tbody>
</table>

**Environment**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>atexit</td>
<td>Set function to be execute on exit</td>
</tr>
<tr>
<td>exit</td>
<td>Terminates the calling process</td>
</tr>
</tbody>
</table>

**Number to string conversions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>itoa</td>
<td>Convert int to string</td>
</tr>
<tr>
<td>lltoa</td>
<td>Convert long long to string</td>
</tr>
<tr>
<td>ltoa</td>
<td>Convert long to string</td>
</tr>
<tr>
<td>ulltoa</td>
<td>Convert unsigned long long to string</td>
</tr>
<tr>
<td>ultoa</td>
<td>Convert unsigned long to string</td>
</tr>
<tr>
<td>utoa</td>
<td>Convert unsigned to string</td>
</tr>
</tbody>
</table>

**Multi-byte/wide character conversion functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mblen</td>
<td>Determine number of bytes in a multi-byte character</td>
</tr>
<tr>
<td>mblen_l</td>
<td>Determine number of bytes in a multi-byte character</td>
</tr>
</tbody>
</table>

**Multi-byte/wide string conversion functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbstowcs</td>
<td>Convert multi-byte string to wide string</td>
</tr>
<tr>
<td>mbstowcs_l</td>
<td>Convert multi-byte string to wide string using specified locale</td>
</tr>
<tr>
<td>mbtowc</td>
<td>Convert multi-byte character to wide character</td>
</tr>
<tr>
<td>mbtowc_l</td>
<td>Convert multi-byte character to wide character</td>
</tr>
</tbody>
</table>
EXIT_FAILURE

Synopsis

```c
#define EXIT_FAILURE 1
```

Description

EXIT_FAILURE pass to `exit` on unsuccessful termination.
EXIT_SUCCESS

Synopsis

#define EXIT_SUCCESS 0

Description

EXIT_SUCCESS pass to exit on successful termination.
MB_CUR_MAX

Synopsis

```c
#define MB_CUR_MAX __RAL_mb_max(__RAL_global_locale)
```

Description

`MB_CUR_MAX` expands to a positive integer expression with type `size_t` that is the maximum number of bytes in a multi-byte character for the extended character set specified by the current locale (category LC_CTYPE). `MB_CUR_MAX` is never greater than `MB_LEN_MAX`. 
RAND_MAX

Synopsis

#define RAND_MAX 32767

Description

RAND_MAX expands to an integer constant expression that is the maximum value returned by rand.
abs

Synopsis

```c
int abs(int j);
```

Description

`abs` returns the absolute value of the integer argument `j`. 
atexit

Synopsis

```c
int atexit(void (*func)(void));
```

Description

`atexit` registers `function` to be called when the application has exited. The functions registered with `atexit` are executed in reverse order of their registration. `atexit` returns 0 on success and non-zero on failure.
**atof**

**Synopsis**

```c
double atof(const char *nptr);
```

**Description**

`atof` converts the initial portion of the string pointed to by `nptr` to a **double** representation.

`atof` does not affect the value of `errno` on an error. If the value of the result cannot be represented, the behavior is undefined.

Except for the behavior on error, `atof` is equivalent to `strtol(nptr, (char **)NULL)`.

`atof` returns the converted value.

**See Also**

`strtol`
atoi

Synopsis

```c
int atoi(const char *nptr);
```

Description

`atoi` converts the initial portion of the string pointed to by `nptr` to an `int` representation.

`atoi` does not affect the value of `errno` on an error. If the value of the result cannot be represented, the behavior is undefined.

Except for the behavior on error, `atoi` is equivalent to `(int)strtol(nptr, (char **)NULL, 10)`.

`atoi` returns the converted value.

See Also

`strtol`
**atol**

**Synopsis**

```c
long int atol(const char *nptr);
```

**Description**

`atol` converts the initial portion of the string pointed to by `nptr` to a `long int` representation.

`atol` does not affect the value of `errno` on an error. If the value of the result cannot be represented, the behavior is undefined.

Except for the behavior on error, `atol` is equivalent to `strtol(nptr, (char **)NULL, 10)`.

`atol` returns the converted value.

**See Also**

`strtol`
atoll

Synopsis

```c
long long int atoll(const char *nptr);
```

Description

atoll converts the initial portion of the string pointed to by nptr to a long long int representation.

atoll does not affect the value of errno on an error. If the value of the result cannot be represented, the behavior is undefined.

Except for the behavior on error, atoll is equivalent to strftime(nptr, (char **)NULL, 10).

atoll returns the converted value.

See Also

strftime
bsearch

Synopsis

```c
void *bsearch(const void *key,
              const void *buf,
              size_t num,
              size_t size,
              int (*compare)(const void *, const void *));
```

Description

bsearch searches the array *base for the specified *key and returns a pointer to the first entry that matches or null if no match. The array should have num elements of size bytes and be sorted by the same algorithm as the compare function.

The compare function should return a negative value if the first parameter is less than second parameter, zero if the parameters are equal, and a positive value if the first parameter is greater than the second parameter.
calloc

Synopsis

```c
void *calloc(size_t nobj,
            size_t size);
```

Description

calloc allocates space for an array of nmemb objects, each of whose size is size. The space is initialized to all zero bits.

calloc returns a null pointer if the space for the array of object cannot be allocated from free memory; if space for the array can be allocated, calloc returns a pointer to the start of the allocated space.
**div**

**Synopsis**

```c
div_t div(int numer,
           int denom);
```

**Description**

`div` computes `numer / denom` and `numer % denom` in a single operation.

`div` returns a structure of type `div_t` comprising both the quotient and the remainder. The structures contain the members `quot` (the quotient) and `rem` (the remainder), each of which has the same type as the arguments `numer` and `denom`. If either part of the result cannot be represented, the behavior is undefined.

**See Also**

`div_t`
**div_t**

**Description**

`div_t` stores the quotient and remainder returned by `div`. 
exit

Synopsis

```c
void exit(int exit_code);
```

Description

`exit` returns to the startup code and performs the appropriate cleanup process.
free

Synopsis

```c
void free(void *p);
```

Description

`free` causes the space pointed to by `ptr` to be deallocated, that is, made available for further allocation. If `ptr` is a null pointer, no action occurs.

If `ptr` does not match a pointer earlier returned by `calloc`, `malloc`, or `realloc`, or if the space has been deallocated by a call to `free` or `realloc`, the behavior is undefined.
itoa

Synopsis

char *itoa(int val,
    char *buf,
    int radix);

Description

itoa converts val to a string in base radix and places the result in buf.

itoa returns buf as the result.

If radix is greater than 36, the result is undefined.

If val is negative and radix is 10, the string has a leading minus sign (-); for all other values of radix, value is considered unsigned and never has a leading minus sign.

See Also

Itoa, lltoa, ultoa, ulltoa, utoa
labs

Synopsis

```c
long int labs(long int j);
```

Description

`labs` returns the absolute value of the long integer argument `j`. 
ldiv

Synopsis

```c
ldiv_t ldiv(long int numer, long int denom);
```

Description

ldiv computes `numer / denom` and `numer % denom` in a single operation.

ldiv returns a structure of type `ldiv_t` comprising both the quotient and the remainder. The structures contain the members `quot` (the quotient) and `rem` (the remainder), each of which has the same type as the arguments `numer` and `denom`. If either part of the result cannot be represented, the behavior is undefined.

See Also

ldiv_t
ldiv_t

Description

ldiv_t stores the quotient and remainder returned by ldiv.
llabs

Synopsis

```c
long long int llabs(long long int j);
```

Description

`llabs` returns the absolute value of the long long integer argument `j`. 
lldiv

Synopsis

```c
lldiv_t lldiv(long long int numer,
              long long int denom);
```

lldiv computes `numer / denom` and `numer % denom` in a single operation.

lldiv returns a structure of type `lldiv_t` comprising both the quotient and the remainder. The structures contain the members `quot` (the quotient) and `rem` (the remainder), each of which has the same type as the arguments `numer` and `denom`. If either part of the result cannot be represented, the behavior is undefined.

See Also

lldiv_t
lldiv_t

Description

lldiv_t stores the quotient and remainder returned by lldiv.
Iltoa

Synopsis

```c
char *lltoa(long long val,
    char *buf,
    int radix);
```

Description

Iltoa converts val to a string in base radix and places the result in buf.

Iltoa returns buf as the result.

If radix is greater than 36, the result is undefined.

If val is negative and radix is 10, the string has a leading minus sign (-); for all other values of radix, value is considered unsigned and never has a leading minus sign.

See Also

itoa, ltoa, ultoa, ulltoa, utoa
Itoa

Synopsis

```c
char *ltoa(long val,
    char *buf,
    int radix);
```

Description

Itoa converts val to a string in base radix and places the result in buf.

Itoa returns buf as the result.

If radix is greater than 36, the result is undefined.

If val is negative and radix is 10, the string has a leading minus sign (-); for all other values of radix, value is considered unsigned and never has a leading minus sign.

See Also

itoa, lltoa, ultoa, ulltoa, utoa
malloc

Synopsis

```c
void *malloc(size_t size);
```

Description

malloc allocates space for an object whose size is specified by `size` and whose value is indeterminate.

malloc returns a null pointer if the space for the object cannot be allocated from free memory; if space for the object can be allocated, malloc returns a pointer to the start of the allocated space.
mblen

Synopsis

```c
int mblen(const char *s, size_t n);
```

Description

The `mblen` function determines the number of bytes contained in the multi-byte character pointed to by `s` in the current locale.

If `s` is a null pointer, `mblen` returns a nonzero or zero value, if multi-byte character encodings, respectively, do or do not have state-dependent encodings.

If `s` is not a null pointer, `mblen` either returns 0 (if `s` points to the null character), or returns the number of bytes that are contained in the multi-byte character (if the next `n` or fewer bytes form a valid multi-byte character), or returns 1 (if they do not form a valid multi-byte character).

Note

Except that the conversion state of the `mbtowc` function is not affected, it is equivalent to

```c
mbtowc((wchar_t *)0, s, n);
```

Note

It is guaranteed that no library function in the Standard C library calls `mblen`.

See Also

`mblen_l`, `mbtowc`
mblen_l

Synopsis

```c
int mblen_l(const char *s,
             size_t n,
             __locale_s *loc);
```

Description

`mblen_l` determines the number of bytes contained in the multi-byte character pointed to by `s` in the locale `loc`.

If `s` is a null pointer, `mblen_l` returns a nonzero or zero value, if multi-byte character encodings, respectively, do or do not have state-dependent encodings.

If `s` is not a null pointer, `mblen_l` either returns 0 (if `s` points to the null character), or returns the number of bytes that are contained in the multi-byte character (if the next `n` or fewer bytes form a valid multi-byte character), or returns 1 (if they do not form a valid multi-byte character).

Note

Except that the conversion state of the `mbtowc_l` function is not affected, it is equivalent to

```c
mbtowc((wchar_t *)0, s, n, loc);
```

Note

It is guaranteed that no library function in the Standard C library calls `mblen_l`.

See Also

`mblen_l, mbtowc_l`
mbstowcs

Synopsis

```c
size_t mbstowcs(wchar_t *pwcs,
    const char *s,
    size_t n);
```

Description

`mbstowcs` converts a sequence of multi-byte characters that begins in the initial shift state from the array pointed to by `s` into a sequence of corresponding wide characters and stores not more than `n` wide characters into the array pointed to by `pwcs`.

No multi-byte characters that follow a null character (which is converted into a null wide character) will be examined or converted. Each multi-byte character is converted as if by a call to the `mbtowc` function, except that the conversion state of the `mbtowc` function is not affected.

No more than `n` elements will be modified in the array pointed to by `pwcs`. If copying takes place between objects that overlap, the behavior is undefined.

`mbstowcs` returns 1 if an invalid multi-byte character is encountered, otherwise `mbstowcs` returns the number of array elements modified (if any), not including a terminating null wide character.
mbstowcs_l

Synopsis

```c
size_t mbstowcs_l(wchar_t *pwcs,
                 const char *s,
                 size_t n,
                 __locale_s *loc);
```

Description

`mbstowcs_l` is as `mbstowcs` except that the local `loc` is used for the conversion as opposed to the current locale.

See Also

`mbstowcs`.
mbtowc

Synopsis

```c
int mbtowc(wchar_t *pwc,
            const char *s,
            size_t n);
```

Description

`mbtowc` converts a single multi-byte character to a wide character in the current locale.

If `s` is a null pointer, `mbtowc` returns a nonzero value if multi-byte character encodings are state-dependent in the current locale, and zero otherwise.

If `s` is not null and the object that `s` points to is a wide-character null character, `mbtowc` returns 0.

If `s` is not null and the object that points to forms a valid multi-byte character, `mbtowc` returns the length in bytes of the multi-byte character.

If the object that points to does not form a valid multi-byte character within the first `n` characters, it returns 1.

See Also

`mbtowc_l`
mbtowc_l

Synopsis

```c
int mbtowc_l(wchar_t *pwc,
    const char *s,
    size_t n,
    __locale_s *loc);
```

Description

`mbtowc_l` converts a single multi-byte character to a wide character in locale `loc`.

If `s` is a null pointer, `mbtowc_l` returns a nonzero value if multi-byte character encodings are state-dependent in the locale `loc`, and zero otherwise.

If `s` is not null and the object that `s` points to is a wide-character null character, `mbtowc_l` returns 0.

If `s` is not null and the object that points to forms a valid multi-byte character, `mbtowc_l` returns the length in bytes of the multi-byte character.

If the object that `s` points to does not form a valid multi-byte character within the first `n` characters, it returns 1.

See Also

`mbtowc`
qsort

Synopsis

```c
void qsort(void *buf,
          size_t num,
          size_t size,
          int (*compare)(const void *, const void *));
```

qsort sorts the array *base using the compare function. The array should have num elements of size bytes. The compare function should return a negative value if the first parameter is less than second parameter, zero if the parameters are equal and a positive value if the first parameter is greater than the second parameter.
rand

Synopsis

```
int rand(void);
```

Description

`rand` computes a sequence of pseudo-random integers in the range 0 to `RAND_MAX`.

`rand` returns the computed pseudo-random integer.
realloc

Synopsis

```c
void *realloc(void *p, size_t size);
```

Description

realloc deallocates the old object pointed to by `p` and returns a pointer to a new object that has the size specified by `size`. The contents of the new object is identical to that of the old object prior to deallocation, up to the lesser of the new and old sizes. Any bytes in the new object beyond the size of the old object have indeterminate values.

If `p` is a null pointer, realloc behaves like realloc for the specified size. If memory for the new object cannot be allocated, the old object is not deallocated and its value is unchanged.

realloc returns a pointer to the new object (which may have the same value as a pointer to the old object), or a null pointer if the new object could not be allocated.

If `p` does not match a pointer earlier returned by calloc, malloc, or realloc, or if the space has been deallocated by a call to free or realloc, the behavior is undefined.
srand

Synopsis

```c
void srand(unsigned int seed);
```

Description

The `srand` function is used to set the initial seed value for the pseudo-random number generator. The argument `seed` is used as a seed for a new sequence of pseudo-random numbers to be returned by subsequent calls to `rand`. If `srand` is called with the same seed value, the same sequence of pseudo-random numbers is generated.

If `rand` is called before any calls to `srand` have been made, a sequence is generated as if `srand` is first called with a seed value of 1.

See Also

- `rand`
**strtod**

**Synopsis**

```c
double strtod(const char *nptr,
               char **endptr);
```

**Description**

**strtod** converts the initial portion of the string pointed to by `nptr` to a `double` representation.

First, **strtod** decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by `isspace`), a subject sequence resembling a floating-point constant, and a final string of one or more unrecognized characters, including the terminating null character of the input string. **strtod** then attempts to convert the subject sequence to a floating-point number, and return the result.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

The expected form of the subject sequence is an optional plus or minus sign followed by a nonempty sequence of decimal digits optionally containing a decimal-point character, then an optional exponent part.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated.

A pointer to the final string is stored in the object pointed to by **strtod**, provided that `endptr` is not a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed, the value of `nptr` is stored in the object pointed to by `endptr`, provided that `endptr` is not a null pointer.

**strtod** returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, `HUGE_VAL` is returned according to the sign of the value, if any, and the value of the macro `errno` is stored in `errno`. 
strtof

Synopsis

```c
float strtof(const char *nptr, char **endptr);
```

Description

strtof converts the initial portion of the string pointed to by nptr to a double representation.

First, strtof decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by isspace), a subject sequence resembling a floating-point constant, and a final string of one or more unrecognized characters, including the terminating null character of the input string. strtof then attempts to convert the subject sequence to a floating-point number, and return the result.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

The expected form of the subject sequence is an optional plus or minus sign followed by a nonempty sequence of decimal digits optionally containing a decimal-point character, then an optional exponent part.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed, the value of nptr is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

strtof returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, HUGE_VALF is returned according to the sign of the value, if any, and the value of the macro errno is stored in errno.
strtol

Synopsis

```c
long int strtol(const char *nptr,
              char **endptr,
              int base);
```

Description

strtol converts the initial portion of the string pointed to by nptr to a long int representation.

First, strtol decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by isspace), a subject sequence resembling an integer represented in some radix determined by the value of base, and a final string of one or more unrecognized characters, including the terminating null character of the input string. strtol then attempts to convert the subject sequence to an integer, and return the result.

When converting, no integer suffix (such as U, L, UL, LL, ULL) is allowed.

If the value of base is zero, the expected form of the subject sequence is an optional plus or minus sign followed by an integer constant.

If the value of base is between 2 and 36 (inclusive), the expected form of the subject sequence is an optional plus or minus sign followed by a sequence of letters and digits representing an integer with the radix specified by base. The letters from a (or A) through z (or Z) represent the values 10 through 35; only letters and digits whose ascribed values are less than that of base are permitted.

If the value of base is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the optional sign.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of base is zero, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of base is between 2 and 36, it is used as the base for conversion.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated.

A pointer to the final string is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed, the value of nptr is stored in the object pointed to by endptr, provided that endptr is not a null pointer.
strtol returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, \texttt{LONG\_MIN} or \texttt{LONG\_MAX} is returned according to the sign of the value, if any, and the value of the macro \texttt{errno} is stored in \texttt{errno}. 


strtoll

Synopsis

```c
long long int strftime(const char *nptr,
               char **endptr,
               int base);
```

Description

`strtoll` converts the initial portion of the string pointed to by `nptr` to a `long int` representation.

First, `strtoll` decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by `isspace`), a subject sequence resembling an integer represented in some radix determined by the value of `base`, and a final string of one or more unrecognized characters, including the terminating null character of the input string. `strtoll` then attempts to convert the subject sequence to an integer, and return the result.

When converting, no integer suffix (such as U, L, UL, LL, ULL) is allowed.

If the value of `base` is zero, the expected form of the subject sequence is an optional plus or minus sign followed by an integer constant.

If the value of `base` is between 2 and 36 (inclusive), the expected form of the subject sequence is an optional plus or minus sign followed by a sequence of letters and digits representing an integer with the radix specified by `base`. The letters from a (or A) through z (or Z) represent the values 10 through 35; only letters and digits whose ascribed values are less than that of `base` are permitted.

If the value of `base` is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the optional sign.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of `base` is zero, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of `base` is between 2 and 36, it is used as the base for conversion.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated.

A pointer to the final string is stored in the object pointed to by `endptr`, provided that `endptr` is not a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed, the value of `nptr` is stored in the object pointed to by `endptr`, provided that `endptr` is not a null pointer.
`strtoll` returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, `LLONG_MIN` or `LLONG_MAX` is returned according to the sign of the value, if any, and the value of the macro `ERANGE` is stored in `errno`.
strtoul

Synopsis

```c
unsigned long int strtoul(const char *nptr, char **endptr, int base);
```

Description

strtoul converts the initial portion of the string pointed to by nptr to a long int representation.

First, strtoul decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by isspace), a subject sequence resembling an integer represented in some radix determined by the value of base, and a final string of one or more unrecognized characters, including the terminating null character of the input string. strtoul then attempts to convert the subject sequence to an integer, and return the result.

When converting, no integer suffix (such as U, L, UL, LL, ULL) is allowed.

If the value of base is zero, the expected form of the subject sequence is an optional plus or minus sign followed by an integer constant.

If the value of base is between 2 and 36 (inclusive), the expected form of the subject sequence is an optional plus or minus sign followed by a sequence of letters and digits representing an integer with the radix specified by base. The letters from a (or A) through z (or Z) represent the values 10 through 35; only letters and digits whose ascribed values are less than that of base are permitted.

If the value of base is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the optional sign.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of base is zero, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of base is between 2 and 36, it is used as the base for conversion.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated.

A pointer to the final string is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed, the value of nptr is stored in the object pointed to by endptr, provided that endptr is not a null pointer.
strtol returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, LONG_MAX or ULONG_MAX is returned according to the sign of the value, if any, and the value of the macro ERANGE is stored in errno.
**strtoull**

**Synopsis**

```c
unsigned long long int strtoull(const char *nptr,
                                  char **endptr,
                                  int base);
```

**Description**

**strtoull** converts the initial portion of the string pointed to by **nptr** to a **long int** representation.

First, **strtoull** decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by **isspace**), a subject sequence resembling an integer represented in some radix determined by the value of **base**, and a final string of one or more unrecognized characters, including the terminating null character of the input string. **strtoull** then attempts to convert the subject sequence to an integer, and return the result.

When converting, no integer suffix (such as U, L, UL, LL, ULL) is allowed.

If the value of **base** is zero, the expected form of the subject sequence is an optional plus or minus sign followed by an integer constant.

If the value of **base** is between 2 and 36 (inclusive), the expected form of the subject sequence is an optional plus or minus sign followed by a sequence of letters and digits representing an integer with the radix specified by **base**. The letters from a (or A) through z (or Z) represent the values 10 through 35; only letters and digits whose ascribed values are less than that of **base** are permitted.

If the value of **base** is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the optional sign.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of **base** is zero, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of **base** is between 2 and 36, it is used as the base for conversion.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated.

A pointer to the final string is stored in the object pointed to by **endptr**, provided that **endptr** is not a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed, the value of **nptr** is stored in the object pointed to by **endptr**, provided that **endptr** is not a null pointer.
strtol returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, `LONGLONG_MAX` or `ULLONG_MAX` is returned according to the sign of the value, if any, and the value of the macro `ERANGE` is stored in `errno`. 
ulltoa

Synopsis

```c
char *ulltoa(unsigned long long val,
             char *buf,
             int radix);
```

Description

ulltoa converts val to a string in base radix and places the result in buf.

ulltoa returns buf as the result.

If radix is greater than 36, the result is undefined.

See Also

itoa, ltoa, lltoa, ultoa, utoa
ultoa

Synopsis

```c
char *ultoa(unsigned long val,
            char *buf,
            int radix);
```

Description

ultoa converts val to a string in base radix and places the result in buf.

ultoa returns buf as the result.

If radix is greater than 36, the result is undefined.

See Also

itoa, ltoa, lltoa, ulltoa, utoa
utoa

Synopsis

```c
char *utoa(unsigned val,
        char *buf,
        int radix);
```

Description

Utoa converts val to a string in base radix and places the result in buf.

Utoa returns buf as the result.

If radix is greater than 36, the result is undefined.

See Also

itoa, ltoa, lltoa, ultoa, ulltoa
<string.h>

Overview

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.

API Summary

<table>
<thead>
<tr>
<th>Copying functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memccpy</td>
<td>Copy memory with specified terminator (POSIX extension)</td>
</tr>
<tr>
<td>memcpy</td>
<td>Copy memory</td>
</tr>
<tr>
<td>memcpy_fast</td>
<td>Copy memory</td>
</tr>
<tr>
<td>memmove</td>
<td>Safely copy overlapping memory</td>
</tr>
<tr>
<td>mempcpy</td>
<td>Copy memory (GNU extension)</td>
</tr>
<tr>
<td>strcat</td>
<td>Concatenate strings</td>
</tr>
<tr>
<td>strcpy</td>
<td>Copy string</td>
</tr>
<tr>
<td>strdup</td>
<td>Duplicate string (POSIX extension)</td>
</tr>
<tr>
<td>strncat</td>
<td>Copy string up to a maximum length with terminator (BSD extension)</td>
</tr>
<tr>
<td>strncpy</td>
<td>Copy string up to a maximum length with terminator (BSD extension)</td>
</tr>
<tr>
<td>strncat</td>
<td>Concatenate strings up to maximum length</td>
</tr>
<tr>
<td>strncpy</td>
<td>Copy string up to a maximum length</td>
</tr>
<tr>
<td>strndup</td>
<td>Duplicate string (POSIX extension)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Comparison functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memcmp</td>
<td>Compare memory</td>
</tr>
<tr>
<td>strcasecmp</td>
<td>Compare strings ignoring case (POSIX extension)</td>
</tr>
</tbody>
</table>
### strcmp
Compare strings

### strncasecmp
Compare strings up to a maximum length ignoring case (POSIX extension)

### strncmp
Compare strings up to a maximum length

#### Search functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memchr</td>
<td>Search memory for a character</td>
</tr>
<tr>
<td>strcasestr</td>
<td>Find first case-insensitive occurrence of a string within string</td>
</tr>
<tr>
<td>strchr</td>
<td>Find character within string</td>
</tr>
<tr>
<td>strcsppn</td>
<td>Compute size of string not prefixed by a set of characters</td>
</tr>
<tr>
<td>strncasestr</td>
<td>Find first case-insensitive occurrence of a string within length-limited string</td>
</tr>
<tr>
<td>strncstr</td>
<td>Find character in a length-limited string</td>
</tr>
<tr>
<td>strnlen</td>
<td>Calculate length of length-limited string (POSIX extension)</td>
</tr>
<tr>
<td>strnstr</td>
<td>Find first occurrence of a string within length-limited string</td>
</tr>
<tr>
<td>strpbrk</td>
<td>Find first occurrence of characters within string</td>
</tr>
<tr>
<td>strrrchr</td>
<td>Find last occurrence of character within string</td>
</tr>
<tr>
<td>strsep</td>
<td>Break string into tokens (4.4BSD extension)</td>
</tr>
<tr>
<td>strspn</td>
<td>Compute size of string prefixed by a set of characters</td>
</tr>
<tr>
<td>strstr</td>
<td>Find first occurrence of a string within string</td>
</tr>
<tr>
<td>strtok</td>
<td>Break string into tokens</td>
</tr>
<tr>
<td>strtok_r</td>
<td>Break string into tokens, reentrant version (POSIX extension)</td>
</tr>
</tbody>
</table>

#### Miscellaneous functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memset</td>
<td>Set memory to character</td>
</tr>
<tr>
<td>strerror</td>
<td>Decode error code</td>
</tr>
<tr>
<td>strlen</td>
<td>Calculate length of string</td>
</tr>
</tbody>
</table>
memccpy

Synopsis

```c
void *memccpy(void *s1,
        const void *s2,
        int c,
        size_t n);
```

Description

memccpy copies at most n characters from the object pointed to by s2 into the object pointed to by s1. The copying stops as soon as n characters are copied or the character c is copied into the destination object pointed to by s1. The behavior of memccpy is undefined if copying takes place between objects that overlap.

memccpy returns a pointer to the character immediately following c in s1, or NULL if c was not found in the first n characters of s2.

Note

memccpy conforms to POSIX.1-2008.
memchr

Synopsis

```
void *memchr(const void *s, int c, size_t n);
```

Description

`memchr` locates the first occurrence of `c` (converted to an `unsigned char`) in the initial `n` characters (each interpreted as `unsigned char`) of the object pointed to by `s`. Unlike `strchr`, `memchr` does not terminate a search when a null character is found in the object pointed to by `s`.

`memchr` returns a pointer to the located character, or a null pointer if `c` does not occur in the object.
**memcmp**

**Synopsis**

```c
int memcmp(const void *s1, const void *s2, size_t n);
```

**Description**

`memcmp` compares the first `n` characters of the object pointed to by `s1` to the first `n` characters of the object pointed to by `s2`. `memcmp` returns an integer greater than, equal to, or less than zero as the object pointed to by `s1` is greater than, equal to, or less than the object pointed to by `s2`. 
memcpy

Synopsis

```c
void *memcpy(void *s1,
             const void *s2,
             size_t n);
```

Description

memcpy copies n characters from the object pointed to by s2 into the object pointed to by s1. The behavior of memcpy is undefined if copying takes place between objects that overlap.

memcpy returns the value of s1.
memcpy_fast

Synopsis

```c
void *memcpy_fast(void *s1,
                  const void *s2,
                  size_t n);
```

Description

memcpy_fast copies n characters from the object pointed to by s2 into the object pointed to by s1. The behavior of memcpy_fast is undefined if copying takes place between objects that overlap. The implementation of memcpy_fast is optimized for speed for all cases of memcpy and as such has a large code memory requirement. This function is implemented for little-endian ARM and 32-bit Thumb-2 instruction sets only.

memcpy_fast returns the value of s1.
memmove

Synopsis

```c
void *memmove(void *s1,
               const void *s2,
               size_t n);
```

Description

`memmove` copies `n` characters from the object pointed to by `s2` into the object pointed to by `s1` ensuring that if `s1` and `s2` overlap, the copy works correctly. Copying takes place as if the `n` characters from the object pointed to by `s2` are first copied into a temporary array of `n` characters that does not overlap the objects pointed to by `s1` and `s2`, and then the `n` characters from the temporary array are copied into the object pointed to by `s1`.

`memmove` returns the value of `s1`. 
mempcpy

Synopsis

```c
void *mempcpy(void *s1,
               const void *s2,
               size_t n);
```

Description

mempcpy copies \( n \) characters from the object pointed to by \( s2 \) into the object pointed to by \( s1 \). The behavior of mempcpy is undefined if copying takes place between objects that overlap.

mempcpy returns a pointer to the byte following the last written byte.

Note

This is an extension found in GNU libc.
### memset

**Synopsis**

```c
void *memset(void *s, int c, size_t n);
```

**Description**

`memset` copies the value of `c` (converted to an `unsigned char`) into each of the first `n` characters of the object pointed to by `s`.

`memset` returns the value of `s`.


strcascmp

Synopsis

```c
int strcasecmp(const char *s1,
               const char *s2);
```

Description

strcascmp compares the string pointed to by s1 to the string pointed to by s2 ignoring differences in case. strcascmp returns an integer greater than, equal to, or less than zero if the string pointed to by s1 is greater than, equal to, or less than the string pointed to by s2.

Note

strcascmp conforms to POSIX.1-2008.
strcasestr

Synopsis

```c
char *strcasestr(const char *s1,
                 const char *s2);
```

Description

`strcasestr` locates the first occurrence in the string pointed to by `s1` of the sequence of characters (excluding the terminating null character) in the string pointed to by `s2` without regard to character case.

`strcasestr` returns a pointer to the located string, or a null pointer if the string is not found. If `s2` points to a string with zero length, `strcasestr` returns `s1`.

Note

`strcasestr` is an extension commonly found in Linux and BSD C libraries.
**strcat**

**Synopsis**

```c
char *strcat(char *s1,
             const char *s2);
```

**Description**

`strcat` appends a copy of the string pointed to by `s2` (including the terminating null character) to the end of the string pointed to by `s1`. The initial character of `s2` overwrites the null character at the end of `s1`. The behavior of `strcat` is undefined if copying takes place between objects that overlap.

`strcat` returns the value of `s1`. 
strchr

Synopsis

```c
char *strchr(const char *s, int c);
```

Description

`strchr` locates the first occurrence of `c` (converted to a `char`) in the string pointed to by `s`. The terminating null character is considered to be part of the string.

`strchr` returns a pointer to the located character, or a null pointer if `c` does not occur in the string.
**strcmpe**

**Synopsis**

```c
int strcmp(const char *s1,
           const char *s2);
```

**Description**

`strcmpe` compares the string pointed to by `s1` to the string pointed to by `s2`. `strcmpe` returns an integer greater than, equal to, or less than zero if the string pointed to by `s1` is greater than, equal to, or less than the string pointed to by `s2`. 
strcpy

Synopsis

```c
char *strcpy(char *s1, const char *s2);
```

Description

`strcpy` copies the string pointed to by `s2` (including the terminating null character) into the array pointed to by `s1`. The behavior of `strcpy` is undefined if copying takes place between objects that overlap.

`strcpy` returns the value of `s1`. 
strcspn

Synopsis

```c
size_t strcspn(const char *s1,
               const char *s2);
```

Description

strcspn computes the length of the maximum initial segment of the string pointed to by s1 which consists entirely of characters not from the string pointed to by s2.

strcspn returns the length of the segment.
strdup

Synopsis

```c
char *strdup(const char *s1);
```

Description

`strdup` duplicates the string pointed to by `s1` by using `malloc` to allocate memory for a copy of `s` and then copying `s`, including the terminating null, to that memory. `strdup` returns a pointer to the new string or a null pointer if the new string cannot be created. The returned pointer can be passed to `free`.

Note

`strdup` conforms to POSIX.1-2008 and SC22 TR 24731-2.
strerror

Synopsis

```c
char *strerror(int num);
```

Description

`strerror` maps the number in `num` to a message string. Typically, the values for `num` come from `errno`, but `strerror` can map any value of type `int` to a message.

`strerror` returns a pointer to the message string. The program must not modify the returned message string. The message may be overwritten by a subsequent call to `strerror`. 
strlcat

Synopsis

```c
size_t strlcat(char *s1,
               const char *s2,
               size_t n);
```

Description

strlcat appends no more than `nstrlen(dst)`1 characters pointed to by `s2` into the array pointed to by `s1` and always terminates the result with a null character if `n` is greater than zero. Both the strings `s1` and `s2` must be terminated with a null character on entry to `strlcat` and a byte for the terminating null should be included in `n`. The behavior of `strlcat` is undefined if copying takes place between objects that overlap.

strlcat returns the number of characters it tried to copy, which is the sum of the lengths of the strings `s1` and `s2` or `n`, whichever is smaller.

Note

strlcat is commonly found in OpenBSD libraries.
strlcpy

Synopsis

```c
size_t strlcpy(char *s1,
               const char *s2,
               size_t n);
```

Description

`strlcpy` copies up to `n` characters from the string pointed to by `s2` into the array pointed to by `s1` and always terminates the result with a null character. The behavior of `strlcpy` is undefined if copying takes place between objects that overlap.

`strlcpy` returns the number of characters it tried to copy, which is the length of the string `s2` or `n`, whichever is smaller.

Note

`strlcpy` is commonly found in OpenBSD libraries and contrasts with `strncpy` in that the resulting string is always terminated with a null character.
strlen

Synopsis

```c
size_t strlen(const char *s);
```

Description

`strlen` returns the length of the string pointed to by `s`, that is the number of characters that precede the terminating null character.
strncasecmp

Synopsis

```c
int strncasecmp(const char *s1,
                const char *s2,
                size_t n);
```

Description

strncasecmp compares not more than n characters from the array pointed to by s1 to the array pointed to by s2 ignoring differences in case. Characters that follow a null character are not compared.

strncasecmp returns an integer greater than, equal to, or less than zero, if the possibly null-terminated array pointed to by s1 is greater than, equal to, or less than the possibly null-terminated array pointed to by s2.

Note

strncasecmp conforms to POSIX.1-2008.
strncasestr

Synopsis

```c
char *strncasestr(const char *s1,
                  const char *s2,
                  size_t n);
```

Description

`strncasestr` searches at most `n` characters to locate the first occurrence in the string pointed to by `s1` of the sequence of characters (excluding the terminating null character) in the string pointed to by `s2` without regard to character case.

`strncasestr` returns a pointer to the located string, or a null pointer if the string is not found. If `s2` points to a string with zero length, `strncasestr` returns `s1`.

Note

`strncasestr` is an extension commonly found in Linux and BSD C libraries.
strncat

Synopsis

```c
char *strncat(char *s1,
             const char *s2,
             size_t n);
```

Description

strncat appends not more than n characters from the array pointed to by s2 to the end of the string pointed to by s1. A null character in s1 and characters that follow it are not appended. The initial character of s2 overwrites the null character at the end of s1. A terminating null character is always appended to the result. The behavior of strncat is undefined if copying takes place between objects that overlap.

strncat returns the value of s1.
strnchr

Synopsis

```
char *strnchr(const char *str,
        size_t n,
        int ch);
```

Description

strnchr searches not more than n characters to locate the first occurrence of c (converted to a char) in the string pointed to by s. The terminating null character is considered to be part of the string.

strnchr returns a pointer to the located character, or a null pointer if c does not occur in the string.
strncmp

Synopsis

```c
int strncmp(const char *s1,
            const char *s2,
            size_t n);
```

Description

`strncmp` compares not more than `n` characters from the array pointed to by `s1` to the array pointed to by `s2`. Characters that follow a null character are not compared.

`strncmp` returns an integer greater than, equal to, or less than zero, if the possibly null-terminated array pointed to by `s1` is greater than, equal to, or less than the possibly null-terminated array pointed to by `s2`.
**strncpy**

**Synopsis**

```c
char *strncpy(char *s1,
              const char *s2,
              size_t n);
```

**Description**

`strncpy` copies not more than `n` characters from the array pointed to by `s2` to the array pointed to by `s1`. Characters that follow a null character in `s2` are not copied. The behavior of `strncpy` is undefined if copying takes place between objects that overlap. If the array pointed to by `s2` is a string that is shorter than `n` characters, null characters are appended to the copy in the array pointed to by `s1`, until `n` characters in all have been written.

`strncpy` returns the value of `s1`.

**Note**

No null character is implicitly appended to the end of `s1`, so `s1` will only be terminated by a null character if the length of the string pointed to by `s2` is less than `n`. 
strndup

Synopsis

```c
char *strndup(const char *s1, size_t n);
```

Description

strndup duplicates at most n characters from the the string pointed to by s1 by using malloc to allocate memory for a copy of s1.

If the length of string pointed to by s1 is greater than n characters, only n characters will be duplicated. If n is greater than the length of string pointed to by s1, all characters in the string are copied into the allocated array including the terminating null character.

strndup returns a pointer to the new string or a null pointer if the new string cannot be created. The returned pointer can be passed to free.

Note

strndup conforms to POSIX.1-2008 and SC22 TR 24731-2.
strnlen

Synopsis

```c
size_t strnlen(const char *s,
               size_t n);
```

Description

strnlen returns the length of the string pointed to by s, up to a maximum of n characters. strnlen only examines the first n characters of the string s.

Note

strnlen conforms to POSIX.1-2008.
**strstr**

**Synopsis**

```c
char *strstr(const char *s1,
            const char *s2,
            size_t n);
```

**Description**

`strstr` searches at most `n` characters to locate the first occurrence in the string pointed to by `s1` of the sequence of characters (excluding the terminating null character) in the string pointed to by `s2`.

`strstr` returns a pointer to the located string, or a null pointer if the string is not found. If `s2` points to a string with zero length, `strstr` returns `s1`.

**Note**

`strstr` is an extension commonly found in Linux and BSD C libraries.
strpbrk

Synopsis

```
char *strpbrk(const char *s1,
        const char *s2);
```

Description

strpbrk locates the first occurrence in the string pointed to by s1 of any character from the string pointed to by s2.

strpbrk returns a pointer to the character, or a null pointer if no character from s2 occurs in s1.
**strrchr**

**Synopsis**

```c
char *strrchr(const char *s, int c);
```

**Description**

`strrchr` locates the last occurrence of `c` (converted to a `char`) in the string pointed to by `s`. The terminating null character is considered to be part of the string.

`strrchr` returns a pointer to the character, or a null pointer if `c` does not occur in the string.
strsep

Synopsis

```c
char *strsep(char **stringp, const char *delim);
```

Description

`strsep` locates, in the string referenced by `*stringp`, the first occurrence of any character in the string `delim` (or the terminating null character) and replaces it with a null character. The location of the next character after the delimiter character (or NULL, if the end of the string was reached) is stored in `*stringp`. The original value of `*stringp` is returned.

An empty field (that is, a character in the string `delim` occurs as the first character of `*stringp`) can be detected by comparing the location referenced by the returned pointer to the null character.

If `*stringp` is initially null, `strsep` returns null.

Note

`strsep` is an extension commonly found in Linux and BSD C libraries.
**strspn**

**Synopsis**

```c
size_t strspn(const char *s1,
              const char *s2);
```

**Description**

`strspn` computes the length of the maximum initial segment of the string pointed to by `s1` which consists entirely of characters from the string pointed to by `s2`.

`strspn` returns the length of the segment.
**strstr**

**Synopsis**

```c
char *strstr(const char *s1,
            const char *s2);
```

**Description**

`strstr` locates the first occurrence in the string pointed to by `s1` of the sequence of characters (excluding the terminating null character) in the string pointed to by `s2`.

`strstr` returns a pointer to the located string, or a null pointer if the string is not found. If `s2` points to a string with zero length, `strstr` returns `s1`. 
strtok

Synopsis

```c
char *strtok(char *s1,
              const char *s2);
```

Description

strtok A sequence of calls to `strtok` breaks the string pointed to by `s1` into a sequence of tokens, each of which is delimited by a character from the string pointed to by `s2`. The first call in the sequence has a non-null first argument; subsequent calls in the sequence have a null first argument. The separator string pointed to by `s2` may be different from call to call.

The first call in the sequence searches the string pointed to by `s1` for the first character that is not contained in the current separator string pointed to by `s2`. If no such character is found, then there are no tokens in the string pointed to by `s1` and `strtok` returns a null pointer. If such a character is found, it is the start of the first token.

`strtok` then searches from there for a character that is contained in the current separator string. If no such character is found, the current token extends to the end of the string pointed to by `s1`, and subsequent searches for a token will return a null pointer. If such a character is found, it is overwritten by a null character, which terminates the current token. `strtok` saves a pointer to the following character, from which the next search for a token will start.

Each subsequent call, with a null pointer as the value of the first argument, starts searching from the saved pointer and behaves as described above.

Note

`strtok` maintains static state and is therefore not reentrant and not thread safe. See `strtok_r` for a thread-safe and reentrant variant.

See Also

`strsep`, `strtok_r`
strtok_r

Synopsis

```c
char *strtok_r(char *s1,
               const char *s2,
               char **s3);
```

Description

strtok_r is a reentrant version of the function strtok where the state is maintained in the object of type char * pointed to by s3.

Note

strtok_r conforms to POSIX.1-2008 and is commonly found in Linux and BSD C libraries.

See Also

strtok.
<time.h>

API Summary

<table>
<thead>
<tr>
<th>Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock_t</td>
<td>Clock type</td>
</tr>
<tr>
<td>time_t</td>
<td>Time type</td>
</tr>
<tr>
<td>tm</td>
<td>Time structure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asctime</td>
<td>Convert a struct tm to a string</td>
</tr>
<tr>
<td>asctime_r</td>
<td>Convert a struct tm to a string</td>
</tr>
<tr>
<td>ctime</td>
<td>Convert a time_t to a string</td>
</tr>
<tr>
<td>ctime_r</td>
<td>Convert a time_t to a string</td>
</tr>
<tr>
<td>difftime</td>
<td>Calculates the difference between two times</td>
</tr>
<tr>
<td>gmtime</td>
<td>Convert a time_t to a struct tm</td>
</tr>
<tr>
<td>gmtime_r</td>
<td>Convert a time_t to a struct tm</td>
</tr>
<tr>
<td>localtime</td>
<td>Convert a time_t to a struct tm</td>
</tr>
<tr>
<td>localtime_r</td>
<td>Convert a time_t to a struct tm</td>
</tr>
<tr>
<td>mktime</td>
<td>Convert a struct tm to time_t</td>
</tr>
<tr>
<td>strftime</td>
<td>Format a struct tm to a string</td>
</tr>
</tbody>
</table>
asctime

Synopsis

```c
char *asctime(const tm *tp);
```

Description

asctime converts the *tp struct to a null terminated string of the form Sun Sep 16 01:03:52 1973. The returned string is held in a static buffer. asctime is not re-entrant.
asctime_r

Synopsis

```c
char *asctime_r(const tm *tp, char *buf);
```

Description

asctime_r converts the *tp struct to a null terminated string of the form Sun Sep 16 01:03:52 1973 in buf and returns buf. The buf must point to an array at least 26 bytes in length.
clock_t

Synopsis

typedef long clock_t;

Description

clock_t is the type returned by the clock function.
ctime

Synopsis

```
char *ctime(const time_t *tp);
```

Description

ctime converts the *tp to a null terminated string. The returned string is held in a static buffer, this function is not re-entrant.
ctime_r

Synopsis

```c
char *ctime_r(const time_t *tp,
              char *buf);
```

Description

cctime_r converts the *tp to a null terminated string in buf and returns buf. The buf must point to an array at least 26 bytes in length.
difftime

Synopsis

```c
double difftime(time_t time2,
    time_t time1);
```

Description

difftime returns time1 - time0 as a double precision number.
gmtime

Synopsis

```c
gmtime(const time_t *tp);
```

Description

gmtime converts the *tp time format to a struct tm time format. The returned value points to a static object - this function is not re-entrant.
gmtime_r

Synopsis

```c
gmtime_r(const time_t *tp,
         tm *result);
```

Description

`gmtime_r` converts the `*tp` time format to a `struct tm` time format in `*result` and returns `result`. 
localtime

Synopsis

```c
localtime(const time_t *tp);
```

Description

`localtime` converts the `*tp` time format to a `struct tm` local time format. The returned value points to a static object - this function is not re-entrant.
localtime_r

Synopsis

```c
localtime_r(const time_t *tp,
            tm *result);
```

Description

`localtime_r` converts the `*tp` time format to a `struct tm` local time format in `*result` and returns `result`. 
**mktime**

**Synopsis**

```c
#include <time.h>

time_t mktime(tm *tp);
```

**Description**

The `mktime` function validates (and updates) the *tp struct to ensure that the `tm_sec`, `tm_min`, `tm_hour`, `tm_mon` fields are within the supported integer ranges and the `tm_mday`, `tm_mon` and `tm_year` fields are consistent. The validated `*tp` struct is converted to the number of seconds since UTC 1 January 1970 and returned.
strft ime

Synopsis

```
size_t strftime(char *s,
size_t smax,
const char *fmt,
const tm *tp);
```

Description

strftime formats the *tp struct to a null terminated string of maximum size smax-1 into the array at *s based on the fmt format string. The format string consists of conversion specifications and ordinary characters. Conversion specifications start with a % character followed by an optional # character. The following conversion specifications are supported:

<table>
<thead>
<tr>
<th>Specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%a</td>
<td>Abbreviated weekday name</td>
</tr>
<tr>
<td>%A</td>
<td>Full weekday name</td>
</tr>
<tr>
<td>%b</td>
<td>Abbreviated month name</td>
</tr>
<tr>
<td>%B</td>
<td>Full month name</td>
</tr>
<tr>
<td>%c</td>
<td>Date and time representation appropriate for locale</td>
</tr>
<tr>
<td>%#c</td>
<td>Date and time formatted as &quot;%A, %B %#d, %Y, %H:%M:%S&quot; (Microsoft extension)</td>
</tr>
<tr>
<td>%C</td>
<td>Century number</td>
</tr>
<tr>
<td>%d</td>
<td>Day of month as a decimal number [01,31]</td>
</tr>
<tr>
<td>%#d</td>
<td>Day of month without leading zero [1,31]</td>
</tr>
<tr>
<td>%D</td>
<td>Date in the form %m/%d/%y (POSIX.1-2008 extension)</td>
</tr>
<tr>
<td>%e</td>
<td>Day of month [ 1,31], single digit preceded by space</td>
</tr>
<tr>
<td>%F</td>
<td>Date in the format %Y-%m-%d</td>
</tr>
<tr>
<td>%h</td>
<td>Abbreviated month name as %b</td>
</tr>
<tr>
<td>%H</td>
<td>Hour in 24-hour format [00,23]</td>
</tr>
<tr>
<td>%#H</td>
<td>Hour in 24-hour format without leading zeros [0,23]</td>
</tr>
<tr>
<td>%I</td>
<td>Hour in 12-hour format [01,12]</td>
</tr>
<tr>
<td>%#I</td>
<td>Hour in 12-hour format without leading zeros [1,12]</td>
</tr>
<tr>
<td>%j</td>
<td>Day of year as a decimal number [001,366]</td>
</tr>
<tr>
<td>%#j</td>
<td>Day of year as a decimal number without leading zeros [1,366]</td>
</tr>
<tr>
<td>%k</td>
<td>Hour in 24-hour clock format [ 0,23] (POSIX.1-2008 extension)</td>
</tr>
<tr>
<td>Format</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>%l</td>
<td>Hour in 12-hour clock format [0,12] (POSIX.1-2008 extension)</td>
</tr>
<tr>
<td>%m</td>
<td>Month as a decimal number [01,12]</td>
</tr>
<tr>
<td>%#m</td>
<td>Month as a decimal number without leading zeros [1,12]</td>
</tr>
<tr>
<td>%M</td>
<td>Minute as a decimal number [00,59]</td>
</tr>
<tr>
<td>%#M</td>
<td>Minute as a decimal number without leading zeros [0,59]</td>
</tr>
<tr>
<td>%n</td>
<td>Insert newline character (POSIX.1-2008 extension)</td>
</tr>
<tr>
<td>%p</td>
<td>Locale's a.m or p.m indicator for 12-hour clock</td>
</tr>
<tr>
<td>%r</td>
<td>Time as %I:%M:%S %p (POSIX.1-2008 extension)</td>
</tr>
<tr>
<td>%R</td>
<td>Time as %H:%M (POSIX.1-2008 extension)</td>
</tr>
<tr>
<td>%S</td>
<td>Second as a decimal number [00,59]</td>
</tr>
<tr>
<td>%#S</td>
<td>Second as a decimal number without leading zeros [0,59]</td>
</tr>
<tr>
<td>%U</td>
<td>Week of year as a decimal number [00,53], Sunday is first day of the week</td>
</tr>
<tr>
<td>%#U</td>
<td>Week of year as a decimal number without leading zeros [0,53], Sunday is first day of the week</td>
</tr>
<tr>
<td>%w</td>
<td>Weekday as a decimal number [0,6], Sunday is 0</td>
</tr>
<tr>
<td>%W</td>
<td>Week number as a decimal number [00,53], Monday is first day of the week</td>
</tr>
<tr>
<td>%#W</td>
<td>Week number as a decimal number without leading zeros [0,53], Monday is first day of the week</td>
</tr>
<tr>
<td>%x</td>
<td>Locale's date representation</td>
</tr>
<tr>
<td>%#x</td>
<td>Locale's long date representation</td>
</tr>
<tr>
<td>%X</td>
<td>Locale's time representation</td>
</tr>
<tr>
<td>%y</td>
<td>Year without century, as a decimal number [00,99]</td>
</tr>
<tr>
<td>%#y</td>
<td>Year without century, as a decimal number without leading zeros [0,99]</td>
</tr>
<tr>
<td>%Y</td>
<td>Year with century, as decimal number</td>
</tr>
<tr>
<td>%z,%Z</td>
<td>Timezone name or abbreviation</td>
</tr>
<tr>
<td>%%</td>
<td>%</td>
</tr>
</tbody>
</table>
time_t

Synopsis

```c
typedef long time_t;
```

Description

`time_t` is a long type that represents the time in number of seconds since UTC 1 January 1970, negative values indicate time before UTC 1 January 1970.
### Synopsis

```c
typedef struct {
    int tm_sec;
    int tm_min;
    int tm_hour;
    int tm_mday;
    int tm_mon;
    int tm_year;
    int tm_wday;
    int tm_yday;
    int tm_isdst;
} tm;
```

### Description

The `tm` structure has the following fields:

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tm_sec</td>
<td>seconds after the minute - [0,59]</td>
</tr>
<tr>
<td>tm_min</td>
<td>minutes after the hour - [0,59]</td>
</tr>
<tr>
<td>tm_hour</td>
<td>hours since midnight - [0,23]</td>
</tr>
<tr>
<td>tm_mday</td>
<td>day of the month - [1,31]</td>
</tr>
<tr>
<td>tm_mon</td>
<td>months since January - [0,11]</td>
</tr>
<tr>
<td>tm_year</td>
<td>years since 1900</td>
</tr>
<tr>
<td>tm_wday</td>
<td>days since Sunday - [0,6]</td>
</tr>
<tr>
<td>tm_yday</td>
<td>days since January 1 - [0,365]</td>
</tr>
<tr>
<td>tm_isdst</td>
<td>daylight savings time flag</td>
</tr>
</tbody>
</table>
### <wchar.h>

#### API Summary

<table>
<thead>
<tr>
<th>Character minimum and maximum values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WCHAR_MAX</strong></td>
</tr>
<tr>
<td><strong>WCHAR_MIN</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constants</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WEOF</strong></td>
<td>End of file indication</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Types</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>wchar_t</strong></td>
<td>Wide character type</td>
</tr>
<tr>
<td><strong>wint_t</strong></td>
<td>Wide integer type</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Copying functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>wcscat</strong></td>
<td>Concatenate strings</td>
</tr>
<tr>
<td><strong>wcsncpy</strong></td>
<td>Copy string</td>
</tr>
<tr>
<td><strong>wcsncat</strong></td>
<td>Concatenate strings up to maximum length</td>
</tr>
<tr>
<td><strong>wcsncpy</strong></td>
<td>Copy string up to a maximum length</td>
</tr>
<tr>
<td><strong>wmemccpy</strong></td>
<td>Copy memory with specified terminator (POSIX extension)</td>
</tr>
<tr>
<td><strong>wmemcpy</strong></td>
<td>Copy memory</td>
</tr>
<tr>
<td><strong>wmemmove</strong></td>
<td>Safely copy overlapping memory</td>
</tr>
<tr>
<td><strong>wmempcpy</strong></td>
<td>Copy memory (GNU extension)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Comparison functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>wcsncmp</strong></td>
<td>Compare strings up to a maximum length</td>
</tr>
<tr>
<td><strong>wmemcmp</strong></td>
<td>Compare memory</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Search functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>wcschr</strong></td>
<td>Find character within string</td>
</tr>
<tr>
<td><strong>wcscspn</strong></td>
<td>Compute size of string not prefixed by a set of characters</td>
</tr>
<tr>
<td><strong>wcsnchr</strong></td>
<td>Find character in a length-limited string</td>
</tr>
<tr>
<td><strong>wcsnlen</strong></td>
<td>Calculate length of length-limited string</td>
</tr>
<tr>
<td><strong>wcsnstr</strong></td>
<td>Find first occurrence of a string within length-limited string</td>
</tr>
<tr>
<td><strong>wcspbrk</strong></td>
<td>Find first occurrence of characters within string</td>
</tr>
<tr>
<td><strong>wcsrchr</strong></td>
<td>Find last occurrence of character within string</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------</td>
</tr>
<tr>
<td>wcsspn</td>
<td>Compute size of string prefixed by a set of characters</td>
</tr>
<tr>
<td>wcsstr</td>
<td>Find first occurrence of a string within string</td>
</tr>
<tr>
<td>wcstok</td>
<td>Break string into tokens</td>
</tr>
<tr>
<td>wcstok_r</td>
<td>Break string into tokens (reentrant version)</td>
</tr>
<tr>
<td>wmemchr</td>
<td>Search memory for a wide character</td>
</tr>
<tr>
<td>wstrsep</td>
<td>Break string into tokens</td>
</tr>
</tbody>
</table>

**Miscellaneous functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wcsdup</td>
<td>Duplicate string</td>
</tr>
<tr>
<td>wcslen</td>
<td>Calculate length of string</td>
</tr>
<tr>
<td>wmemset</td>
<td>Set memory to wide character</td>
</tr>
</tbody>
</table>

**Multi-byte/wide string conversion functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbtowc</td>
<td>Convert multi-byte character to wide character</td>
</tr>
<tr>
<td>mbtowc_l</td>
<td>Convert multi-byte character to wide character</td>
</tr>
<tr>
<td>msbinit</td>
<td>Query conversion state</td>
</tr>
<tr>
<td>wcrtomb</td>
<td>Convert wide character to multi-byte character (restartable)</td>
</tr>
<tr>
<td>wcrtomb_l</td>
<td>Convert wide character to multi-byte character (restartable)</td>
</tr>
<tr>
<td>wctob</td>
<td>Convert wide character to single-byte character</td>
</tr>
<tr>
<td>wctob_l</td>
<td>Convert wide character to single-byte character</td>
</tr>
</tbody>
</table>

**Multi-byte to wide character conversions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbrlen</td>
<td>Determine number of bytes in a multi-byte character</td>
</tr>
<tr>
<td>mbrlen_l</td>
<td>Determine number of bytes in a multi-byte character</td>
</tr>
<tr>
<td>mbsrtowcs</td>
<td>Convert multi-byte string to wide character string</td>
</tr>
<tr>
<td>mbsrtowcs_l</td>
<td>Convert multi-byte string to wide character string</td>
</tr>
</tbody>
</table>

**Single-byte to wide character conversions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>btowc</td>
<td>Convert single-byte character to wide character</td>
</tr>
<tr>
<td>btowc_l</td>
<td>Convert single-byte character to wide character</td>
</tr>
</tbody>
</table>
WCHAR_MAX

Synopsis

#define WCHAR_MAX ...

Description

WCHAR_MAX is the maximum value for an object of type wchar_t. Although capable of storing larger values, the maximum value implemented by the conversion functions in the library is the value 0x10FFFF defined by ISO 10646.
**WCHAR_MIN**

**Synopsis**

```c
#define WCHAR_MIN ...
```

**Description**

`WCHAR_MIN` is the minimum value for an object of type `wchar_t`. 
WEOF

Synopsis

#define WEOF ((wint_t)-0U)

Description

WEOF expands to a constant value that does not correspond to any character in the wide character set. It is typically used to indicate an end of file condition.
btowc

Synopsis

```c
wint_t btowc(int c);
```

Description

The `btowc` function determines whether `c` constitutes a valid single-byte character. If `c` is a valid single-byte character, `btowc` returns the wide character representation of that character. If `c` has the value `EOF` or if `(unsigned char)c` does not constitute a valid single-byte character in the initial shift state, `btowc` returns `WEOF`. 
btowc_l

Synopsis

```
wint_t btowc_l(int c,
              locale_t loc);
```

Description

The `btowc_l` function determines whether `c` constitutes a valid single-byte character in the locale `loc`. If `c` is a valid single-byte character, `btowc_l` returns the wide character representation of that character.

`btowc_l` returns WEOF if `c` has the value EOF or if `(unsigned char)c` does not constitute a valid single-byte character in the initial shift state.
**mbrlen**

**Synopsis**

```c
size_t mbrlen(const char *s,
             size_t n,
             mbstate_t *ps);
```

**Note**

The `mbrlen` function is equivalent to the call:

```c
mbrtowc(NULL, s, n, ps != NULL ? ps : &internal);
```

where `internal` is the `mbstate_t` object for the `mbrlen` function, except that the expression designated by `ps` is evaluated only once.
mbrlen_l

Synopsis

```c
size_t mbrlen_l(const char *s,
               size_t n,
               mbstate_t *ps,
               locale_t loc);
```

Note

mbrlen_l function is equivalent to the call:

```c
mbrtowc_l(NULL, s, n, ps != NULL ? ps : &internal, loc);
```

where internal is the mbstate_t object for the mbrlen function, except that the expression designated by ps is evaluated only once.
mbrtowc

Synopsis

```c
size_t mbrtowc(wchar_t *pwc,
    const char *s,
    size_t n,
    mbstate_t *ps);
```

Description

mbrtowc converts a single multi-byte character to a wide character in the current locale.

If `s` is a null pointer, `mbrtowc` is equivalent to `mbrtowc(NULL, "", 1, ps)`, ignoring `pwc` and `n`.

If `s` is not null and the object that `s` points to is a wide-character null character, `mbrtowc` returns 0.

If `s` is not null and the object that points to forms a valid multi-byte character with a most `n` bytes, `mbrtowc` returns the length in bytes of the multi-byte character and stores that wide character to the object pointed to by `pwc` (if `pwc` is not null).

If the object that points to forms an incomplete, but possibly valid, multi-byte character, `mbrtowc` returns 2. If the object that points to does not form a partial multi-byte character, `mbrtowc` returns 1.

See Also

mbtowc, mbrtowc_l
**mbrtowc_l**

**Synopsis**

```c
size_t mbtowc_l(wchar_t *pwc, const char *s, size_t n, mbstate_t *ps, locale_t loc);
```

**Description**

`mbtowc_l` converts a single multi-byte character to a wide character in the locale `loc`.

If `s` is a null pointer, `mbtowc_l` is equivalent to `mbtowc(NULL, "", 1, ps)`, ignoring `pwc` and `n`.

If `s` is not null and the object that `s` points to is a wide-character null character, `mbtowc_l` returns 0.

If `s` is not null and the object that points to forms a valid multi-byte character with a most `n` bytes, `mbtowc_l` returns the length in bytes of the multi-byte character and stores that wide character to the object pointed to by `pwc` (if `pwc` is not null).

If the object that points to forms an incomplete, but possibly valid, multi-byte character, `mbtowc_l` returns 2. If the object that points to does not form a partial multi-byte character, `mbtowc_l` returns 1.

**See Also**

`mbtowc`, `mbtowc_l`
mbsrtowcs

Synopsis

```c
size_t mbsrtowcs(wchar_t *dst,
                 const char **src,
                 size_t len,
                 mbstate_t *ps);
```

Description

mbsrtowcs converts a sequence of multi-byte characters that begins in the conversion state described by the object pointed to by ps, from the array indirectly pointed to by src into a sequence of corresponding wide characters If dst is not a null pointer, the converted characters are stored into the array pointed to by dst. Conversion continues up to and including a terminating null character, which is also stored.

Conversion stops earlier in two cases: when a sequence of bytes is encountered that does not form a valid multi-byte character, or (if dst is not a null pointer) when len wide characters have been stored into the array pointed to by dst. Each conversion takes place as if by a call to the mbrtowc function.

If dst is not a null pointer, the pointer object pointed to by src is assigned either a null pointer (if conversion stopped due to reaching a terminating null character) or the address just past the last multi-byte character converted (if any). If conversion stopped due to reaching a terminating null character and if dst is not a null pointer, the resulting state described is the initial conversion state.

See Also

mbsrtowcs_l, mbrtowc
mbsrtowcs_l

Synopsis

```c
size_t mbsrtowcs_l(wchar_t *dst,
    const char **src,
    size_t len,
    mbstate_t *ps,
    locale_t loc);
```

Description

mbsrtowcs_l converts a sequence of multi-byte characters that begins in the conversion state described by the object pointed to by `ps`, from the array indirectly pointed to by `src` into a sequence of corresponding wide characters. If `dst` is not a null pointer, the converted characters are stored into the array pointed to by `dst`. Conversion continues up to and including a terminating null character, which is also stored.

Conversion stops earlier in two cases: when a sequence of bytes is encountered that does not form a valid multi-byte character, or (if `dst` is not a null pointer) when `len` wide characters have been stored into the array pointed to by `dst`. Each conversion takes place as if by a call to the `mbrtowc` function.

If `dst` is not a null pointer, the pointer object pointed to by `src` is assigned either a null pointer (if conversion stopped due to reaching a terminating null character) or the address just past the last multi-byte character converted (if any). If conversion stopped due to reaching a terminating null character and if `dst` is not a null pointer, the resulting state described is the initial conversion state.

See Also

mbsrtowcs_l, mbtowc
**msbinit**

**Synopsis**

```c
int msbinit(const mbstate_t *ps);
```

**Description**

`msbinit` function returns nonzero if `ps` is a null pointer or if the pointed-to object describes an initial conversion state; otherwise, `msbinit` returns zero.
**wchar_t**

**Synopsis**

```c
typedef __RAL_WCHAR_T wchar_t;
```

**Description**

`wchar_t` holds a single wide character.

Depending on implementation you can control whether `wchar_t` is represented by a short 16-bit type or the standard 32-bit type.
**wcrtom**

**Synopsis**

```c
size_t wcrtom(char *s,
       wchar_t wc,
       mbstate_t *ps);
```

If `s` is a null pointer, `wcrtom` function is equivalent to the call `wcrtom(buf, L'\0', ps)` where `buf` is an internal buffer.

If `s` is not a null pointer, `wcrtom` determines the number of bytes needed to represent the multibyte character that corresponds to the wide character given by `wc`, and stores the multibyte character representation in the array whose first element is pointed to by `s`. At most `MB_CUR_MAX` bytes are stored. If `wc` is a null wide character, a null byte is stored; the resulting state described is the initial conversion state.

`wcrtom` returns the number of bytes stored in the array object. When `wc` is not a valid wide character, an encoding error occurs: `wcrtom` stores the value of the macro `EILSEQ` in `errno` and returns `(size_t)(-1);` the conversion state is unspecified.
**wcrtomb_l**

**Synopsis**

```c
size_t wcrtomb_l(char *s,
                 wchar_t wc,
                 mbstate_t *ps,
                 locale_t loc);
```

If `s` is a null pointer, `wcrtomb_l` function is equivalent to the call `wcrtomb_l(buf, L'\0', ps, loc)` where `buf` is an internal buffer.

If `s` is not a null pointer, `wcrtomb_l` determines the number of bytes needed to represent the multibyte character that corresponds to the wide character given by `wc`, and stores the multibyte character representation in the array whose first element is pointed to by `s`. At most `MB_CUR_MAX` bytes are stored. If `wc` is a null wide character, a null byte is stored; the resulting state described is the initial conversion state.

`wcrtomb_l` returns the number of bytes stored in the array object. When `wc` is not a valid wide character, an encoding error occurs: `wcrtomb_l` stores the value of the macro `EILSEQ` in `errno` and returns `(size_t)(-1)`; the conversion state is unspecified.
**wcscat**

**Synopsis**

```c
wchar_t *wcscat(wchar_t *s1,
                const wchar_t *s2);
```

**Description**

`wcscat` appends a copy of the wide string pointed to by `s2` (including the terminating null wide character) to the end of the wide string pointed to by `s1`. The initial character of `s2` overwrites the null wide character at the end of `s1`. The behavior of `wcscat` is undefined if copying takes place between objects that overlap.

`wcscat` returns the value of `s1`. 
**wcschr**

**Synopsis**

```c
wchar_t *wcschr(const wchar_t *s, wchar_t c);
```

**Description**

`wcschr` locates the first occurrence of `c` in the wide string pointed to by `s`. The terminating wide null character is considered to be part of the string.

`wcschr` returns a pointer to the located wide character, or a null pointer if `c` does not occur in the string.
wcscmp

Synopsis

```c
int wcscmp(const wchar_t *s1,
            const wchar_t *s2);
```

Description

wcscmp compares the wide string pointed to by s1 to the wide string pointed to by s2. wcscmp returns an integer greater than, equal to, or less than zero if the wide string pointed to by s1 is greater than, equal to, or less than the wide string pointed to by s2.
**wcscpy**

**Synopsis**

```c
wchar_t *wcscpy(wchar_t *s1, const wchar_t *s2);
```

**Description**

*wcscpy* copies the wide string pointed to by *s2* (including the terminating null wide character) into the array pointed to by *s1*. The behavior of *wcscpy* is undefined if copying takes place between objects that overlap.

*wcscpy* returns the value of *s1*. 
wcscspn

Synopsis

```c
size_t wcscspn(const wchar_t *s1,
               const wchar_t *s2);
```

Description

`wcscspn` computes the length of the maximum initial segment of the wide string pointed to by `s1` which consists entirely of wide characters not from the wide string pointed to by `s2`.

`wcscspn` returns the length of the segment.
\textbf{wcsdup}

\textbf{Synopsis}

\begin{verbatim}
wchar_t *wcsdup(const wchar_t *s1);
\end{verbatim}

\textbf{Description}

\texttt{wcsdup} duplicates the wide string pointed to by \texttt{s1} by using \texttt{malloc} to allocate memory for a copy of \texttt{s} and then copying \texttt{s}, including the terminating wide null character, to that memory. The returned pointer can be passed to \texttt{free}. \texttt{wcsdup} returns a pointer to the new wide string or a null pointer if the new string cannot be created.

\textbf{Note}

\texttt{wcsdup} is an extension commonly found in Linux and BSD C libraries.
**wcslen**

**Synopsis**

```c
size_t wcslen(const wchar_t *s);
```

**Description**

`wcslen` returns the length of the wide string pointed to by `s`, that is the number of wide characters that precede the terminating null wide character.
wcsncat

Synopsis

```c
wchar_t *wcsncat(wchar_t *s1,
    const wchar_t *s2,
    size_t n);
```

Description

`wcsncat` appends not more than `n` wide characters from the array pointed to by `s2` to the end of the wide string pointed to by `s1`. A null wide character in `s1` and wide characters that follow it are not appended. The initial wide character of `s2` overwrites the null wide character at the end of `s1`. A terminating wide null character is always appended to the result. The behavior of `wcsncat` is undefined if copying takes place between objects that overlap.

`wcsncat` returns the value of `s1`. 
**wcsnchr**

**Synopsis**

```c
wchar_t *wcsnchr(const wchar_t *str, size_t n, wchar_t ch);
```

**Description**

`wcsnchr` searches not more than `n` wide characters to locate the first occurrence of `c` in the wide string pointed to by `s`. The terminating wide null character is considered to be part of the wide string.

`wcsnchr` returns a pointer to the located wide character, or a null pointer if `c` does not occur in the string.
wcsncmp

Synopsis

```c
int wcsncmp(const wchar_t *s1,
            const wchar_t *s2,
            size_t n);
```

Description

`wcsncmp` compares not more than `n` wide characters from the array pointed to by `s1` to the array pointed to by `s2`. Characters that follow a null wide character are not compared.

`wcsncmp` returns an integer greater than, equal to, or less than zero, if the possibly null-terminated array pointed to by `s1` is greater than, equal to, or less than the possibly null-terminated array pointed to by `s2`. 


**wcsncpy**

**Synopsis**

```c
wchar_t *wcsncpy(wchar_t *s1,
                 const wchar_t *s2,
                 size_t n);
```

**Description**

`wcsncpy` copies not more than `n` wide characters from the array pointed to by `s2` to the array pointed to by `s1`. Wide characters that follow a null wide character in `s2` are not copied. The behavior of `wcsncpy` is undefined if copying takes place between objects that overlap. If the array pointed to by `s2` is a wide string that is shorter than `n` wide characters, null wide characters are appended to the copy in the array pointed to by `s1`, until `n` characters in all have been written.

`wcsncpy` returns the value of `s1`. 
**wcsnlen**

**Synopsis**

```c
size_t wcsnlen(const wchar_t *s, size_t n);
```

**Description**

This returns the length of the wide string pointed to by `s`, up to a maximum of `n` wide characters. **wcsnlen** only examines the first `n` wide characters of the string `s`.

**Note**

**wcsnlen** is an extension commonly found in Linux and BSD C libraries.
**wcsnstr**

**Synopsis**

```c
wchar_t *wcsnstr(const wchar_t *s1,
                  const wchar_t *s2,
                  size_t n);
```

**Description**

*wcsnstr* searches at most *n* wide characters to locate the first occurrence in the wide string pointed to by *s1* of the sequence of wide characters (excluding the terminating null wide character) in the wide string pointed to by *s2*.

*wcsnstr* returns a pointer to the located string, or a null pointer if the string is not found. If *s2* points to a string with zero length, *wcsnstr* returns *s1*.

**Note**

*wcsnstr* is an extension commonly found in Linux and BSD C libraries.
**wcspbrk**

**Synopsis**

```c
wchar_t *wcspbrk(const wchar_t *s1,
                  const wchar_t *s2);
```

**Description**

The function **wcspbrk** locates the first occurrence in the wide string pointed to by **s1** of any wide character from the wide string pointed to by **s2**.

**wcspbrk** returns a pointer to the wide character, or a null pointer if no wide character from **s2** occurs in **s1**.
wcsrchr

Synopsis

```c
wchar_t *wcsrchr(const wchar_t *s, wchar_t c);
```

Description

`wcsrchr` locates the last occurrence of `c` in the wide string pointed to by `s`. The terminating wide null character is considered to be part of the string.

`wcsrchr` returns a pointer to the wide character, or a null pointer if `c` does not occur in the wide string.
**wcspn**

**Synopsis**

```c
size_t wcspn(const wchar_t *s1,
        const wchar_t *s2);
```

**Description**

*wcspn* computes the length of the maximum initial segment of the wide string pointed to by *s1* which consists entirely of wide characters from the wide string pointed to by *s2*.

*wcspn* returns the length of the segment.
**wcsstr**

**Synopsis**

```c
wchar_t *wcsstr(const wchar_t *s1,
               const wchar_t *s2);
```

**Description**

`wcsstr` locates the first occurrence in the wide string pointed to by `s1` of the sequence of wide characters (excluding the terminating null wide character) in the wide string pointed to by `s2`.

`wcsstr` returns a pointer to the located wide string, or a null pointer if the wide string is not found. If `s2` points to a wide string with zero length, `wcsstr` returns `s1`. 
**wcstok**

**Synopsis**

```c
wchar_t *wcstok(wchar_t *s1, const wchar_t *s2);
```

**Description**

`wcstok` A sequence of calls to `wcstok` breaks the wide string pointed to by `s1` into a sequence of tokens, each of which is delimited by a wide character from the wide string pointed to by `s2`. The first call in the sequence has a non-null first argument; subsequent calls in the sequence have a null first argument. The separator wide string pointed to by `s2` may be different from call to call.

The first call in the sequence searches the wide string pointed to by `s1` for the first wide character that is not contained in the current separator wide string pointed to by `s2`. If no such wide character is found, then there are no tokens in the wide string pointed to by `s1` and `wcstok` returns a null pointer. If such a wide character is found, it is the start of the first token.

`wcstok` then searches from there for a wide character that is contained in the current wide separator string. If no such wide character is found, the current token extends to the end of the wide string pointed to by `s1`, and subsequent searches for a token will return a null pointer. If such a wide character is found, it is overwritten by a wide null character, which terminates the current token. `wcstok` saves a pointer to the following wide character, from which the next search for a token will start.

Each subsequent call, with a null pointer as the value of the first argument, starts searching from the saved pointer and behaves as described above.

**Note**

`wcstok` maintains static state and is therefore not reentrant and not thread safe. See `wcstok_r` for a thread-safe and reentrant variant.
wcstok_r

Synopsis

```c
wchar_t *wcstok_r(wchar_t *s1, const wchar_t *s2, wchar_t **s3);
```

Description

`wcstok_r` is a reentrant version of the function `wcstok` where the state is maintained in the object of type `wchar_t` pointed to by `s3`.

Note

`wcstok_r` is an extension commonly found in Linux and BSD C libraries.

See Also

`wcstok`.
wctob

Synopsis

```c
int wctob(wint_t c);
```

Description

wctob determines whether `c` corresponds to a member of the extended character set whose multi-byte character representation is a single byte when in the initial shift state in the current locale.

Description

this returns EOF if `c` does not correspond to a multi-byte character with length one in the initial shift state. Otherwise, it returns the single-byte representation of that character as an unsigned char converted to an int.
wctob_l

Synopsis

```
int wctob_l(wint_t c,
locale_t loc);
```

Description

`wctob_l` determines whether `c` corresponds to a member of the extended character set whose multi-byte character representation is a single byte when in the initial shift state in locale `loc`.

Description

`wctob_l` returns `EOF` if `c` does not correspond to a multi-byte character with length one in the initial shift state. Otherwise, it returns the single-byte representation of that character as an `unsigned char` converted to an `int`. 
wint_t

Synopsis

typedef long wint_t;

Description

wint_t is an integer type that is unchanged by default argument promotions that can hold any value corresponding to members of the extended character set, as well as at least one value that does not correspond to any member of the extended character set (WEOF).
wmemccpy

Synopsis

```c
wchar_t *wmemccpy(wchar_t *s1,
const wchar_t *s2,
wchar_t c,
size_t n);
```

Description

*wmemccpy* copies at most *n* wide characters from the object pointed to by *s2* into the object pointed to by *s1*. The copying stops as soon as *n* wide characters are copied or the wide character *c* is copied into the destination object pointed to by *s1*. The behavior of *wmemccpy* is undefined if copying takes place between objects that overlap.

*wmemccpy* returns a pointer to the wide character immediately following *c* in *s1*, or **NULL** if *c* was not found in the first *n* wide characters of *s2*.

Note

*wmemccpy* conforms to POSIX.1-2008.
wmemchr

Synopsis

```c
wchar_t *wmemchr(const wchar_t *s, wchar_t c, size_t n);
```

Description

`wmemchr` locates the first occurrence of `c` in the initial `n` characters of the object pointed to by `s`. Unlike `wcschr`, `wmemchr` does not terminate a search when a null wide character is found in the object pointed to by `s`.

`wmemchr` returns a pointer to the located wide character, or a null pointer if `c` does not occur in the object.
wmemcmp

Synopsis

```c
int wmemcmp(const wchar_t *s1,
             const wchar_t *s2,
             size_t n);
```

Description

`wmemcmp` compares the first `n` wide characters of the object pointed to by `s1` to the first `n` wide characters of the object pointed to by `s2`. `wmemcmp` returns an integer greater than, equal to, or less than zero as the object pointed to by `s1` is greater than, equal to, or less than the object pointed to by `s2`. 
wmemcpy

Synopsis

```c
wchar_t *wmemcpy(wchar_t *s1,
                 const wchar_t *s2,
                 size_t n);
```

Description

`wmemcpy` copies `n` wide characters from the object pointed to by `s2` into the object pointed to by `s1`. The behavior of `wmemcpy` is undefined if copying takes place between objects that overlap.

`wmemcpy` returns the value of `s1`.


**wmemmove**

**Synopsis**

```c
wchar_t *wmemmove(wchar_t *s1,
                   const wchar_t *s2,
                   size_t n);
```

**Description**

`wmemmove` copies *n* wide characters from the object pointed to by *s2* into the object pointed to by *s1* ensuring that if *s1* and *s2* overlap, the copy works correctly. Copying takes place as if the *n* wide characters from the object pointed to by *s2* are first copied into a temporary array of *n* wide characters that does not overlap the objects pointed to by *s1* and *s2*, and then the *n* wide characters from the temporary array are copied into the object pointed to by *s1*.

`wmemmove` returns the value of *s1*. 
**wmempcpy**

**Synopsis**

```c
wchar_t *wmempcpy(wchar_t *s1, const wchar_t *s2, size_t n);
```

**Description**

*wmempcpy* copies *n* wide characters from the object pointed to by *s2* into the object pointed to by *s1*. The behavior of *wmempcpy* is undefined if copying takes place between objects that overlap.

*wmempcpy* returns a pointer to the wide character following the last written wide character.

**Note**

This is an extension found in GNU libc.
**wmemset**

**Synopsis**

```c
wchar_t *wmemset(wchar_t *s, wchar_t c, size_t n);
```

**Description**

`wmemset` copies the value of `c` into each of the first `n` wide characters of the object pointed to by `s`.

`wmemset` returns the value of `s`. 
**wstrsep**

**Synopsis**

```c
wchar_t *wstrsep(wchar_t **stringp, const wchar_t *delim);
```

**Description**

`wstrsep` locates, in the wide string referenced by `*stringp`, the first occurrence of any wide character in the wide string `delim` (or the terminating wide null character) and replaces it with a wide null character. The location of the next character after the delimiter wide character (or NULL, if the end of the string was reached) is stored in `*stringp`. The original value of `*stringp` is returned.

An empty field (that is, a wide character in the string `delim` occurs as the first wide character of `*stringp` can be detected by comparing the location referenced by the returned pointer to a wide null character.

If `*stringp` is initially null, `wstrsep` returns null.

**Note**

`wstrsep` is not an ISO C function, but appears in BSD4.4 and Linux.
API Summary

**Classification functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iswalnum</td>
<td>Is character alphanumeric?</td>
</tr>
<tr>
<td>iswalpha</td>
<td>Is character alphabetic?</td>
</tr>
<tr>
<td>iswblank</td>
<td>Is character blank?</td>
</tr>
<tr>
<td>iswcntrl</td>
<td>Is character a control?</td>
</tr>
<tr>
<td>iswctype</td>
<td>Determine character type</td>
</tr>
<tr>
<td>iswdigit</td>
<td>Is character a decimal digit?</td>
</tr>
<tr>
<td>iswgraph</td>
<td>Is character a control?</td>
</tr>
<tr>
<td>iswlower</td>
<td>Is character a lowercase letter?</td>
</tr>
<tr>
<td>iswprint</td>
<td>Is character printable?</td>
</tr>
<tr>
<td>iswpunct</td>
<td>Is character punctuation?</td>
</tr>
<tr>
<td>iswspace</td>
<td>Is character a whitespace character?</td>
</tr>
<tr>
<td>iswupper</td>
<td>Is character an uppercase letter?</td>
</tr>
<tr>
<td>iswxdigit</td>
<td>Is character a hexadecimal digit?</td>
</tr>
<tr>
<td>wctype</td>
<td>Construct character class</td>
</tr>
</tbody>
</table>

**Conversion functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>towctrans</td>
<td>Translate character</td>
</tr>
<tr>
<td>tolower</td>
<td>Convert uppercase character to lowercase</td>
</tr>
<tr>
<td>towupper</td>
<td>Convert lowercase character to uppercase</td>
</tr>
<tr>
<td>wctrans</td>
<td>Construct character mapping</td>
</tr>
</tbody>
</table>

**Classification functions (extended)**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iswalnum.l</td>
<td>Is character alphanumeric?</td>
</tr>
<tr>
<td>iswalpha.l</td>
<td>Is character alphabetic?</td>
</tr>
<tr>
<td>iswblank.l</td>
<td>Is character blank?</td>
</tr>
<tr>
<td>iswcntrl.l</td>
<td>Is character a control?</td>
</tr>
<tr>
<td>iswctype.l</td>
<td>Determine character type</td>
</tr>
<tr>
<td>iswdigit.l</td>
<td>Is character a decimal digit?</td>
</tr>
<tr>
<td>iswgraph.l</td>
<td>Is character a control?</td>
</tr>
<tr>
<td>iswlower.l</td>
<td>Is character a lowercase letter?</td>
</tr>
<tr>
<td>iswprint.l</td>
<td>Is character printable?</td>
</tr>
<tr>
<td>iswpunct.l</td>
<td>Is character punctuation?</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>iswspace_l</td>
<td>Is character a whitespace character?</td>
</tr>
<tr>
<td>iswupper_l</td>
<td>Is character an uppercase letter?</td>
</tr>
<tr>
<td>iswxdigit_l</td>
<td>Is character a hexadecimal digit?</td>
</tr>
<tr>
<td></td>
<td><strong>Conversion functions (extended)</strong></td>
</tr>
<tr>
<td>towctrans_l</td>
<td>Translate character</td>
</tr>
<tr>
<td>tolower_l</td>
<td>Convert uppercase character to lowercase</td>
</tr>
<tr>
<td>towupper_l</td>
<td>Convert lowercase character to uppercase</td>
</tr>
<tr>
<td>wcstrans_l</td>
<td>Construct character mapping</td>
</tr>
</tbody>
</table>
iswalnum

Synopsis

```c
int iswalnum(wint_t c);
```

Description

iswalnum tests for any wide character for which iswalpha or iswdigit is true.
iswalnum_l

Synopsis

```c
int iswalnum_l(wint_t c,
            locale_t loc);
```

Description

`iswalnum_l` tests for any wide character for which `iswalpha_l` or `iswdigit_l` is true in the locale `loc`. 
iswalpha

Synopsis

```c
int iswalpha(wint_t c);
```

Description

`iswalpha` returns true if the wide character `c` is alphabetic. Any character for which `iswupper` or `iswlower` returns true is considered alphabetic in addition to any of the locale-specific set of alphabetic characters for which none of `iswcntrl`, `iswdigit`, `iswpunct`, or `iswspace` is true.

In the C locale, `iswalpha` returns nonzero (true) if and only if `iswupper` or `iswlower` return true for the value of the argument `c`. 
iswalpha_l

Synopsis

```c
int iswalpha_l(wint_t c, locale_t loc);
```

Description

`iswalpha_l` returns true if the wide character `c` is alphabetic in the locale `loc`. Any character for which `iswupper_l` or `iswlower_l` returns true is considered alphabetic in addition to any of the locale-specific set of alphabetic characters for which none of `iswalpha_l`, `iswdigit_l`, `iswpunct_l`, or `iswspace_l` is true.
iswblank

Synopsis

```c
int iswblank(wint_t c);
```

Description

iswblank tests for any wide character that is a standard blank wide character or is one of a locale-specific set of wide characters for which iswspace is true and that is used to separate words within a line of text. The standard blank wide are space and horizontal tab.

In the C locale, iswblank returns true only for the standard blank characters.
iswblank_l

Synopsis

```c
int iswblank_l(wint_t c,
    locale_t loc);
```

Description

`iswblank_l` tests for any wide character that is a standard blank wide character in the locale `loc` or is one of a locale-specific set of wide characters for which `iswspace_l` is true and that is used to separate words within a line of text. The standard blank wide are space and horizontal tab.
iswcntrl

Synopsis

```c
int iswcntrl(wint_t c);
```

Description

iswcntrl tests for any wide character that is a control character.
iswcntrl_l

Synopsis

```c
int iswcntrl_l(wint_t c, locale_t loc);
```

Description

iswcntrl_l tests for any wide character that is a control character in the locale loc.
iswctype

Synopsis

```c
int iswctype(wint_t c, wctype_t t);
```

Description

iswctype determines whether the wide character c has the property described by t in the current locale.
iswctype_l

Synopsis

```c
int iswctype_l(wint_t c,
               wctype_t t,
               locale_t loc);
```

Description

`iswctype_l` determines whether the wide character `c` has the property described by `t` in the locale `loc`. 
iswdigit

Synopsis

```c
int iswdigit(wint_t c);
```

Description

`iswdigit` tests for any wide character that corresponds to a decimal-digit character.
iswdigit_l

Synopsis

```c
int iswdigit_l(wint_t c,
               locale_t loc);
```

Description

iswdigit_l tests for any wide character that corresponds to a decimal-digit character in the locale `loc`. 
iswgraph

Synopsis

```c
int iswgraph(wint_t c);
```

Description

iswgraph tests for any wide character for which iswprint is true and iswspace is false.
iswgraph_l

Synopsis

```c
int iswgraph_l(wint_t c,
    locale_t loc);
```

Description

iswgraph_l tests for any wide character for which iswprint is true and iswspace is false in the locale loc.
iswlower

Synopsis

```c
int iswlower(wint_t c);
```

Description

iswlower tests for any wide character that corresponds to a lowercase letter or is one of a locale-specific set of wide characters for which none of iswctrl, iswdigit, iswpunct, or iswspace is true.
iswlower_l

Synopsis

```c
int iswlower_l(wint_t c,
               locale_t loc);
```

Description

iswlower_l tests for any wide character that corresponds to a lowercase letter in the locale loc or is one of a locale-specific set of wide characters for which none of iswcntrl_l, iswdigit_l, iswpunct_l, or iswspace_l is true.
**iswprint**

**Synopsis**

```c
int iswprint(wint_t c);
```

**Description**

`iswprint` returns nonzero (true) if and only if the value of the argument `c` is any printing character.
**iswprint_l**

**Synopsis**

```c
int iswprint_l(wint_t c,
               locale_t loc);
```

**Description**

`iswprint_l` returns nonzero (true) if and only if the value of the argument `c` is any printing character in the locale `loc`. 
iswpunct

Synopsis

```c
int iswpunct(wint_t c);
```

Description

iswpunct tests for any printing wide character that is one of a locale-specific set of punctuation wide characters for which neither iswspace nor iswalnum is true.
iswpunct_l

Synopsis

```c
int iswpunct_l(wint_t c,
        locale_t loc);
```

Description

iswpunct_l tests for any printing wide character that is one of a locale-specific set of punctuation wide characters in locale loc for which neither iswspace_l nor iswalnum_l is true.
iswspace

Synopsis

```c
int iswspace(wint_t c);
```

Description

`iswspace` tests for any wide character that corresponds to a locale-specific set of white-space wide characters for which none of `iswalnum`, `iswgraph`, or `iswpunct` is true.
**iswspace_l**

**Synopsis**

```c
int iswspace_l(wint_t c,
               locale_t loc);
```

**Description**

`iswspace_l` tests for any wide character that corresponds to a locale-specific set of white-space wide characters in the locale `loc` for which none of `iswalnum`, `iswgraph_l`, or `iswpunct_l` is true.
iswupper

Synopsis

```c
int iswupper(wint_t c);
```

Description

`iswupper` tests for any wide character that corresponds to an uppercase letter or is one of a locale-specific set of wide characters for which none of `iswcntrl`, `iswdigit`, `iswpunct`, or `iswspace` is true.
iswupper_l

Synopsis

```c
int iswupper_l(wint_t c,
               locale_t loc);
```

Description

iswupper_l tests for any wide character that corresponds to an uppercase letter or is one of a locale-specific set of wide characters in the locale loc for which none of iswcntrl_l, iswdigit_l, iswpunct_l, or iswspace_l is true.
iswxdigit

Synopsis

```c
int iswxdigit(wint_t c);
```

Description

iswxdigit tests for any wide character that corresponds to a hexadecimal digit.
iswxdigit_l

Synopsis

```c
int iswxdigit_l(wint_t c, locale_t loc);
```

Description

iswxdigit_l tests for any wide character that corresponds to a hexadecimal digit in the locale loc.
towctrans

Synopsis

```c
wint_t towctrans(wint_t c,
                 wctrans_t t);
```

Description

towctrans maps the wide character c using the mapping described by t in the current locale.
towctrans_l

Synopsis

```c
wint_t towctrans_l(wint_t c,
                   wctrans_t t,
                   locale_t loc);
```

Description

towctrans_l maps the wide character c using the mapping described by t in the current locale.
towlower

Synopsis

```
    wint_t towlower(wint_t c);
```

Description

`towlower` converts an uppercase letter to a corresponding lowercase letter.

If the argument `c` is a wide character for which `iswupper` is true and there are one or more corresponding wide characters, in the current locale, for which `iswlower` is true, `towlower` returns one (and always the same one for any given locale) of the corresponding wide characters; otherwise, `c` is returned unchanged.
towlower_l

Synopsis

```c
wint_t towlower_l(wint_t c,
                  locale_t loc);
```

Description

towlower_l converts an uppercase letter to a corresponding lowercase letter in locale loc.

If the argument c is a wide character for which iswupper_l is true and there are one or more corresponding wide characters, in the locale loc, for which iswlower_l is true, towlower_l returns one (and always the same one for any given locale) of the corresponding wide characters; otherwise, c is returned unchanged.
towupper

Synopsis

```c
wint_t towupper(wint_t c);
```

Description

towupper converts a lowercase letter to a corresponding uppercase letter.

If the argument c is a wide character for which iswlower is true and there are one or more corresponding wide characters, in the current current locale, for which iswupper is true, towupper returns one (and always the same one for any given locale) of the corresponding wide characters; otherwise, c is returned unchanged.
towupper_l

Synopsis

```c
wint_t towupper_l(wint_t c,
locale_t loc);
```

Description

towupper_l converts a lowercase letter to a corresponding uppercase letter in locale loc.

If the argument c is a wide character for which iswlower_l is true and there are one or more corresponding wide characters, in the locale loc, for which iswupper_l is true, towupper_l returns one (and always the same one for any given locale) of the corresponding wide characters; otherwise, c is returned unchanged.
**wctrans**

**Synopsis**

```c
wctrans_t wctrans(const char *property);
```

**Description**

`wctrans` constructs a value of type `wctrans_t` that describes a mapping between wide characters identified by the string argument `property`.

If `property` identifies a valid mapping of wide characters in the current locale, `wctrans` returns a nonzero value that is valid as the second argument to `towctrans`; otherwise, it returns zero.

**Note**

The only mappings supported are "tolower" and "toupper".
wctrans_l

Synopsis

```c
wctrans_t wctrans_l(const char *property,
                     locale_t loc);
```

Description

wctrans_l constructs a value of type `wctrans_t` that describes a mapping between wide characters identified by the string argument `property` in locale `loc`.

If `property` identifies a valid mapping of wide characters in the locale `loc`, `wctrans_l` returns a nonzero value that is valid as the second argument to `towctrans_l`; otherwise, it returns zero.

Note

The only mappings supported are "tolower" and "toupper".
**wctype**

**Synopsis**

```c
wctype_t wctype(const char *property);
```

**Description**

`wctype` constructs a value of type `wctype_t` that describes a class of wide characters identified by the string argument `property`.

If `property` identifies a valid class of wide characters in the current locale, `wctype` returns a nonzero value that is valid as the second argument to `iswctype`; otherwise, it returns zero.

**Note**

The only mappings supported are "alnum", "alpha", "blank", "cntrl", "digit", "graph", "lower", "print", "punct", "space", "upper", and "xdigit".
<xlocale.h>

API Summary

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>duplocale</td>
<td>Duplicate current locale data</td>
</tr>
<tr>
<td>freelocale</td>
<td>Free a locale</td>
</tr>
<tr>
<td>localeconv_l</td>
<td>Get locale data</td>
</tr>
<tr>
<td>newlocale</td>
<td>Create a new locale</td>
</tr>
</tbody>
</table>
duplocale

Synopsis

locale_t duplocale(locale_t loc);

Description

duplocale duplicates the locale object referenced by loc.

If there is insufficient memory to duplicate loc, duplocale returns NULL and sets errno to ENOMEM as required by POSIX.1-2008.

Duplicated locales must be freed with freelocale.

This is different behavior from the GNU glibc implementation which makes no mention of setting errno on failure.

Note

This extension is derived from BSD, POSIX.1, and glibc.
freelocale

Synopsis

```c
int freelocale(locale_t loc);
```

Description

`freelocale` frees the storage associated with `loc`.

`freelocale` zero on success, 1 on error.
localeconv_l

Synopsis

    localeconv_l(locale_t loc);

Description

localeconv_l returns a pointer to a structure of type lconv with the corresponding values for the locale loc filled in.
newlocale

Synopsis

locale_t newlocale(int category_mask,
    const char *locale,
    locale_t base);

Description

newlocale creates a new locale object or modifies an existing one. If the base argument is NULL, a new locale object is created.

category_mask specifies the locale categories to be set or modified. Values for category_mask are constructed by a bitwise-inclusive OR of the symbolic constants LC_CTYPE_MASK, LC_NUMERIC_MASK, LC_TIME_MASK, LC_COLLATE_MASK, LC_MONETARY_MASK, and LC_MESSAGES_MASK.

For each category with the corresponding bit set in category_mask, the data from the locale named by locale is used. In the case of modifying an existing locale object, the data from the locale named by locale replaces the existing data within the locale object. If a completely new locale object is created, the data for all sections not requested by category_mask are taken from the default locale.

The locales C and POSIX are equivalent and defined for all settings of category_mask:

If locale is NULL, then the C locale is used. If locale is an empty string, newlocale will use the default locale.

If base is NULL, the current locale is used. If base is LC_GLOBAL_LOCALE, the global locale is used.

If mask is LC_ALL_MASK, base is ignored.

Note

POSIX.1-2008 does not specify whether the locale object pointed to by base is modified or whether it is freed and a new locale object created.

Implementation

The category mask LC_MESSAGES_MASK is not implemented as POSIX messages are not implemented.
C++ Library User Guide

SEGGER Embedded Studio for ARM provides a limited C++ library suitable for use in an embedded application.

Standard library

The following C++ standard header files are provided in $(StudioDir)/include:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;cassert&gt;</code></td>
<td>C++ wrapper on <code>assert.h</code>.</td>
</tr>
<tr>
<td><code>&lt;cctype&gt;</code></td>
<td>C++ wrapper on <code>ctype.h</code>.</td>
</tr>
<tr>
<td><code>&lt;cerrno&gt;</code></td>
<td>C++ wrapper on <code>errno.h</code>.</td>
</tr>
<tr>
<td><code>&lt;cfloat&gt;</code></td>
<td>C++ wrapper on <code>float.h</code>.</td>
</tr>
<tr>
<td><code>&lt;ciso646&gt;</code></td>
<td>C++ wrapper on <code>iso646.h</code>.</td>
</tr>
<tr>
<td><code>&lt;climits&gt;</code></td>
<td>C++ wrapper on <code>limits.h</code>.</td>
</tr>
<tr>
<td><code>&lt;locale&gt;</code></td>
<td>C++ wrapper on <code>locale.h</code>.</td>
</tr>
<tr>
<td><code>&lt;cmath&gt;</code></td>
<td>C++ wrapper on <code>math.h</code>.</td>
</tr>
<tr>
<td><code>&lt;csetjmp&gt;</code></td>
<td>C++ wrapper on <code>setjmp.h</code>.</td>
</tr>
<tr>
<td><code>&lt;cstdarg&gt;</code></td>
<td>C++ wrapper on <code>stdarg.h</code>.</td>
</tr>
<tr>
<td><code>&lt;cstddef&gt;</code></td>
<td>C++ wrapper on <code>stddef.h</code>.</td>
</tr>
<tr>
<td><code>&lt;cstdint&gt;</code></td>
<td>C++ wrapper on <code>cstdint.h</code>.</td>
</tr>
<tr>
<td><code>&lt;cstdio&gt;</code></td>
<td>C++ wrapper on <code>stdio.h</code>.</td>
</tr>
<tr>
<td><code>&lt;cstdlib&gt;</code></td>
<td>C++ wrapper on <code>stdlib.h</code>.</td>
</tr>
<tr>
<td>Header</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td><code>&lt;cstring&gt;</code></td>
<td>C++ wrapper on <code>string.h</code></td>
</tr>
<tr>
<td><code>&lt;ctime&gt;</code></td>
<td>C++ wrapper on <code>time.h</code></td>
</tr>
<tr>
<td><code>&lt;cwchar&gt;</code></td>
<td>C++ wrapper on <code>wchar.h</code></td>
</tr>
<tr>
<td><code>&lt;cwctype&gt;</code></td>
<td>C++ wrapper on <code>wchar.h</code></td>
</tr>
<tr>
<td><code>&lt;exception&gt;</code></td>
<td>Definitions for exceptions.</td>
</tr>
<tr>
<td><code>&lt;new&gt;</code></td>
<td>Types and definitions for placement new and delete.</td>
</tr>
<tr>
<td><code>&lt;typeinfo&gt;</code></td>
<td>Definitions for RTTI.</td>
</tr>
</tbody>
</table>
Standard template library

The C++ STL functionality of STLPort is provided as a separate library package use Tools > Package Manager to install this package.
Subset API reference

This section contains a subset reference to the SEGGER Embedded Studio for ARM C++ library.
<new> - memory allocation

The header file `<new>` defines functions for memory allocation.

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>set_new_handler</code></td>
<td>Establish a function which is called when memory allocation fails.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>operator delete</code></td>
<td>Heap storage deallocators operator.</td>
</tr>
<tr>
<td><code>operator new</code></td>
<td>Heap storage allocators operator.</td>
</tr>
</tbody>
</table>
operator delete

Synopsis

\begin{verbatim}
void operator delete(void *ptr) throw();

void operator delete[](void *ptr) throw();
\end{verbatim}

Description

operator delete deallocates space of an object.

operator delete will do nothing if ptr is null. If ptr is not null then it should have been returned from a call to operator new.

operator delete[] has the same behaviour as operator delete but is used for array deallocation.

Portability

Standard C++.
**operator new**

**Synopsis**

```cpp
template<typename T> T *operator new(size_t size);
```

**Description**

`operator new` allocates space for an object whose size is specified by `size` and whose value is indeterminate.

`operator new` returns a null pointer if the space for the object cannot be allocated from free memory; if space for the object can be allocated, `operator new` returns a pointer to the start of the allocated space.

`operator new[]` has the same behaviour as `operator new` but is used for array allocation.

**Portability**

The implementation is not standard. The standard C++ implementation should throw an exception if memory allocation fails.
set_new_handler

Synopsis

typedef void (*new_handler)();

new_handler set_new_handler(new_handler) throw();

Description

ded_new_handler establishes a new_handler function.

set_new_handler establishes a new_handler function that is called when operator new fails to allocate the requested memory. If the new_handler function returns then operator new will attempt to allocate the memory again. The new_handler function can throw an exception to implement standard C++ behaviour for memory allocation failure.

Portability

Standard C++.
Utilities Reference
Compiler driver

This section describes the switches accepted by the compiler driver, cc. The compiler driver is capable of controlling compilation by all supported language compilers and the final link by the linker. It can also construct libraries automatically.

In contrast to many compilation and assembly language development systems, with SEGGER Embedded Studio for ARM you don’t invoke the assembler or compiler directly. Instead you’ll normally use the compiler driver cc as it provides an easy way to get files compiled, assembled, and linked. This section will introduce you to using the compiler driver to convert your source files to object files, executables, or other formats.

We recommend that you use the compiler driver rather than use the assembler or compiler directly because there the driver can assemble multiple files using one command line and can invoke the linker for you too. There is no reason why you should not invoke the assembler or compiler directly yourself, but you’ll find that typing in all the required options is quite tedious-and why do that when cc will provide them for you automatically?
File naming conventions

The compiler driver uses file extensions to distinguish the language the source file is written in. The compiler driver recognizes the extension .c as C source files, .cpp, .cc or .cxx as C++ source files, .s and .asm as assembly code files.

The compiler driver recognizes the extension .o as object files, .a as library files, .ld as linker script files and .xml as special-purpose XML files.

We strongly recommend that you adopt these extensions for your source files and object files because you’ll find that using the tools is much easier if you do.

C language files

When the compiler driver finds a file with a .c extension, it runs the C compiler to convert it to object code.

C++ language files

When the compiler driver finds a file with a .cpp extension, it runs the C++ compiler to convert it to object code.

Assembly language files

When the compiler driver finds a file with a .s or .asm extension, it runs the C preprocessor and then the assembler to convert it to object code.

Object code files

When the compiler driver finds a file with a .o or .a extension, it passes it to the linker to include it in the final application.
Command-line options

This section describes the command-line options accepted by the SEGGER Embedded Studio for ARM compiler driver.
-allow-multiple-definition (Allow multiple symbol definition)

Syntax

-allow-multiple-definition

Description

Do not generate an error when linking multiple symbols of the same name.
-ansi (Warn about potential ANSI problems)

Syntax

-ansi

Description

Warn about potential problems that conflict with the relevant ANSI or ISO standard for the files that are compiled.
-ar (Archive output)

Syntax

-ar

Description

This switch instructs the compiler driver to archive all output files into a library. Using -ar implies -c.

Example

The following command compiles file1.c, file2.asm, and file3.c to object code and archives them into the library file libfunc.a together with the object file file4.o.

```
cc -ar file1.c file2.asm file3.c file4.o -o libfunc.a
```
-arch (Set ARM architecture)

Syntax
-arch=a
-arch=list

Description
Specifies the ARM architecture to generate code for and the library variants to link with.

Example
To force compilation for V7A architecture you would use:
```
c -arch=v7A
```

To list supported architectures:
```
c -arch=list
```
-be (Big Endian)

Syntax
- be
- be8

Description
Generate code for a big endian (word or byte) target. Default generates little endian code.
-builtins (Use Builtins)

Syntax

-builtins

Description

Use builtin compiler functions, for example memcpy. Default does not use builtin compiler functions.
-c (Compile to object code, do not link)

Syntax
-c

Description
All named files are compiled to object code modules, but are not linked. You can use the -o option to name the output if you just supply one input filename.

Example
The following command compiles file1.c and file4.c to produce the object files file1.o and file4.o.

```
cc -c file1.c file4.c
```

The following command compiles file1.c and produces the object file obj/file1.o.

```
cc -c file1.c -o obj/file1.o
```
-clang (Use clang compiler/assembler)

Syntax
-clang

Description
Use the clang compiler and assembler. Default is to use the GNU compiler and assembler.
-cmselib (Create CMSE import library)

Syntax

-cmselib=/

Description

Create an import library (an object file) containing the symbols that represent the addresses of secure gateways to the entry functions of the linked executable.
-codec (Set file codec)

Syntax

-codec=c
-codec=list

Description

Set the file codec to use for the source file.

Example

To set the codec for a utf-8 encoded file:

```
cc -codec=utf-8
```

To list supported codecs:

```
cc -codec=list
```
-common (Allocate globals in common)

Syntax
-common

Description
Allocate declarations of zero initialized variables. This enables variables that have been declared (perhaps multiple times) but not defined to be allocated. The default requires a single definition of each zero initialized variable.
-cpu (Set ARM cpu core)

**Syntax**
- -cpu=c
- -cpu=list

**Description**
Specifies the cpu core to generate code for and the libraries to link against.

**Example**
To force compilation for Cortex-M3 core you would use:

```
c -cpu=Cortex-M3
```

To list supported cores:

```
c -cpu=list
```
-d (Define linker symbol)

Syntax

-d name=value

Description

You can define linker symbols using the -d option. The symbol definitions are passed to linker.

Example

The following defines the symbol, STACK_SIZE with a value of 512.

-d STACK_SIZE=512
-debugio (Define debugio implementation)

Syntax

-debugio=bkpt
-debugio=dcc
-debugio=mempoll

Description

Specifies the debugio implementation to link with. The default for architectures that have the ARM instruction set will use dcc and architectures that have only the Thumb-2 instruction set will use mempoll.

Example

The following selects the breakpoint debugio implementation for a cortex-m3 cpu

```
cc -cpu=Cortex-M3 -debugio=bkpt
```
-depend (Generate dependency file)

Syntax

- depend file

Description

Create a dependency file in file (suitable for inclusion into a makefile) when compiling a source file.

```
cc -c main.c -depend main.d
```
-D (Define macro symbol)

Syntax

-D name
-D name=value

Description

You can define preprocessor macros using the -D option. The macro definitions are passed on to the respective language compiler which is responsible for interpreting the definitions and providing them to the programmer within the language.

The first form above defines the macro name but without an associated replacement value, and the second defines the same macro with the replacement value value.

Example

The following defines two macros, SUPPORT_FLOAT with a value of 1 and LITTLE_ENDIAN with no replacement value.

-D SUPPORT_FLOAT=1 -DLITTLE_ENDIAN
-emit-relocs (Emit relocations)

Syntax

-emit-reloc

Description

Keep relocations in the executable file
-e (Set entry point symbol)

Syntax

-ename

Description

Linker option to set the entry point symbol to be name. The debugger will start execution from this symbol.
-exceptions (Enable C++ Exception Support)

Syntax
-exceptions

Description
Enables C++ exceptions to be compiled.
-E (Preprocess)

Syntax

-E

Description

This option preprocesses the supplied file and outputs the result to the standard output.

Example

The following preprocesses the file file.c supplying the macros, SUPPORT_FLOAT with a value of 1 and LITTLE_ENDIAN.

```
-E -DSUPPORT_FLOAT=1 -DLITTLE_ENDIAN file.c
```
-fill (Fill gaps)

Syntax

-\texttt{fill} = b

Description

Specify the byte value \( b \) to fill gaps in the output file produced by the linker.
-fabi (Floating Point Code Generation)

Syntax

-fabi=softfp
-fabi=hard

Description

Specifies the type of floating point code generation. The default is to use the software floating point implementation. If you use softfp then FPU instructions are generated, floating point arguments to functions are supplied in CPU registers. If you use hard then FPU instructions are generated, floating point arguments to functions are supplied in FPU registers.
-fpu (Set ARM FPU)

Syntax

-fpu=a
-fpu=list

Description

Specifies the floating point unit to generate code for when the fpabi option has been supplied.

Example

The following selects the fpu for a Cortex-M4

```
cc -cpu=Cortex-M4 -fpu=FPv4-SP-D16
```

To list supported cores:

```
cc -fpu-list
```
-framepointer (Enable generation of framepointer)

Syntax

-framepointer

Description

The -framepointer option instructs the compiler to store the stack frame pointer in a register.
-F (Set output format)

Syntax

-fmt

Description

The -F option instructs the compiler driver to generate an additional output file in the format fmt. The compiler driver supports the following formats:

- Fbin Create a .bin file
- Fhex Create a .hex file
- Fsrec Create a .srec file

The compiler driver will always output a .elf file as specified with the -o option. The name of the additional output file is the same as the .elf file with the file extension changed.

For example

```
cc file.c -o file.elf -Fbin
```

will generate the files file.elf and file.bin.
-g (Generate debugging information)

Syntax

- g
- g1
- g2
- g3

Description

The -g option instructs the compiler and assembler to generate source level debugging information.

The -g1 option instructs the compiler to generate backtrace and line number debugging information.

The -g2 option instructs the compiler to generate backtrace, line number and variable display debugging information.

The -g3 option instructs the compiler to generate backtrace, line number, variable display and macro display debugging information.

The default is to not generate any debugging information.
-hascmse (Generate cmse instructions)

Syntax
-hascmse

Description
The -hascmse option allows the compiler to generate code for the secure state of the v8m architecture.
-hascrc (Generate crc instructions)

Syntax

-hascrc

Description

The -hascrc option allows the compiler to generate crc instructions for v8a architecture.
-hascrypto (Generate crypto instructions)

Syntax

-hascrypto

Description

The -hascrypto option allows the compiler to generate crypto instructions for v8a architecture.
-hasdsp (Generate dsp instructions)

Syntax

-hasdsp

Description

The -hasdsp option allows the compiler to generate dsp instructions for v8m architecture.
-hasidiv (Generate integer divide instructions)

Syntax

-hasidiv

Description

The -hasidiv option instructs the compiler to generate integer divide instructions for v7a and v7r architectures.
-hassmalldivider (Do not generate multiply instructions)

Syntax

-hassmalldivider

Description

The -hassmalldivider option instructs the compiler to avoid generating multiply instructions for v6m architectures depending on code to be generated and optimization level requested.
-help (Display help information)

Syntax

-help

Description

Displays a short summary of the options accepted by the compiler driver.
-**instrument** (Instrument functions)

**Syntax**

- **instrument**

**Description**

This option causes the compiler to insert instrumentation calls on function entry and exit.
-I (Define user include directories)

Syntax

-Idirectory

Description

In order to find include files the compiler driver arranges for the compilers to search a number of standard directories. You can add directories to the search path using the -I switch which is passed on to each of the language processors.

You can specify more than one include directory by separating each directory component with either a comma or semicolon.
-I- (Exclude standard include directories)

Syntax
-\r

Description
Usually the compiler and assembler search for include files in the standard include directory created when the product is installed. If for some reason you wish to exclude these system locations from being searched when compiling a file, the -I- option will do this for you.
-J (Define system include directories)

Syntax

-J directory

Description

The -J option adds directory to the end of the list of directories to search for source files included (using triangular brackets) by the #include preprocessor command.

You can specify more than one include directory by separating each directory component with either a comma or semicolon in the property.
-kasm (Keep assembly code)

Syntax
-kasm

Description
The -kasm option instructs the compiler driver to keep intermediate assembly code files.
-kldscript (Keep linker script)

Syntax
-kldscript

Description
The -kldscript option instructs the compiler driver to keep generated linker script files.
-kpp (Keep preprocessor output)

Syntax

-kpp

Description

The -kpp option instructs the compiler driver to generate and keep intermediate preprocessor files.
-K (Keep linker symbol)

Syntax

-K name

Description

The linker removes unused code and data from the output file. This process is called deadstripping. To prevent the linker from deadstripping unreferenced code and data you wish to keep, you must use the -K command line option to force inclusion of symbols.

Example

If you have a C function, contextSwitch that must be kept in the output file (and which the linker will normally remove), you can force its inclusion using:

-KcontextSwitch
-l- (Do not link standard libraries)

Syntax

-l-

Description

The -l option instructs the compiler driver not to link standard libraries. If you use this option you must supply your own library functions or libraries.
-longcalls (Generate long calling sequences)

Syntax

-longcalls

Description

The -longcalls option causes the compiler to generate long call code sequences.
-lto (Enable link time optimization)

Syntax

-lto

Description

The -lto option causes the compiler to generate intermediate code which is optimized before the link.
-L (Set library directory path)

Syntax

-L dir

Description

Sets the library directory to dir. If -L is not specified on the command line, the default location to search for libraries is set to $(InstallDir)/lib.
-memorymap (Memory map file)

Syntax

-memorymap file

Description

The -memorymap option supplies the memory map file which is used to define the memory segments referenced in the section placement file. See Memory Map file format for a description of the memory map file format.

Example

-memorymap MemoryMap.xml
-memorymapmacros (Memory map macros)

Syntax

-memorymapmacros macros

Description

The -memorymapmacros option supplies macro definitions that are applied to the memory map file.

Example

The macros FLASH_START and FLASH_SIZE are defined for the memory map using:

-memorymapmacros "FLASH_START=0x08000000;FLASH_SIZE=0x10000"
-M (Display linkage map)

Syntax

-M

Description

The -M option prints a linkage map named the same as the linker output file with the .map file extension.
-n (Dry run, no execution)

Syntax

-n

Description

When -n is specified, the compiler driver processes options as usual, but does not execute any subprocesses to compile, assemble, archive or link applications.
-nointerwork (No interwork code for v4t)

Syntax

-nointerwork

Description

The -nointerwork option disables generation of bx lr instructions for v4t architectures.
-nowarn-mismatch (No warning on architecture mismatch)

Syntax

-nowarn-mismatch

Description

When -nowarn-mismatch is specified, the linker will ignore architecture mismatches on object file and libraries.
-nowarn-enumsize (No warning on enum size mismatch)

Syntax

-nowarn-enumsize

Description

When -nowarn-enumsize is specified, the linker will ignore enum size mismatches on object files and libraries.
-nowarn-wcharsize (No warning on wide character size mismatch)

Syntax

-nowarn-wcharsize

Description

When -nowarn-wcharsize is specified, the linker will ignore wide character size mismatches on object files and libraries.
-nostderr (No stderr output)

Syntax

-nostderr

Description

When -nostderr is specified, any stderr output of subprocesses is redirected to stdout.
-O (Optimize output)

Syntax

-Ox

Description

Pass the optimization option \(-Ox\) to the compiler and select library variant. The following options are supported:

- \(-O0\) No optimization, use libraries built with \(-O1\).
- \(-O1\) Level 1 optimization, use libraries built with \(-O1\).
- \(-O2\) Level 2 optimization, use libraries built with \(-O1\).
- \(-O3\) Level 3 optimization, use libraries built with \(-O1\).
- \(-Oz\) Optimize for more size, use libraries built with \(-Os\).
- \(-Os\) Optimize for size, use libraries built with \(-Os\).
-o (Set output file name)

Syntax

-o filename

Description

The -o option instructs the compiler driver to write linker or archiver output to filename.
-patch (Run patch command)

Syntax

-patch cmd

Description

The -patch option instructs the compiler driver to run the cmd after the link but before the creation of the additional output file. The macro $\text{(TargetPath)}$ is expanded to the full path of the linked executable.

Example

This example will run the command mypatch replacing $\text{(TargetPath)}$ with myoutput.elf

```
-patch "mypatch $(TargetPath)" -o myoutput.elf -Fbin
```

The mypatch command can modify myoutput.elf before the creation of the myoutput.bin.
-placement (Section placement file)

Syntax

-placement file

Description

The -placement option supplies the section placement file which is used to control the placement of program sections in the memory map segments. See Section Placement file format for a description of the section placement file format.

Example

```bash
-memorymap MemoryMap.xml -placement flash.xml
```
-placementmacros (Section placement macros)

Syntax

-placementmacros macros

Description

The -placementmacros option supplies macro definitions that are applied to the section placement file.

Example

The macros FLASH_START and FLASH_SIZE are defined for the section placement using:

```
-placementmacros "FLASH_START=0x08000000;FLASH_SIZE=0x10000"
```
-placementsegments (Section placement segments)

Syntax

-placementsegments segments

Description

The -placementsegments option supplies segments descriptions to the section placement file. You can use this rather than supplying a memory map file.

Example

A simple memory map with FLASH and SRAM can be supplied as follows:

```
-placementsegments "FLASH RX 0x0 0x10000;SRAM RWX 0x20000000 0x10000" -placement flash.xml
```
-printf (Select printf capability)

Syntax

-printf=c

Description

The -printf option selects the printf capability for the linked executable. The options are:

-printf=i[p][w] integer is supported, optional width and precision and optional wchar
-printf=l[p][w] long integer is supported, optional width and precision and optional wchar
-printf=ll[p][w] long long integer is supported, optional width and precision and optional wchar
-printf=f[ll][w] float, width and precision supported, optional long long and optional wchar
-printf=d[ll][w] double, width and precision supported, optional long long and optional wchar

Example

The minimal sized printf

-printf=i

The maximal functionality printf

-printf=dllw
-rtti (Enable C++ RTTI Support)

Syntax

-rtti

Description

Enables C++ run-time type information to be compiled.
-R (Set section name)

Syntax

-Rx name

Description

These options name the default name of the sections generated by the compiler/assembler to be name. The options are:

-Rc name change the default name of the code section
-Rd name change the default name of the data section
-Rk name change the default name of the const section
-Rz name change the default name of the bss section
-scanf (Select scanf capability)

Syntax
-scanf = c

Description
The -scanf option selects the scanf capability for the linked executable. The options are:

- -scanf=i[c] integer is supported, optional %[...] and %[^...] character class
- -scanf=l[c] long integer is supported, optional %[...] and %[^...] character class
- -scanf=ll[c] long long integer is supported, optional %[...] and %[^...] character class
- -scanf=d[ll][c] floating point is supported, optional long long and %[...] and %[^...] character class

Example
The minimal sized scanf

-scanf=i

The maximal functionality scanf

-scanf=dllc
-segger (Use SEGGER assembler/compiler/linker)

Syntax

-clang

Description

Use the SEGGER assembler, compiler and linker. Default is to use the GNU assembler, compiler and linker.
-shortenums (Minimal sized enums)

Syntax

-shortenums

Description

The -shortenums option instructs the compiler to set the size of an enumeration type to the smallest appropriate data type.
-shortwchar (16-bit wide chars)

Syntax

-shortwchar

Description

The -shortwchar option instructs the compiler to set the size of a wide character to 16-bit.
-simd (Generate vector processing code)

Syntax

-simd=neon

Description

The -simd option instructs the compiler to generate vector processing code.
-std (Select language standard)

Syntax

-std=s
-std=list

Description

The -std option sets the language standard to use.

Example

To set the language to c99 core you would use:

```
c c -std=c99
```

To list supported language standards:

```
c c -std=list
```
-strip (Strip symbols from executable)

Syntax

-stripsymbols

-stripdebug

Description

The -stripsymbols removes symbols from the linked executable.

The -stripdebug removes debugging information from the linked executable.
-symbols (Link symbols)

Syntax

-symbols=s

Description

The -symbols option supplies the symbols file s to the linker.
-thumb (Generate thumb code)

Syntax

-thumb

Description

The -thumb option instructs the compiler to generate thumb code rather than ARM code and link in thumb libraries. This option is NOT needed for Cortex-M architectures.
-T (Supply linker script)

Syntax

-Tfile

Description

The -T option supplies the file to the linker as a linker script.
-U (Undefine macro symbol)

Syntax

-U\textit{name}

Description

The -U option undefines the preprocessor macro \textit{name}. 
-unwindtables (Generate unwind tables)

Syntax

-unwindtables

Description

The -unwind option instructs the compiler to generate unwind tables.
-v (Verbose execution)

Syntax

-v

Description

The -v switch displays command lines executed by the compiler driver.
-vectorize (Generate vector processing code)

Syntax
- vectorize

Description
The -vectorize option instructs the compiler to generate vector processing code.
-w (Suppress warnings)

Syntax

-w

Description

This option instructs the compiler, assembler, and linker not to issue any warnings.
-we (Treat warnings as errors)

Syntax

-we

Description

This option directs the compiler, assembler, and linker to treat all warnings as errors.
-W (Pass option to tool)

Syntax

-W tool option

Description

The -W command-line option passes option directly to the specified tool. Supported tools are

- Wa pass option to assembler
- Wc pass option to compiler
- Wl pass option to linker

Example

The following example passes the (compiler specific) -version option to the compiler

```
cc -Wc-version
```
-x (Specify file types)

Syntax

-x type

Description

The -x option causes the compiler driver to treat subsequent files to be of the following file type:

-xa archives/libraries
-xasm assembly code files
-xc C code files
-xc++ C++ code files
-xo object code files

Example

The following command line enables an assembly code file with the extension .arm to be assembled.

```
cc -xasm a.arm
```
Command-Line Project Builder

`emBuild` is a program used to build your software from the command line without using SEGGER Embedded Studio. You can, for example, use `emBuild` for nightly (automated) builds, production builds, and batch builds.
Building with a SEGGER Embedded Studio project file

You can specify a SEGGER Embedded Studio project file:

**Syntax**

```
emBuild [options] project-file
```

You must specify a configuration to build using `-config`. For instance:

```
emBuild -config "V5T Thumb LE Release" arm.emProject
```

The above example uses the configuration **V5T Thumb LE Release** to build all projects in the solution contained in `arm.emProject`.

To build a specific project that is in a solution, you can specify it using the `-project` option. For example:

```
emBuild -config "V5T Thumb LE Release" -project "libm" libc.emProject
```

This example will use the configuration **V5T Thumb LE Release** to build the project `libm` that is contained in `libc.emProject`.

If your project file imports other project files (using the `<import>` mechanism), when denoting projects you must specify the solution names as a comma-separated list in parentheses after the project name:

```
emBuild -config "V5T Thumb LE Release" -project "libc(C Library)" arm.emProject
```

**libc(C Library)** specifies the `libc` project in the **C Library** solution that has been imported by the project file `arm.emProject`.

To build a specific solution that has been imported from other project files, you can use the `-solution` option. This option takes the solution names as a comma-separated list. For example:

```
emBuild -config "ARM Debug" -solution "ARM Targets,EB55" arm.emProject
```

In this example, **ARM Targets,EB55** specifies the **EB55** solution imported by the **ARM Targets** solution, which was itself imported by the project file `arm.emProject`.

You can do a batch build using the `-batch` option:

```
emBuild -config "ARM Debug" -batch libc.emProject
```

This will build the projects in `libc.emProject` that are marked for batch build in the configuration **ARM Debug**.

By default, a **make-style** build will be done, i.e., the dates of input files are checked against the dates of output files, and the build is avoided if the output is up to date. You can force a complete build by using the `-rebuild` option. Alternatively, to remove all output files, use the `-clean` option.
To see the commands being used in the build, use the -echo option. To also see why commands are being executed, use the -verbose option. You can see what commands will be executed, without executing them, by using the -show option.
Building without a SEGGER Embedded Studio project file

To use `emBuild` without a SEGGER Embedded Studio project, specify the name of an installed project template, the name of the project, and the files to build. For example:

```
emBuild -config -template LM3S_EXE -project myproject -file main.c
```

Or, instead of a template, you can specify a project type:

```
emBuild -config -type "Library" -project myproject -file main.c
```

You can specify project properties with the `-property` option:

```
emBuild -property Target=LM3S811
```
Command-line options

This section describes the command-line options accepted by emBuild.
-batch (Batch build)

Syntax

-batch

Description

Perform a batch build.
-config (Select build configuration)

Syntax

- config name

Description

Specify the configuration for a build. If the configuration name can’t be found, emBuild will list the available configurations.
-clean (Remove output files)

Syntax

-clean

Description

Remove all output files resulting from the build process.
-D (Define macro)

Syntax

-D macro=value

Description

Define a SEGGER Embedded Studio for ARM macro value for the build process.
-echo (Show command lines)

Syntax

-echo

Description

Show the command lines as they are executed.
-file (Build a named file)

Syntax

-file name

Description

Build the file name. Use with -template or -type.
-packagesdir (Specify packages directory)

Syntax

-packagesdir dir

Description

Override the default value of the $(PackagesDir) macro.
-project (Specify project to build)

Syntax

-project name

Description

Specify the name of the project to build. When used with a project file, if emBuild can't find the specified project, the names of available projects are listed.
-property (Set project property)

Syntax

-property name=value

Description

Specify the value of a project property use with -template or -type. If emBuild cannot find the specified property, a list of the properties is shown.
-rebuild (Always rebuild)

Syntax

-rebuild

Description

Always execute the build commands.
-show (Dry run, don't execute)

Syntax
-show

Description
Show the command lines that would be executed, but do not execute them.
-solution (Specify solution to build)

Syntax

-solution name

Description

Specify the name of the solution to build. If emBuild cannot find the given solution, the valid solution names are listed.
-studiodir (Specify SEGGER Embedded Studio directory)

Syntax

-studiodir name

Description

Override the default value of the $(StudioDir) macro.
-template (Specify project template)

Syntax

-template name

Description

Specify the project template to use. If emBuild cannot find the specified template then a list of template names is shown.
-time (Time the build)

Syntax

-time

Description

Show the time taken for the build.
-threadnum (Specify number of build threads)

Syntax

-threadnum \( n \)

Description

Specify the number of build threads to use for the build. The default is zero which will use the number of processor cores on your machine.
-type (Specify project type)

Syntax

-type name

Description

Specify the project type to use. If emBuild cannot find the specified project type then a list of project type names is shown.
-verbose (Show build information)

Syntax

-verbose

Description

Show extra information relating to the build process.
Command-Line Simulator

`emSim` is a program that allows you to run SEGGER Embedded Studio’s instruction set simulator from the command line.

The primary purpose of `emSim` is to enable command line tests to be run.

Syntax

`emSim` `file [options] args`
Command-line options

This section describes the command-line options accepted by emSim.
file (Elf executable file)

Description

This is the name of the elf file to run on the simulator. The file will be run until it makes a debug request to exit. The simulator will allocate memory regions based on the elf program sections.

Example

```
emSim app.elf
```
-segments (Specify memory segments)

Syntax

-segments start;size;

Description

Specify the memory segments to simulate.

Example

```
emSim app.elf -segments 0x08000000;0x10000;0x20000000;0x10000
```
args (User arguments)

Description

The arguments supplied to the elf file in the argc/argv parameters to the main function.

emSim app.elf hello world
Command-Line Scripting

`emScript` is a program that allows you to run SEGGER Embedded Studio's JavaScript (ECMAScript) interpreter from the command line.

The primary purpose of `emScript` is to facilitate the creation of platform-independent build scripts.

**Syntax**

`emScript [options] file`
Command-line options

This section describes the command-line options accepted by emScript.
-define (Define global variable)

Syntax

-define variable=value

Description
-help (Show usage)

Syntax

-help

Description

Display usage information and command line options.
-load (Load script file)

Syntax

-load path

Description

Loads the script file path.
-define (Verbose output)

**Syntax**

-verbose

**Description**

Produces verbose output.
emScript classes

translate
emScript provides the following predefined classes:

- BinaryFile
- CWSys
- ElfFile
- WScript
Example uses

The following example demonstrates using emScript to increment a build number:

First, add a JavaScript file to your project called incbuild.js containing the following code:

```javascript
function incbuild()
{
    var file = "buildnum.h"
    var text = "#define BUILDNUMBER "
    var s = CWSys.readStringFromFile(file);
    var n;
    if (s == undefined)
        n = 1;
    else
        n = eval(s.substring(text.length)) + 1;
    CWSys.writeStringToFile(file, text + n);
}

// Executed when script loaded.
incbuild();
```

Add a file called getbuildnum.h to your project containing the following code:

```c
#ifndef GETBUILDNUM_H
#define GETBUILDNUM_H

unsigned getBuildNumber();

#endif
```

Add a file called getbuildnum.c to your project containing the following code:

```c
#include "getbuildnum.h"
#include "buildnum.h"

unsigned getBuildNumber()
{
    return BUILDNUMBER;
}
```

Now, to combine these:

Set the Build Options > Always Rebuild project property of getbuildnum.c to Yes.
Set the User Build Step Options > Pre-Compile Command project property of getbuildnum.c to "$\{StudioDir\}/bin/emScript" -load "$\{ProjectDir\}/incbuild.js".
Embed

Embed is a program that converts a binary file into a C/C++ array definition.

The primary purpose of the Embed tool is to provide a simple method of embedding files into an application. This may be useful if you want to include firmware images, bitmaps, etc. in your application without having to read them first from an external source.

Syntax

`embed variable_name input_file output_file`

`variable_name` is the name of the C/C++ array to be initialised with the binary data.

`input_file` is the path to the binary input file.

`output_file` is the path to the C/C++ source file to generate.

Example

To convert a binary file `image.bin` to a C/C++ file called `image.h`:

```
embed img image.bin image.h
```

This will generate the following output in `image.h`:

```
static const unsigned char img[] = {
  0x5B, 0x95, 0xA4, 0x56, 0x16, 0x5F, 0x2D, 0x47,
  0xC5, 0x04, 0x8D, 0x73, 0x40, 0x31, 0x66,
  0x3E, 0x81, 0x90, 0x39, 0xA3, 0x8E, 0x22, 0x37,
  0x3C, 0x63, 0xC8, 0x30, 0x90, 0xC0, 0x54, 0xA4,
  0xA2, 0x74, 0xC2, 0x8C, 0x1D, 0x56, 0x57, 0x05,
  0x45, 0xCE, 0x92, 0xA0, 0x08, 0x2C, 0x39,
  0x92, 0x59, 0xA9, 0x9D, 0x01, 0x30, 0x59, 0x9F,
  0xC5, 0xEA, 0xCE, 0x35, 0x4F, 0x4B, 0x05, 0xBF
};
```
Header file generator

The command line program `mkhdr` generates a C or C++ header file from a SEGGER Embedded Studio for ARM memory map file.
Using the header generator

For each register definition in the memory map file a corresponding `#define` is generated in the header file. The `#define` is named the same as the register name and is defined as a volatile pointer to the address.

The type of the pointer is derived from the size of the register. A four-byte register generates an unsigned long pointer. A two-byte register generates an unsigned short pointer. A one-byte register will generates an unsigned char pointer.

If a register definition in the memory map file has bitfields then preprocessor symbols are generated for each bitfield. Each bitfield will have two preprocessor symbols generated, one representing the mask and one defining the start bit position. The bitfield preprocessor symbol names are formed by prepending the register name to the bitfield name. The mask definition has `_MASK` appended to it and the start definition has `_BIT` appended to it.

For example consider the following definitions in the the file `memorymap.xml`.

```xml
<RegisterGroup start="0xFFFFF000" name="AIC">
  <Register start="+0x00" size="4" name="AIC_SMR0">
    <BitField size="3" name="PRIOR" start="0"/>
    <BitField size="2" name="SRCTYPE" start="5"/>
  </Register>
  ...
</RegisterGroup>
```

We can generate the header file associated with this file using:

```bash
mkhdr memorymap.xml memorymap.h
```

This generates the following definitions in the file `memorymap.h`.

```c
#define AIC_SMR0 (*((volatile unsigned long *)0xFFFFF000))
#define AIC_SMR0_PRIOR_MASK 0x7
#define AIC_SMR0_PRIOR_BIT 0
#define AIC_SMR0_SRCTYPE_MASK 0x60
#define AIC_SMR0_SRCTYPE_BIT 5
```

These definitions can be used in the following way in a C/C++ program:

Reading a register

```c
unsigned r = AIC_SMR0;
```

Writing a register

```c
AIC_SMR0 = (priority << AIC_SMR0_PRIOR_BIT) | (srctype << AIC_SMR0_SRCTYPE_BIT);
```

Reading a bitfield

```c
unsigned srctype = (AIC_SMR0 & AIC_SMR0_SRCTYPE_MASK) >> AIC_SMR0_SRCTYPE_BIT;
```

Writing a bitfield

```c
AIC_SMR0 = (AIC_SMR0 & ~AIC_SMR0_SRCTYPE_MASK) | (srctype & AIC_SMR0_SRCTYPE_MASK) << AIC_SMR0_SRCTYPE_BIT;
```
Command line options

This section describes the command line options accepted by the header file generator.

Syntax

`mkhdr inputfile outfile targetname [option]`

`inputfile` is the name of the source SEGGER Embedded Studio for ARM memory map file. `outfile` is the name of the file to write.
-regbaseoffsets (Use offsets from peripheral base)

Syntax

-regbaseoffsets

Description

Instructs the header generator to include offsets of registers from the peripheral base.
-nobitfields (Inhibit bitfield macros)

Syntax

-nobitfields

Description

Instructs the header generator not to generate any definitions for bitfields.
Linker script file generator

The command line program mkld generates a GNU ld linker script from a SEGGER Embedded Studio for ARM memory map or section placement file.

Syntax

mkld -memory-map-file inputfile outputfile [options]

inputfile is the name of the SEGGER Embedded Studio for ARM memory map file to generate the ld script from.
outputfile is the name of the ld script file to write.
Command-line options

This section describes the command-line options accepted by `mkld`. 
-check-segment-overflow

Syntax

-check-segment-overflow

Description

Add checks for memory segment overflow to the linker script.
-disable-missing-runin-error

Syntax

-disable-missing-runin-error

Description

Discard any sections with a missing run in section.
-memory-map-file

Syntax

-memory-map-file filename

Description

Generate a GNU ld linker script from the SEGGER Embedded Studio for ARM memory map file filename.
-memory-map-macros

Syntax

-**memory-map-macros** *macro=value[;macro=value]*

Description

Define SEGGER Embedded Studio for ARM macros to use when reading a memory map file.
-no-check-unplaced-sections

Syntax

-no-check-unplaced-sections

Description

Removes checks for unplaced memory sections from the linker script.
-section-placement-file

Syntax

-section-placement-file filename

Description

Generate a GNU ld linker script from the SEGGER Embedded Studio for ARM section placement file filename. If this option is used, a memory map file should also be specified with the -memory-map-file option.
-section-placement-macros

Syntax

-section-placement-macros macro=value [ ; macro=value ]

Description

Define SEGGER Embedded Studio for ARM macros to use when reading a section placement file.
-symbols

Syntax

-symbols symbol=value[;symbol=value]

Description

Add extra symbol definitions to the ld linker script.
Package generator

To create a package the program `mkpkg` can be used. The set of files to put into the package should be in the desired location in the `$ PackagesDir ` directory. The `mkpkg` command should be run with `$ PackagesDir ` as the working directory and all files to go into the package must be referred to using relative paths. A package must have a package description file that is placed in the `$ PackagesDir `/ packages directory. The package description file name must end with `.package.xml`. If a package is to create entries in the new project wizard then it must have a file name `project_templates.xml`. For example, a package for the mythical FX150 processor would supply the following files:

- A project template file called `targets/FX150/project_templates.xml`. The format of the project templates file is described in [Project Templates file format](#).
- The `$ PackagesDir `-relative files that define the functionality of the package.
- A package description file called `packages/FX150_package.xml`. The format of the package description file is described in [Package Description file format](#).

The package file `FX150.emPackage` would be created using the following command line:

```
mkpkg -c packages/FX150.emPackage targets/FX150/project_templates.xml  packages/FX150_package.xml
```

You can list the contents of the package using the `-t` option:

```
mkpkg -t packages/FX150.emPackage
```

You can remove an entry from a package using the `-d` option:

```
mkpkg -d packages/FX150.emPackage -d fileName
```

You can add or replace a file into an existing package using the `-r` option:

```
mkpkg -r packages/FX150.emPackage -r fileName
```

You can extract files from an existing package using the `-x` option:

```
mkpkg -x packages/FX150.emPackage outputDirectory
```

You can automate the package creation process using a Combining project type.

Using the new project wizard create a combining project in the directory `$ PackagesDir `. Set the Output File Path property to be `$ PackagesDir /packages/mypackage.emPackage `. Set the Combine command property to `$ (StudioDir)/bin/mkpkg -c $(CombiningOutputFilePath) $(CombiningRelInputPaths) `. Add the files you want to go into the package into the project using the Project Explorer. Right-click the project node in the Project Explorer and choose Build.

When a package is installed, the files in the package are copied into the desired `$ PackagesDir `-relative locations. When a file is copied into the `$ PackagesDir /packages ` directory and its filename ends with
the file $(PackagesDir)/packages/installed_packages.xml is updated with an entry:

```xml
  <include filename="FX150_package.xml" />
```

During development of a package you can manually edit this file. The same applies to the file $(PackagesDir)/targets/project_templates.xml which will contain a reference to your project_templates.xml file.

**Usage:**

```
mkpkg [options] packageFileName file1 file2
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c</td>
<td>Create a new package.</td>
</tr>
<tr>
<td>-compress level</td>
<td>Change compression level (0 for none, 9 for maximum).</td>
</tr>
<tr>
<td>-d</td>
<td>Remove files from a package.</td>
</tr>
<tr>
<td>-f</td>
<td>Output files to stdout.</td>
</tr>
<tr>
<td>-r</td>
<td>Replace files in a package.</td>
</tr>
<tr>
<td>-readonly</td>
<td>Force all files to have read only attribute.</td>
</tr>
<tr>
<td>-t</td>
<td>List the contents of a package.</td>
</tr>
<tr>
<td>-v</td>
<td>Be chatty.</td>
</tr>
<tr>
<td>-V</td>
<td>Show version information.</td>
</tr>
<tr>
<td>-x</td>
<td>Extract files from a package.</td>
</tr>
</tbody>
</table>
Appendices
File formats

This section describes the file formats SEGGER Embedded Studio for ARM uses:

Memory Map file format
   Describes the memory map file format that defines memory regions and registers in a microcontroller.

Section Placement file format
   Describes the section placement file format that maps program sections to memory areas in the target microcontroller.

Project file format
   Describes the format of SEGGER Embedded Studio project files.

Project Templates file format
   Describes the format of project template files used by the New Project wizard.

Property Groups file format
   Describes the format of the property groups file you can use to define meta-properties.

Package Description file format
   Describes the format of the package description files you use to create packages other users can install in SEGGER Embedded Studio.

External Tools file format
   Describes the format of external tool configuration files you use to extend SEGGER Embedded Studio.

Debugger Type Interpretation file format
   Describes the format of the debugger type interpretation file.
Memory Map file format

SEGGER Embedded Studio memory-map files are structured using XML syntax for its simple construction and parsing.

The first entry of the project file defines the XML document type used to validate the file format.

```
<!DOCTYPE Board_Memory_Definition_File>
```

The next entry is the Root element. There can only be one Root element in a memory map file:

```
<Root name="My Board">
```

A Root element has a name attribute every element in a memory map file has a name attribute. Names should be unique within a hierarchy level. Within a Root element, there are MemorySegment elements that represent regions within the memory map.

```
<Root name="My Board">
  <MemorySegment name="Flash" start="0x1000" size="0x200" access="ReadOnly">
```

MemorySegment elements have the following attributes:

- **start:** The start address of the memory segment. A simple expression, usually a hexadecimal number with a 0x prefix.
- **size:** The size of the memory segment. A simple expression, usually a hexadecimal number with a 0x prefix.
- **access:** The permissible access types of the memory segment. One of ReadOnly, Read/Write, WriteOnly, or None.
- **address_symbol:** A symbolic name for the start address of the memory segment.
- **size_symbol:** A symbolic name for the size of the memory segment.
- **address_symbol:** A symbolic name for the end address of the memory segment.

RegisterGroup elements are used to organize registers into groups. Register elements are used to define peripheral registers:

```
<Root name="My Board">
  <MemorySegment name="System" start="0x2000" size="0x200">
    <RegisterGroup name="Peripheral" start="0x2100" size="0x10">
      <Register name="Register1" start="+0x8" size="4">
```

RegisterGroup elements have the same attributes as MemorySegment elements. Register elements have the following attributes:

- **name:** Register names should be valid C/C++ identifier names, i.e., alphanumeric characters and underscores are allowed but names cannot start with a number.
- **start:** The start address of the memory segment. Either a C-style hexadecimal number or, if given a + prefix, an offset from the enclosing element’s start address.
- **size:** The size of the register in bytes, either 1, 2, or 4.
access: The same as the `access` attribute of the `MemorySegment` element.

`address_symbol`: The same as the `address_symbol` attribute of the `MemorySegment` element.

A `Register` element can contain `BitField` elements that represent the bits in a peripheral register:

```xml
<Register name="Register1" start="+0x8" size="4">
  <BitField name="Bits_0_to_3" start="0" size="4"/>
</Register>
```

`BitField` elements have the following attributes:

- `name`: The same as the `name` attribute of the `RegisterGroup` element.
- `start`: The starting bit position, 031.
- `size`: The total number of bits, 132.

A `Bitfield` element can contain `Enum` elements:

```xml
<Register name="Register1" start="+0x8" size="4">
  <BitField name="Bits_0_to_3" start="0" size="4"/>
  <Enum name="Enum3" start="3"/>
  <Enum name="Enum5" start="5"/>
</Register>
```

You can import CMSIS SVD files (see [http://www.onarm.com/](http://www.onarm.com/)) into a memory map using the `ImportSVD` element:

```xml
<ImportSVD filename="$(TargetsDir)/targets/Manufacturer1/Processor1.svd.xml">
```

The `filename` attribute is an absolute filename which is macro-expanded using SEGGER Embedded Studio for ARM system macros.

When a memory map file is loaded either for the memory map viewer or to be used for linking or debugging, it is preprocessed using the (as yet undocumented) SEGGER Embedded Studio for ARM XML preprocessor.
Section Placement file format

SEGGER Embedded Studio section-placement files are structured using XML syntax to enable simple construction and parsing.

The first entry of the project file defines the XML document type used to validate the file format:

```xml
<!DOCTYPE Linker_Placement_File>
```

The next entry is the `Root` element. There can only be one `Root` element in a memory map file:

```xml
<Root name="Flash Placement">
```

A `Root` element has a `name` attribute. Every element in a section-placement file has a `name` attribute. Each name should be unique within its hierarchy level. Within a `Root` element, there are `MemorySegment` elements. These correspond to memory regions defined in a memory map file that will be used in conjunction with the section-placement file when linking a program. For example:

```xml
<Root name="Flash Placement">
  <MemorySegment name="FLASH">
```

A `MemorySegment` contains `ProgramSection` elements that represent program sections created by the C/C++ compiler and assembler. The order of `ProgramSection` elements within a `MemorySegment` element represents the order in which the sections will be placed when linking a program. The first `ProgramSection` will be placed first and the last one will be placed last.

```xml
<Root name="My Board">
  <MemorySegment name="FLASH">
    <ProgramSection name=".text">
```

`ProgramSection` elements have the following attributes:

- `address_symbol`: A symbolic name for the start address of the section.
- `alignment`: The required alignment of the program section; a decimal number specifying the byte alignment.
- `end_symbol`: A symbolic name for the end address of the section.
- `fill`: The optional value used to fill unspecified regions of memory, a hexadecimal number with a 0x prefix.
- `inputsections`: An expression describing the input sections to be placed in this section. If you omit this (recommended) and the section name isn't one of `.text`, `.dtors`, `.ctors`, `.data`, `.rodata`, or `.bss`, then the equivalent input section of `*(.name .name.*)` is supplied to the linker.
- `keep`: If `Yes`, the section will be kept even if none of the symbols are referenced by the rest of the program.
- `load`: If `Yes`, the section is loaded. If `No`, the section isn't loaded.
- `place_from_segment_end`: If `Yes`, this section and following sections will be placed at the end of the segment. Please note that this will only succeed if the section and all following sections have a fixed size specified with the `size` attribute.
- `runin`: This specifies the name of the section to copy this section to. Multiple sections can be specified separated by a semicolon, the first section that exists will be used.
runoffset: This specifies an offset from the load address that the section will be run from.
size: The optional size of the program section in bytes, a hexadecimal number with a 0x prefix.
size_symbol: A symbolic name for the size of the section.
start: The optional start address of the program section, a hexadecimal number with a 0x prefix.

When a section placement file is used for linking it is preprocessed using the (as yet undocumented) SEGGER Embedded Studio for ARM XML preprocessor.
Project file format

SEGGER Embedded Studio project files are held in text files with the .emProject extension. Because you may want to edit project files, and perhaps generate them, they are structured using XML syntax to enable simple construction and parsing.

The first entry of the project file defines the XML document type used to validate the file format:

```
<!DOCTYPE CrossStudio_Project_File>
```

The next entry is the solution element; there can only be one solution element in a project file. This specifies the solution name displayed in the Project Explorer and has a version attribute that defines the file-format version of the project file. Solutions can contain projects, projects can contain folders and files, and folders can contain folders and files. This hierarchy is reflected in the XML nesting for example:

```
<solution version="1" Name="solutionname">
  <project Name="projectname">
    <file Name="filename" />
    <folder Name="foldername">
      <file Name="filename2" />
    </folder>
  </project>
</solution>
```

Note that each entry has a Name attribute. Names of project elements must be unique to the solution, and names of folder elements must be unique to the project, but names of files do not need to unique.

Each file element must have a file_name attribute that is unique to the project. Ideally, the file_name is a file path relative to the project (or solution directory), but you can also specify a full file path, if you want to. File paths are case-sensitive and use / as the directory separator. They may contain macro instantiations, so file paths cannot contain the "$" character. For example:

```
<file file_name="$(StudioDir)/source/crt0.s" Name="crt0.s" />
```

will be expanded using the value of $(StudioDir) when the file is referenced from SEGGER Embedded Studio.

Project properties are held in configuration elements with the Name attribute of the configuration element corresponding to the configuration name, e.g., "Debug". At a given project level (i.e., solution, project, folder), there can only be one named configuration element. i.e., all properties defined for a configuration are in single configuration element.

```
<project Name="projectname">
  <configuration project_type="Library" Name="Common" />
  <configuration Name="Release" build_debug_information="No" />
</project>
```

You can use the import element to link projects:

```
<import file_name="target/libc.emProject" />
```
Project Templates file format

The SEGGER Embedded Studio New Project dialog works from a file called project_templates.xml in the targets subdirectory of the SEGGER Embedded Studio installation directory. Because you may want to add your own new project types, they are structured using XML syntax to enable simple construction and parsing.

The first entry of the project file defines the XML document type used to validate the file format:

```xml
<!DOCTYPE Project_Templates_File>
```

The next entry is the projects element, which is used to group a set of new project entries into an XML hierarchy.

```xml
<projects>
  <project/>
</projects>
```

Each entry has a project element that contains the class of the project (attribute caption), the name of the project (attribute name), its type (attribute type) and a description (attribute description). For example:

```xml
<project caption="ARM Evaluator7T" name="Executable"
  description="An executable for an ARM Evaluator7T." type="Executable"/>
```

The project type can be one of these:

- **Executable**: a fully linked executable.
- **Library**: a static library.
- **Object file**: an object file.
- **Staging**: a staging project.
- **Combining**: a combining project.
- **Externally Built Executable**: an externally built executable.

The configurations to be created for the project are defined using the configuration element, which must have a name attribute:

```xml
<configuration name="ARM RAM Release"/>
```

The property values to be created for the project are defined using the property element. If you have a defined value, you can specify this using the value attribute and, optionally, set the property in a defined configuration, such as:

```xml
<property name="target_reset_script" configuration="RAM"
  value="Evaluator7T_ResetWithRamAtZero()"/>
```

Alternatively, you can include a property that will be shown to the user, prompting them to supply a value as part of the new-project process.

```xml
<property name="linker_output_format"/>
```
The folders to be created are defined using the `folder` element. The `folder` element must have a `name` attribute and can also have a `filter` attribute. For example:

```xml
<folder name="Source Files" filter="c;cpp;cxx;cc;h;s;asm;inc"/>
```

The files to be in the project are specified using the `file` element. You can use build-system macros (see [Project macros](#)) to specify files located in the SEGGER Embedded Studio installation directory. Files will be copied to the project directory or just left as references, depending on the value of the `source` attribute:

```xml
<file name="main.c" source="$(StudioDir)/samples/Shared/main.c"/>
<file name="$(StudioDir)/source/thumb_crt0.s"/>
```

You can define the set of configurations that can be referred to in the top-level `configurations` element:

```xml
<configurations>
  <configuration>
  </configuration>
</configurations>
```

This contains the set of all configurations that can be created when a project is created. Each configuration is defined using a `configuration` element, which can define the property values for that configuration. For example:

```xml
<configuration name="Debug">
  <property name="build_debug_information" value="Yes">
```

965
Property Groups file format

The SEGGER Embedded Studio project system provides a means to create new properties that change a number of project property settings and can also set C pre-processor definitions when selected. Such properties are called property groups and are defined in a property-groups file. The property-group file to use for a project is defined by the Property Groups File property. These files usually define target-specific properties and are structured using XML syntax to enable simple construction and parsing.

The first entry of the property groups file defines the XML document type, which is used to validate the file format:

```xml
<!DOCTYPE CrossStudio_Group_Values>
```

The next entry is the propertyGroups element, which is used to group a set of property groups entries into an XML hierarchy:

```xml
<propertyGroups>
  <grouphdots
    <grouphdots
  </propertyGroups>
```

Each group has the name of the group (attribute name), the name of the options category (attribute group), short (attribute short) and long (attribute long) help descriptions, and a default value (attribute default). For example:

```xml
<group short="Target Processor" group="Build Options" short="Target Processor"
  long="Select a set of target options" name="Target" default="STR912FW44" />
```

Each group has a number of groupEntry elements that define the enumerations of the group.

```xml
<group>
  <groupEntry>
    <groupEntry>
  </group>
```

Each groupEntry has the name of the entry (attribute name), e.g.:

```xml
<groupEntry name="STR910FW32">
```

A groupEntry has the property values and C pre-processor definitions that are set when the groupEntry is selected; they are specified with property and cdefine elements. For example:

```xml
<groupEntry>
  <property>
    <cdefine>
    <property>
    <cdefine>
  </groupEntry>
```
A property element has the property's name (attribute name), its value (attribute value), and an optional configuration (attribute configuration):

```xml
<property name="linker_memory_map_file" value="${StudioDir}/targets/ST_STR91x/ST_STR910FM32_MemoryMap.xml" />
```

A define element has the C preprocessor name (attribute name) and its value (attribute value):

```xml
<define value="STR910FM32" name="TARGET_PROCESSOR" />
```
Package Description file format

Package-description files are XML files used by SEGGER Embedded Studio to describe a support package, its contents, and any dependencies it has on other packages.

Each package file must contain one package element that describes the package. Optionally, the package element can contain a collection of file, history, and documentation elements to be used by SEGGER Embedded Studio for documentation purposes.

The filename of the package-description file should match that of the package and end in ".package.xml".

Below is an example of two package-description files. The first is for a base chip-support package for the LPC2000; the second is for a board-support package dependent on the first:

Philips_LPC2000_package.xml

```xml
<!DOCTYPE CrossStudio_Package_Description_File>
<package cpu_manufacturer="Philips" cpu_family="LPC2000" version="1.1" ses_versions="8:1-" author="SEGGER">
  <file file_name="$(TargetsDir)/Philips_LPC210X/arm_target_Philips_LPC210X.htm" title="LPC2000 Support Package Documentation" />
  <file file_name="$(TargetsDir)/Philips_LPC210X/Loader.emProject" title="LPC2000 Loader Application Solution" />
  <group title="System Files">
    <file file_name="$(TargetsDir)/Philips_LPC210X/Philips_LPC210X_Startup.s" title="LPC2000 Startup Code" />
    <file file_name="$(TargetsDir)/Philips_LPC210X/Philips_LPC210X_Target.js" title="LPC2000 Target Script" />
  </group>
  <history>
    <version name="1.1">
      <description>Corrected LPC21xx header files and memory maps to include GPIO ports 2 and 3.</description>
    </version>
    <version name="1.0">
      <description>Initial Release.</description>
    </version>
  </history>
  <documentation>
    <section name="Supported Targets">
      <p>This CPU support package supports the following LPC2000 targets:
        <ul>
          <li>LPC2103</li>
          <li>LPC2104</li>
          <li>LPC2105</li>
          <li>LPC2106</li>
          <li>LPC2131</li>
          <li>LPC2132</li>
          <li>LPC2134</li>
          <li>LPC2136</li>
          <li>LPC2138</li>
        </ul>
      </p>
    </section>
  </documentation>
</package>
```
CrossFire_LPC2138_package.xml

```xml
<package cpu_manufacturer="Philips" cpu_family="LPC2000" cpu_name="LPC2138"
    board_manufacturer="Rowley Associates" board_name="CrossFire LPC2138"
    dependencies="Philips_LPC2000" version="1.0">
    <file file_name="$(SamplesDir)/CrossFire_LPC2138/CrossFire_LPC2138.emProject"
        title="CrossFire LPC2138 Samples Solution" />
    <file file_name="$(SamplesDir)/CrossFire_LPC2138/ctl/ctl.emProject" title="CrossFire LPC2138 CTL Samples Solution" />
</package>
```

Package elements

The `package` element describes the support package, its contents, and any dependencies it has on other packages. Valid attributes for this element are:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>author</td>
<td>The author of the package.</td>
</tr>
<tr>
<td>board_manufacturer</td>
<td>The manufacturer of the board supported by the package (if omitted, CPU manufacturer will be used).</td>
</tr>
<tr>
<td>board_name</td>
<td>The name of the specific board supported by the package (only required for board-support packages).</td>
</tr>
<tr>
<td>company_name</td>
<td>The name of the company to group the package under in the package dialogs. (if omitted, the Board/CPU manufacturer will be used).</td>
</tr>
<tr>
<td>cpu_family</td>
<td>The family name of the CPU supported by the package (optional).</td>
</tr>
<tr>
<td>cpu_manufacturer</td>
<td>The manufacturer of the CPU supported by the package.</td>
</tr>
<tr>
<td>cpu_name</td>
<td>The name of the specific CPU supported by the package (may be omitted if the CPU family is specified).</td>
</tr>
<tr>
<td>ses_versions</td>
<td>A string describing which version of SEGGER Embedded Studio supports the package (optional). The format of the string is <code>target_id_number:version_range_string</code>.</td>
</tr>
<tr>
<td>description</td>
<td>A description of the package (optional).</td>
</tr>
<tr>
<td>dependencies</td>
<td>A semicolon-separated list of packages the package requires to be installed in order to work.</td>
</tr>
<tr>
<td>installation_directory</td>
<td>The directory in which the package should be installed (optional - if undefined, defaults to &quot;${PackagesDir}$&quot;).</td>
</tr>
<tr>
<td>title</td>
<td>A short description of the package (optional).</td>
</tr>
</tbody>
</table>
version  | The package version number.

**File elements**

The `file` element is used by SEGGER Embedded Studio for documentation purposes by adding links to files of interest within the package such as example project files and documentation.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_name</td>
<td>The file path of the file.</td>
</tr>
<tr>
<td>title</td>
<td>A description of the file.</td>
</tr>
</tbody>
</table>

Optionally, `file` elements can be grouped into categories using the `group` element.

**Group elements**

The `group` element is used for categorizing files described by `file` elements into a particular group.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>Title of the group.</td>
</tr>
</tbody>
</table>

**History elements**

The `history` element is used to hold a description of the package’s version history.

The `history` element should contain a collection of `version` elements.

**Version element**

The `version` element is used to hold the description of a particular version of the package.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the version being described.</td>
</tr>
</tbody>
</table>

The `version` element should contain a collection of `description` elements.

**Description elements**

Each `description` element contains text that describes a feature of the package version.
Documentation elements

The documentation element is used to provide arbitrary documentation for the package.

The documentation element should contain a collection of one or more section elements.

Section elements

The section element contains package documentation in XHTML format.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The title of the documentation section.</td>
</tr>
</tbody>
</table>

**target_id_number**

The following table lists the possible target ID numbers:

<table>
<thead>
<tr>
<th>Target</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>8</td>
</tr>
<tr>
<td>RISC-V</td>
<td>20</td>
</tr>
</tbody>
</table>

**version_range_string**

The version_range_string can be any of the following:

- `version_number`: The package will only work on `version_number`.
- `version_number-`: The package will work on `version_number` or any future version.
- `-version_number`: The package will work on `version_number` or any earlier version.
- `low_version_number-high_version_number`: The package will work on `low_version_number`, `high_version_number` or any version in between.
External Tools file format

SEGGER Embedded Studio external-tool configuration files are structured using XML syntax for its simple construction and parsing.

Tool configuration files

The SEGGER Embedded Studio application will read the tool configuration file when it starts up. By default, SEGGER Embedded Studio will read the file 

```text
$(StudioUserDir)/tools.xml
```

Structure

All tools are wrapped in a `tools` element:

```xml
<tools>
  <item name="logical name">
    ...
  </item>
</tools>
```

Inside the `tools` element are `item` elements that define each tool:

```xml
<tools>
  <item name="logical name">
    ...
  </item>
</tools>
```

The `item` element requires an `name` attribute, which is an internal name for the tool, and has an optional `wait` element. When SEGGER Embedded Studio invokes the tool on a file or project, it uses the `wait` element to determine whether it should wait for the external tool to complete before continuing. If the `wait` attribute is not provided or is set to `yes`, SEGGER Embedded Studio will wait for external tool to complete.

The way that the tool is presented in SEGGER Embedded Studio is configured by elements inside the `item` element.

menu

The `menu` element defines the wording used inside menus. You can place a shortcut to the menu using an ampersand, which must be escaped using `&amp` in XML, before the shortcut letter. For instance:

```xml
<menu>&amp;PC-lint (Unit Check)</menu>
```

text

The optional `text` element defines the wording used in contexts other than menus, for instance when the tool appears as a tool button with a label. If `text` is not provided, the tool’s textual appearance outside the menu is taken from the `menu` element (and is presented without an shortcut underline). For instance:
tip

The optional **tip** element defines the status tip, shown on the status line, when moving over the tool inside SEGGER Embedded Studio:

```xml
<tip>Run a PC-lint unit checkout on the selected file or folder</tip>
```

key

The optional **key** element defines the accelerator key, or key chord, to use to invoke the tool using the keyboard. You can construct the key sequence using modifiers Ctrl, Shift, and Alt, and can specify more than one key in a sequence (note: Windows and Linux only; OS X does not provide key chords). For instance:

```xml
<key>Ctrl+L, Ctrl+I</key>
```

message

The optional **message** element defines the text shown in the tool log in SEGGER Embedded Studio when running the tool. For example:

```xml
<message>Linting</message>
```

match

The optional **match** element defines which documents the tool will operate on. The match is performed using the file extension of the document. If the file extension of the document matches one of the wildcards provided, the tool will run on that document. If there is no **match** element, the tool will run on all documents. For instance:

```xml
<match>*.c;*.cpp</match>
```

commands

The **commands** element defines the command line to run to invoke the tool. The command line is expanded using macros applicable to the file derived from the current build configuration and the project settings. Most importantly, the standard **$(InputPath)** macro expands to a full pathname for the target file.

Additional macros constructed by SEGGER Embedded Studio are:

- **$(DEFINES)** is the set of -D options applicable to the current file, derived from the current configuration and project settings.
- **$(INCLUDES)** is the set of -I options applicable to the current file, derived from the current configuration and project settings.

For instance:
<commands>
  $(LINTDIR)/lint-nt $(LINTDIR)/lint $(LINTDIR)/co-gcc
  $(DEFINES) $(INCLUDES) -D__GNUC__ -u -b +macros -w2 -e537 +fie +ffn -width(0,4) -hF1
  $(InputPath)
</commands>

In this example we intend $(LINTDIR) to point to the directly where PC-lint is installed and for $(LINTDIR) to be defined as a SEGGER Embedded Studio global macro. You can set global macros using Tools > Options > Building > Global Macros.

Note that additional " entities are placed around pathnames in the commands section this is to ensure that paths that contain spaces are correctly interpreted when the command is executed by SEGGER Embedded Studio.
Debugger Type Interpretation file format

SEGGER Embedded Studio debugger type interpretation files are used by the debugger to provide list and string displays of C++ template container types. The files are structured using XML syntax for its simple construction and parsing.

Consider the following C++ template type

```cpp
template <class _Type>
class VeryBasicArray
{
private:
    size_t m_Count;
    _Type * m_pData;
public:
    VeryBasicArray(size_t count)
        : m_Count(count),
          m_pData(new _Type[count])
    {
    }

VeryBasicArray<int> basicArray(5);
```

To display a variable of this type as a list the type interpretation file contains the following entry

```xml
<List Name="VeryBasicArray<*>"
    Head="(($(T)*)HEAD).m_pData"
    Data="*($(T0)*)CURRENT"
    Length="($(T)*)HEAD.m_Count"
    Next="CURRENT+sizeof($(T0))"/>
```

The **Name** attribute is used to match the template type name note that the &lt and &gt xml entities are used to match the template argument.

When an entry has been matched the head of the list is located by evaluating the debugger expression in the **Head** attribute. The debugger expressions can contain macros that refer to the matched template type and will use the symbols HEAD and CURRENT.

The macro $(T) refers to the instantiated template type, for the above example $(T)=VeryBasicArray<int>.

The template arguments are referred to using macros $(T0), for the above example $(T0)=int.

The symbol HEAD is the address of the variable being displayed, for the above example if the variable `basicArray` is allocated at address 0x20004000 then the **Head** expression

```cpp
((VeryBasicArray<int>*)0x20004000).m_pData
```

will be evaluated by the debugger, note that the . operator and the -> operator are equivalent in debugger expressions.

To display an element the debugger will evaluate the **Data** expression. This expression contains the symbol CURRENT which is the address of the element to display, for the above example the first element is at the address `basicArray.m_pData` which is allocated at address 0x20008000 then the **Data** expression
will be evaluated by the debugger.

To increment the **CURRENT** symbol the **Next** expression

```c
0x20008000 + sizeof(int)
```

will be evaluated by the debugger.

Before the **CURRENT** symbol is incremented the debugger needs to check if it is at the end of list. The can be done either as a **Condition** expression or as a **Length** expression

```c
((VeryBasicArray<int> */)0x20004000).m_Count
```

The **String** display is simpler than the **List** display since the characters are contiguous and optionally null terminated. The **Data** and **Length** expressions are supported, for example

```c
<String Name="string"
  Data="*(((T) *)HEAD)._M_start_of_storage._M_data"
  Length="*((T) *)HEAD)._M_finish-((T) *)HEAD)._M_start_of_storage._M_data"/>
```

is used to display STLPort std::string types.
## Building Environment Options

### Build

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Automatically Build Before Debug</strong></td>
<td>Enables auto-building of a project before downloading if it is out of date.</td>
</tr>
<tr>
<td>Environment/Build/Build Before Debug</td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Confirm Debugger Stop</strong></td>
<td>Present a warning when you start to build that requires the debugger to stop.</td>
</tr>
<tr>
<td>Environment/Build/Confirm Debugger Stop</td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Display ETA</strong></td>
<td>Selects whether to attempt to compute and display the ETA on building.</td>
</tr>
<tr>
<td>Environment/Build/Display ETA</td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Display Progress Bar</strong></td>
<td>Selects whether to display progress bar on building.</td>
</tr>
<tr>
<td>Environment/Build/Display Progress Bar</td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Echo Build Command Lines</strong></td>
<td>Selects whether build command lines are written to the build log.</td>
</tr>
<tr>
<td>Environment/Build/Show Command Lines</td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Echo Raw Error/Warning Output</strong></td>
<td>Selects whether the unprocessed error and warning output from tools is displayed in the build log.</td>
</tr>
<tr>
<td>Environment/Build/Show Unparsed Error Output</td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Find Error After Building</strong></td>
<td>Moves the cursor to the first diagnostic after a build completes with errors.</td>
</tr>
<tr>
<td>Environment/Build/Find Error After Build</td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Global Macros</strong></td>
<td>Build macros that are shared across all solutions and projects e.g. paths to library files.</td>
</tr>
<tr>
<td>Environment/Macros/Global Macros</td>
<td>StringList</td>
</tr>
<tr>
<td><strong>Keep Going On Error</strong></td>
<td>Build doesn't stop on error.</td>
</tr>
<tr>
<td>Environment/Build/Keep Going On Error</td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Save Project File Before Building</strong></td>
<td>Selects whether to save the project file prior to build.</td>
</tr>
<tr>
<td>Environment/Build/Save Project File On Build</td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Show Build Information</strong></td>
<td>Show build information.</td>
</tr>
<tr>
<td>Environment/Build/Show Build Information</td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Toolchain Root Directory</strong></td>
<td>Specifies where to find the toolchain (compilers etc).</td>
</tr>
<tr>
<td>Environment/Build/Tool Chain Root Directory</td>
<td>String</td>
</tr>
</tbody>
</table>
## Build Acceleration

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable Unity Build</td>
<td>Ignore Unity Build project properties and always build individual project components.</td>
</tr>
<tr>
<td>Environment/Build/Disable Unity BuildBoolean</td>
<td></td>
</tr>
<tr>
<td>Parallel Building Threads</td>
<td>The number of threads to launch when building.</td>
</tr>
<tr>
<td>Environment/Build/Building ThreadsIntegerRange</td>
<td></td>
</tr>
<tr>
<td>Parallel Project Building</td>
<td>Selects whether to build projects or files within projects in parallel.</td>
</tr>
<tr>
<td>Environment/Build/Parallel Project BuildingBoolean</td>
<td></td>
</tr>
</tbody>
</table>

## Compatibility

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Assembler Variant</td>
<td>Specifies the default assembler variant to use.</td>
</tr>
<tr>
<td>ARM/Build/Assembler Variant</td>
<td></td>
</tr>
<tr>
<td>Default Enumeration</td>
<td></td>
</tr>
<tr>
<td>Default Compiler Variant</td>
<td>Specifies the default compiler variant to use.</td>
</tr>
<tr>
<td>ARM/Build/Compiler Variant</td>
<td></td>
</tr>
<tr>
<td>Default Enumeration</td>
<td></td>
</tr>
<tr>
<td>Default Use Compiler Driver</td>
<td>Specifies the default for the Use Compiler Driver project property.</td>
</tr>
<tr>
<td>ARM/Build/Use Compiler Driver Boolean</td>
<td></td>
</tr>
<tr>
<td>Installation Directory</td>
<td>The installation directory to be used for building - the value $(StudioDir) is set to.</td>
</tr>
<tr>
<td>ARM/Build/StudioDir DirectoryDirPath</td>
<td></td>
</tr>
</tbody>
</table>

## Window

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Build Log On Build</td>
<td>Show the build log when a build starts.</td>
</tr>
<tr>
<td>Environment/Show Transcript On BuildBoolean</td>
<td></td>
</tr>
</tbody>
</table>
## Debugging Environment Options

### Breakpoint

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Disassembly Breakpoints On Debug Stop</td>
<td>Clear Disassembly Breakpoints On Debug Stop.</td>
</tr>
<tr>
<td>Environment/Debugger/Clear Disassembly Breakpoint Boolean</td>
<td></td>
</tr>
</tbody>
</table>

### Display

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close Disassembly On Mode Switch</td>
<td>Close Disassembly On Mode Switch.</td>
</tr>
<tr>
<td>Environment/Debugger/Close Disassembly On Mode Switch Boolean</td>
<td></td>
</tr>
<tr>
<td>Data Tips Display a Maximum Of</td>
<td>Selects the maximum number of array elements displayed in a data tip.</td>
</tr>
<tr>
<td>Environment/Debugger/Maximum Array Elements Displayed</td>
<td></td>
</tr>
<tr>
<td>IntegerRange</td>
<td></td>
</tr>
<tr>
<td>Default Display Mode</td>
<td>Selects the format that data values are shown in.</td>
</tr>
<tr>
<td>Environment/Debugger/Default Variable Display Mode Enumeration</td>
<td></td>
</tr>
<tr>
<td>Display Floating Point Number In</td>
<td>The printf format directive used to display floating point numbers.</td>
</tr>
<tr>
<td>Environment/Debugger/Floating Point Format Display Custom</td>
<td></td>
</tr>
<tr>
<td>Maximum Backtrace Calls</td>
<td>Selects the maximum number of calls when backtracing.</td>
</tr>
<tr>
<td>Environment/Debugger/Maximum Backtrace Calls IntegerRange</td>
<td></td>
</tr>
<tr>
<td>Prompt To Display If More Than</td>
<td>The array size to display with prompt.</td>
</tr>
<tr>
<td>Environment/Debugger/Array Elements Prompt Size IntegerRange</td>
<td></td>
</tr>
<tr>
<td>Show Data Tips In Text Editor</td>
<td>Show Data Tips In Text Editor.</td>
</tr>
<tr>
<td>Environment/Debugger/Show Data Tips Boolean</td>
<td></td>
</tr>
<tr>
<td>Show Labels In Disassembly</td>
<td>Show Labels In Disassembly.</td>
</tr>
<tr>
<td>Environment/Debugger/Disassembly Show Labels Boolean</td>
<td></td>
</tr>
<tr>
<td>Show Source In Disassembly</td>
<td>Show Source In Disassembly.</td>
</tr>
<tr>
<td>Environment/Debugger/Disassembly Show Source Boolean</td>
<td></td>
</tr>
<tr>
<td>Show char * as null terminated string</td>
<td>Show char * as null terminated string.</td>
</tr>
<tr>
<td>Environment/Debugger/Display Char Ptr As String Boolean</td>
<td></td>
</tr>
</tbody>
</table>
Source Path
Environment/Debugger/Source Path StringList

Global search path to find source files.

Extended Data Tips

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>Display ASCII extended data tips.</td>
</tr>
<tr>
<td>Binary</td>
<td>Display Binary extended data tips.</td>
</tr>
<tr>
<td>Decimal</td>
<td>Display Decimal extended data tips.</td>
</tr>
<tr>
<td>Hexadecimal</td>
<td>Display Hexadecimal extended data tips.</td>
</tr>
<tr>
<td>Octal</td>
<td>Display Octal extended data tips.</td>
</tr>
<tr>
<td>Unsigned Decimal</td>
<td>Display Unsigned Decimal extended data tips.</td>
</tr>
</tbody>
</table>

Ozone

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ozone Executable</td>
<td>The path to the Ozone executable.</td>
</tr>
</tbody>
</table>

Window

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Debug Terminal On Run</td>
<td>Clear the debug terminal automatically when a program is run.</td>
</tr>
<tr>
<td>Hide Output Window On Successful Load</td>
<td>Hide the Output window when a load completes without error.</td>
</tr>
<tr>
<td>Show Target Log On Load</td>
<td>Debugging/Show Transcript On Load</td>
</tr>
</tbody>
</table>
# IDE Environment Options

## Browser

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Text Size</strong></td>
<td>Sets the text size of the integrated HTML and help browser.</td>
</tr>
<tr>
<td>Environment/Browser/Text Size Enumeration</td>
<td></td>
</tr>
<tr>
<td><strong>Underline Hyperlinks In Browser</strong></td>
<td>Enables underlining of hypertext links in the integrated HTML and help browser.</td>
</tr>
<tr>
<td>Environment/Browser/Underline Web Links Boolean</td>
<td></td>
</tr>
</tbody>
</table>

## File Extension

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ELF Archive File Extensions</strong></td>
<td>The file extensions used for ELF archive files.</td>
</tr>
<tr>
<td>ElfDwarf/Environment/Archive File Extensions StringList</td>
<td></td>
</tr>
<tr>
<td><strong>ELF Executable File Extensions</strong></td>
<td>The file extensions used for ELF executable files.</td>
</tr>
<tr>
<td>ElfDwarf/Environment/Executable File Extensions StringList</td>
<td></td>
</tr>
<tr>
<td><strong>ELF Object File Extensions</strong></td>
<td>The file extensions used for ELF object files.</td>
</tr>
<tr>
<td>ElfDwarf/Environment/Object File Extensions StringList</td>
<td></td>
</tr>
<tr>
<td><strong>Show ELF Header</strong></td>
<td>Display ELF Headers when executable and object files are displayed in text editor.</td>
</tr>
<tr>
<td>ElfDwarf/Environment/Show ELF Header Boolean</td>
<td></td>
</tr>
</tbody>
</table>

## File Search

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Collapse Search Results</strong></td>
<td>Whether to initially collapse search results.</td>
</tr>
<tr>
<td>Find In Files/Collapse Results Boolean</td>
<td></td>
</tr>
<tr>
<td><strong>Files To Search</strong></td>
<td>The wildcard used to match files in Find In Files searches.</td>
</tr>
<tr>
<td>Find In Files/File Type StringList</td>
<td></td>
</tr>
<tr>
<td><strong>Find History</strong></td>
<td>The list of strings recently used in searches.</td>
</tr>
<tr>
<td>Find In Files/Find History StringList</td>
<td></td>
</tr>
<tr>
<td><strong>Flat Search Result Output</strong></td>
<td>Whether to display file search results as a flat list.</td>
</tr>
<tr>
<td>Find In Files/Flat Output Boolean</td>
<td></td>
</tr>
</tbody>
</table>
### Embedded Studio for ARM Reference Manual

### Appendices

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Folder History</strong></td>
<td>The set of folders recently used in file searches.</td>
</tr>
<tr>
<td>Find In Files/Folder History StringList</td>
<td></td>
</tr>
<tr>
<td><strong>Match Case</strong></td>
<td>Whether the case of letters must match exactly when searching.</td>
</tr>
<tr>
<td>Find In Files/Match Case Boolean</td>
<td></td>
</tr>
<tr>
<td><strong>Match Whole Word</strong></td>
<td>Whether the whole word must match when searching.</td>
</tr>
<tr>
<td>Find In Files/Match Whole Word Boolean</td>
<td></td>
</tr>
<tr>
<td><strong>Replace History</strong></td>
<td>The list of strings recently used in searches.</td>
</tr>
<tr>
<td>Find In Files/Replace History StringList</td>
<td></td>
</tr>
<tr>
<td><strong>Search Dependencies</strong></td>
<td>Controls searching of dependent files.</td>
</tr>
<tr>
<td>Find In Files/Search Dependencies Boolean</td>
<td></td>
</tr>
<tr>
<td><strong>Search In</strong></td>
<td>Where to look to find files.</td>
</tr>
<tr>
<td>Find In Files/Context Enumeration</td>
<td></td>
</tr>
<tr>
<td><strong>Use Regular Expressions</strong></td>
<td>Whether to use a regular expression or plain text search.</td>
</tr>
<tr>
<td>Find In Files/Use RegExp Boolean</td>
<td></td>
</tr>
</tbody>
</table>

### Find And Replace

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Greedy Regular Expressions</strong></td>
<td>Enables greedy matching when using regular expressions.</td>
</tr>
<tr>
<td>Find/Greedy RegExp Boolean</td>
<td></td>
</tr>
</tbody>
</table>

### Internet

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Automatically Check For Packages</strong></td>
<td>Specifies whether to enable downloading of the list of available packages.</td>
</tr>
<tr>
<td>Environment/Internet/Check Packages Boolean</td>
<td></td>
</tr>
<tr>
<td><strong>Automatically Check For Updates</strong></td>
<td>Specifies whether to check for software updates.</td>
</tr>
<tr>
<td>Environment/Internet/Check Updates Boolean</td>
<td></td>
</tr>
<tr>
<td><strong>Check For Latest News</strong></td>
<td>Specifies whether to update the latest news window.</td>
</tr>
<tr>
<td>Environment/Internet/RSS Update Boolean</td>
<td></td>
</tr>
<tr>
<td><strong>Enable Connection Debugging</strong></td>
<td>Controls debugging traces of internet connections and downloads.</td>
</tr>
<tr>
<td>Environment/Internet/Enable Debugging Boolean</td>
<td></td>
</tr>
<tr>
<td><strong>External Web Browser</strong></td>
<td>The path to the external web browser to use when accessing non-local files.</td>
</tr>
<tr>
<td>Environment/External Web Browser FileName</td>
<td></td>
</tr>
<tr>
<td><strong>HTTP Caching</strong></td>
<td>Specifies if caching should be permitted when carrying out HTTP requests.</td>
</tr>
<tr>
<td>Environment/Internet/HTTP Caching Boolean</td>
<td></td>
</tr>
<tr>
<td><strong>HTTP Proxy Host</strong></td>
<td>Specifies the IP address or hostname of the HTTP proxy server. If empty, no HTTP proxy server will be used.</td>
</tr>
<tr>
<td>Environment/Internet/HTTP Proxy Server String</td>
<td></td>
</tr>
</tbody>
</table>
### HTTP Proxy Port
**Environment/Internet/HTTP Proxy Port**
Specifies the HTTP proxy server’s port number.

### Maximum Download History Items
**Environment/Internet/Max Download History Items**
The maximum amount of download history kept in the downloads window.

### Use Content Delivery Network
**Environment/Package/Use Content Delivery Network**
Specifies whether to use content delivery network to deliver packages.

---

### Launcher

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Confirm Check Solution Target</strong></td>
<td>Specifies whether the SEGGER Embedded Studio launcher should present a warning if the solution being launched targets a device it does not support.</td>
</tr>
<tr>
<td><strong>Launch Latest Installations Only</strong></td>
<td>Specifies whether the SEGGER Embedded Studio launcher should only consider the latest installations when deciding which one to use.</td>
</tr>
<tr>
<td><strong>Launcher Enabled</strong></td>
<td>Specifies whether the SEGGER Embedded Studio launcher should be used when the operating system or an external application requests a file to be opened.</td>
</tr>
</tbody>
</table>

### Licensing

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Check J-Link For Licenses</strong></td>
<td>Specifies whether to check J-Link’s for licenses.</td>
</tr>
</tbody>
</table>

### Package Manager

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Check Solution Package Dependencies</strong></td>
<td>Specifies whether to check package dependencies when a solution is loaded.</td>
</tr>
<tr>
<td><strong>Package Directory</strong></td>
<td>Specifies the directory packages are installed to.</td>
</tr>
<tr>
<td><strong>Show Check For Packages Dialog</strong></td>
<td>Specifies whether the package manager should prompt for a package list refresh.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Show Logos</td>
<td>Specifies whether the package manager should display company logos.</td>
</tr>
<tr>
<td>Verify Package Downloads</td>
<td>Specifies whether to carry out an MD5 sum check on downloaded package files.</td>
</tr>
</tbody>
</table>

## Print

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bottom Margin</td>
<td>The page's bottom margin in millimetres.</td>
</tr>
<tr>
<td>Left Margin</td>
<td>The page's left margin in millimetres.</td>
</tr>
<tr>
<td>Page Orientation</td>
<td>The page's orientation.</td>
</tr>
<tr>
<td>Page Size</td>
<td>The page's size.</td>
</tr>
<tr>
<td>Right Margin</td>
<td>The page's right margin in millimetres.</td>
</tr>
<tr>
<td>Top Margin</td>
<td>The page's top margin in millimetres.</td>
</tr>
</tbody>
</table>

## Startup

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow Multiple SEGGER Embedded Studios</td>
<td>Allow more than one SEGGER Embedded Studio to run at the same time.</td>
</tr>
<tr>
<td>Load Last Project On Startup</td>
<td>Specifies whether to load the last project the next time SEGGER Embedded Studio runs.</td>
</tr>
<tr>
<td>New Project Directory</td>
<td>The directory where projects are created.</td>
</tr>
<tr>
<td>Show Support Expired Dialog</td>
<td>Specifies whether to show the support expired dialog.</td>
</tr>
<tr>
<td>Splash Screen</td>
<td>How to display the splash screen on startup.</td>
</tr>
</tbody>
</table>
## Status Bar

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Visible) Environment/Status Bar Boolean</td>
<td>Show or hide the status bar.</td>
</tr>
<tr>
<td>Show Build Status Pane Environment/General/Status Bar/Show Build Status Boolean</td>
<td>Show or hide the Build pane in the status bar.</td>
</tr>
<tr>
<td>Show Caret Position Pane Environment/General/Status Bar/Show Caret Pos Boolean</td>
<td>Show or hide the Caret Position pane in the status bar.</td>
</tr>
<tr>
<td>Show Insert/Overwrite Status Pane Environment/General/Status Bar/Show Insert Mode Boolean</td>
<td>Show or hide the Insert/Overwrite pane in the status bar.</td>
</tr>
<tr>
<td>Show Read-Only Status Pane Environment/General/Status Bar/Show Read Only Boolean</td>
<td>Show or hide the Read Only pane in the status bar.</td>
</tr>
<tr>
<td>Show Size Grip Environment/General/Status Bar/Show Size Grip Boolean</td>
<td>Show or hide the status bar size grip.</td>
</tr>
<tr>
<td>Show Target Pane Environment/General/Status Bar/Show Target Boolean</td>
<td>Show or hide the Target pane in the status bar.</td>
</tr>
<tr>
<td>Show Time Pane Environment/General/Status Bar/Show Time Boolean</td>
<td>Show or hide the Time pane in the status bar.</td>
</tr>
</tbody>
</table>

## Title Bar

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Full Solution Path Environment/General/Title Bar/Show Full Solution Path Boolean</td>
<td>Show the full solution path in title bar.</td>
</tr>
</tbody>
</table>

## User Interface

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Main Font Environment/Application Main Font Font</td>
<td>The font to use for the user interface as a whole.</td>
</tr>
<tr>
<td>Environment/Setting</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Application Monospace Font</strong>&lt;br&gt;FontFixedPitchFont</td>
<td>The fixed-size font to use for the user interface as a whole.</td>
</tr>
<tr>
<td><strong>Error Display Timeout</strong>&lt;br&gt;Password &amp; Error Display Timeout IntelRange</td>
<td>The minimum time, in seconds, that errors are shown for in the status bar.</td>
</tr>
<tr>
<td><strong>Errors Are Displayed</strong>&lt;br&gt;Error Display Mode Enumeration</td>
<td>How errors are reported in SEGGER Embedded Studio.</td>
</tr>
<tr>
<td><strong>File Size Display Units</strong>&lt;br&gt;Size Display Unit Enumeration</td>
<td>How to display sizes of items in the user interface. SI defines 1kB=1000 bytes, IEC defines 1kib=1024 bytes, Alternate SI defines 1kB=1024 bytes.</td>
</tr>
<tr>
<td><strong>Number File Names in Menus</strong>&lt;br&gt;Number Menus Boolean</td>
<td>Number the first nine file names in menus for quick keyboard access.</td>
</tr>
<tr>
<td><strong>Qt Style Sheet</strong>&lt;br&gt;Qt Style Sheet FileName</td>
<td>The Qt style sheet to use in order to customize the user interface (experimental).</td>
</tr>
<tr>
<td><strong>Show Large Icons In Toolbars</strong>&lt;br&gt;Large Icons Boolean</td>
<td>Show large or small icons on toolbars.</td>
</tr>
<tr>
<td><strong>Show Ribbon</strong>&lt;br&gt;Ribbon/Show Boolean</td>
<td>Show or hide the ribbon.</td>
</tr>
<tr>
<td><strong>Show Window Selector On Ctrl+Tab</strong>&lt;br&gt;Show Selector Boolean</td>
<td>Present the Window Selector on Next Window and Previous Window commands activated from the keyboard.</td>
</tr>
<tr>
<td><strong>Theme</strong>&lt;br&gt;Studio Theme Enumeration</td>
<td>The user interface style and color theme to use.</td>
</tr>
<tr>
<td><strong>Window Menu Contains At Most</strong>&lt;br&gt;Max Window Menu Items IntegerRange</td>
<td>The maximum number of windows appearing in the Windows menu.</td>
</tr>
</tbody>
</table>
# Programming Language Environment Options

## Assembly Language

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column Guide Columns</td>
<td>The columns that guides are drawn for.</td>
</tr>
<tr>
<td>Text Editor/Indent/Assembly Language/Column Guides</td>
<td></td>
</tr>
<tr>
<td>Indent Closing Brace</td>
<td>Indent the closing brace of compound statements.</td>
</tr>
<tr>
<td>Text Editor/Indent/Assembly Language/Close Brace</td>
<td></td>
</tr>
<tr>
<td>Indent Context</td>
<td>The number of lines to use for context when indenting.</td>
</tr>
<tr>
<td>Text Editor/Indent/Assembly Language/Context Lines</td>
<td></td>
</tr>
<tr>
<td>Indent Mode</td>
<td>How to indent when a new line is inserted.</td>
</tr>
<tr>
<td>Text Editor/Indent/Assembly Language/Indent Mode</td>
<td></td>
</tr>
<tr>
<td>Indent Opening Brace</td>
<td>Indent the opening brace of compound statements.</td>
</tr>
<tr>
<td>Text Editor/Indent/Assembly Language/Open Brace</td>
<td></td>
</tr>
<tr>
<td>Indent Size</td>
<td>The number of columns to indent a code block.</td>
</tr>
<tr>
<td>Text Editor/Indent/Assembly Language/Size</td>
<td></td>
</tr>
<tr>
<td>Tab Size</td>
<td>The number of columns between tabstops.</td>
</tr>
<tr>
<td>Text Editor/Indent/Assembly Language/Tab Size</td>
<td></td>
</tr>
<tr>
<td>Use Tabs</td>
<td>Insert tabs when indenting.</td>
</tr>
<tr>
<td>Text Editor/Indent/Assembly Language/Use Tabs</td>
<td></td>
</tr>
<tr>
<td>User-Defined Keywords</td>
<td>Additional identifiers to highlight as keywords.</td>
</tr>
<tr>
<td>Text Editor/Indent/Assembly Language/Keywords</td>
<td></td>
</tr>
</tbody>
</table>

## C and C++

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column Guide Columns</td>
<td>The columns that guides are drawn for.</td>
</tr>
<tr>
<td>Text Editor/Indent/C and C++/Column Guides</td>
<td></td>
</tr>
</tbody>
</table>
### Indent Closing Brace

**Text Editor/Indent/C and C++/Close Brace**

Indent the closing brace of compound statements.

### Indent Context

**Text Editor/Indent/C and C++/Context Lines**

The number of lines to use for context when indenting.

### Indent Mode

**Text Editor/Indent/C and C++/Indent Mode**

How to indent when a new line is inserted.

### Indent Opening Brace

**Text Editor/Indent/C and C++/Open Brace**

Indent the opening brace of compound statements.

### Indent Size

**Text Editor/Indent/C and C++/Size**

The number of columns to indent a code block.

### Tab Size

**Text Editor/Indent/C and C++/Tab Size**

The number of columns between tabstops.

### Use Tabs

**Text Editor/Indent/C and C++/Use Tabs**

Insert tabs when indenting.

### User-Defined Keywords

**Text Editor/Indent/C and C++/Keywords**

Additional identifiers to highlight as keywords.

### Default

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Column Guide Columns</strong></td>
<td>The columns that guides are drawn for.</td>
</tr>
<tr>
<td><strong>Text Editor/Indent/Default/Column Guides</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Indent Closing Brace</strong></td>
<td>Indent the closing brace of compound statements.</td>
</tr>
<tr>
<td><strong>Text Editor/Indent/Default/Close Brace</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Indent Context</strong></td>
<td>The number of lines to use for context when indenting.</td>
</tr>
<tr>
<td><strong>Text Editor/Indent/Default/Context Lines</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Indent Mode</strong></td>
<td>How to indent when a new line is inserted.</td>
</tr>
<tr>
<td><strong>Text Editor/Indent/Default/Indent Mode</strong></td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td><strong>Indent Opening Brace</strong></td>
<td>Indent the opening brace of compound statements.</td>
</tr>
<tr>
<td>Text Editor/Indent/Default/Open</td>
<td></td>
</tr>
<tr>
<td><strong>Indent Size</strong></td>
<td>The number of columns to indent a code block.</td>
</tr>
<tr>
<td>Text Editor/Indent/Default/SizeIntegerRange</td>
<td></td>
</tr>
<tr>
<td><strong>Tab Size</strong></td>
<td>The number of columns between tabstops.</td>
</tr>
<tr>
<td>Text Editor/Indent/Default/Tab SizeIntegerRange</td>
<td></td>
</tr>
<tr>
<td><strong>Use Tabs</strong></td>
<td>Insert tabs when indenting.</td>
</tr>
<tr>
<td>Text Editor/Indent/Default/Use TabsBoolean</td>
<td></td>
</tr>
<tr>
<td><strong>User-Defined Keywords</strong></td>
<td>Additional identifiers to highlight as keywords.</td>
</tr>
<tr>
<td>Text Editor/Indent/Default/KeywordsStringList</td>
<td></td>
</tr>
</tbody>
</table>

**Java**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Column Guide Columns</strong></td>
<td>The columns that guides are drawn for.</td>
</tr>
<tr>
<td>Text Editor/Indent/Java/Column GuidesString</td>
<td></td>
</tr>
<tr>
<td><strong>Indent Closing Brace</strong></td>
<td>Indent the closing brace of compound statements.</td>
</tr>
<tr>
<td>Text Editor/Indent/Java/Close BraceBoolean</td>
<td></td>
</tr>
<tr>
<td><strong>Indent Context</strong></td>
<td>The number of lines to use for context when indenting.</td>
</tr>
<tr>
<td>Text Editor/Indent/Java/Context LinesIntegerRange</td>
<td></td>
</tr>
<tr>
<td><strong>Indent Mode</strong></td>
<td>How to indent when a new line is inserted.</td>
</tr>
<tr>
<td>Text Editor/Indent/Java/Indent ModeEnumeration</td>
<td></td>
</tr>
<tr>
<td><strong>Indent Opening Brace</strong></td>
<td>Indent the opening brace of compound statements.</td>
</tr>
<tr>
<td>Text Editor/Indent/Java/Open BraceBoolean</td>
<td></td>
</tr>
<tr>
<td><strong>Indent Size</strong></td>
<td>The number of columns to indent a code block.</td>
</tr>
<tr>
<td>Text Editor/Indent/Java/SizeIntegerRange</td>
<td></td>
</tr>
<tr>
<td><strong>Tab Size</strong></td>
<td>The number of columns between tabstops.</td>
</tr>
<tr>
<td>Text Editor/Indent/Java/Tab SizeIntegerRange</td>
<td></td>
</tr>
<tr>
<td><strong>Use Tabs</strong></td>
<td>Insert tabs when indenting.</td>
</tr>
<tr>
<td>Text Editor/Indent/Java/Use TabsBoolean</td>
<td></td>
</tr>
<tr>
<td><strong>User-Defined Keywords</strong></td>
<td>Additional identifiers to highlight as keywords.</td>
</tr>
<tr>
<td>Text Editor/Indent/Java/KeywordsStringList</td>
<td></td>
</tr>
</tbody>
</table>
Source Control Environment Options

External Tools

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diff Command Line</td>
<td>The diff command line.</td>
</tr>
<tr>
<td>Environment/Source Code Control/DiffCommand</td>
<td></td>
</tr>
<tr>
<td>Merge Command Line</td>
<td>The merge command line.</td>
</tr>
<tr>
<td>Environment/Source Code Control/MergeCommand</td>
<td></td>
</tr>
</tbody>
</table>

Preference

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Immediately</td>
<td>Bypasses the confirmation dialog and immediately adds items to source control.</td>
</tr>
<tr>
<td>Environment/Source Code Control/Immediate AddBoolean</td>
<td></td>
</tr>
<tr>
<td>Commit Immediately</td>
<td>Bypasses the confirmation dialog and immediately commits items.</td>
</tr>
<tr>
<td>Environment/Source Code Control/Immediate CommitBoolean</td>
<td></td>
</tr>
<tr>
<td>Get Immediately</td>
<td>Bypasses the confirmation dialog and immediately gets items from source control.</td>
</tr>
<tr>
<td>Environment/Source Code Control/Immediate GetBoolean</td>
<td></td>
</tr>
<tr>
<td>Lock Immediately</td>
<td>Bypasses the confirmation dialog and immediately locks items.</td>
</tr>
<tr>
<td>Environment/Source Code Control/Immediate LockBoolean</td>
<td></td>
</tr>
<tr>
<td>Remove Immediately</td>
<td>Bypasses the confirmation dialog and immediately removes items from source control.</td>
</tr>
<tr>
<td>Environment/Source Code Control/Immediate RemoveBoolean</td>
<td></td>
</tr>
<tr>
<td>Resolved Immediately</td>
<td>Bypasses the confirmation dialog and immediately mark items resolved.</td>
</tr>
<tr>
<td>Environment/Source Code Control/Immediate ResolvedBoolean</td>
<td></td>
</tr>
<tr>
<td>Revert Immediately</td>
<td>Bypasses the confirmation dialog and immediately revert items.</td>
</tr>
<tr>
<td>Environment/Source Code Control/Immediate RevertBoolean</td>
<td></td>
</tr>
<tr>
<td>Unlock Immediately</td>
<td>Bypasses the confirmation dialog and immediately unlocks items.</td>
</tr>
<tr>
<td>Environment/Source Code Control/Immediate UnlockBoolean</td>
<td></td>
</tr>
<tr>
<td>Update Immediately Environment/Source Code Control/Immediate Update Boolean</td>
<td>Bypasses the confirmation dialog and immediately updates items.</td>
</tr>
</tbody>
</table>
Text Editor Environment Options

Auto Recovery

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Recovery Backup Time</td>
<td>The time in minutes between saving of auto recovery backups files or 0 to disable generation of backup files.</td>
</tr>
<tr>
<td>Auto Recovery Keep Time</td>
<td>The time in days to keep unrecovered backup files or 0 to disable deletion of unrecovered backup files.</td>
</tr>
</tbody>
</table>

Cursor Fence

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bottom Margin</td>
<td>The number of lines in the bottom margin.</td>
</tr>
<tr>
<td>Keep Cursor Within Fence</td>
<td>Enable margins to fence and scroll around the cursor.</td>
</tr>
<tr>
<td>Left Margin</td>
<td>The number of characters in the left margin.</td>
</tr>
<tr>
<td>Right Margin</td>
<td>The number of characters in the right margin.</td>
</tr>
<tr>
<td>Top Margin</td>
<td>The number of lines in the right margin.</td>
</tr>
</tbody>
</table>

Editing

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow Drag and Drop Editing</td>
<td>Enables dragging and dropping of selections in the text editor.</td>
</tr>
<tr>
<td>Bold Popup Diagnostic Messages</td>
<td>Displays popup diagnostic messages in bold for easier reading.</td>
</tr>
<tr>
<td>Column-mode Tab</td>
<td>Tab key moves to the next textual column using the line above.</td>
</tr>
<tr>
<td>Confirm Modified File Reload</td>
<td>Display a confirmation prompt before reloading a file that has been modified on disk.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Copy Action When Nothing Selected</td>
<td>What Copy copies when nothing is selected.</td>
</tr>
<tr>
<td>Cut Action When Nothing Selected</td>
<td>What Cut cuts when nothing is selected.</td>
</tr>
<tr>
<td>Cut Single Blank Line</td>
<td>Selects whether to place text on the clipboard when a single blank line is cut. When set to Yes, cutting a single blank line will put the blank line on the clipboard. When set to No, cutting a single blank line deletes the line and does not place it on the clipboard.</td>
</tr>
<tr>
<td>Diagnostic Cycle Mode</td>
<td>Iterates through diagnostics either from most severe to least severe or in reported order.</td>
</tr>
<tr>
<td>Edit Read-Only Files</td>
<td>Allow editing of read-only files.</td>
</tr>
<tr>
<td>Enable Virtual Space</td>
<td>Permit the cursor to move into locations that do not currently contain text.</td>
</tr>
<tr>
<td>Numeric Keypad Editing</td>
<td>Selects whether the numeric keypad plus and minus buttons copy and cut text.</td>
</tr>
<tr>
<td>Tab Key Indents Preprocessor Directives</td>
<td>Enables or disables the indentation of C preprocessor directives when using tab key indentation on a selection.</td>
</tr>
<tr>
<td>Undo And Redo Behavior</td>
<td>How Undo and Redo group your typing when it is undone and redone.</td>
</tr>
</tbody>
</table>

## Find And Replace

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case Sensitive Matching</td>
<td>Enables or disables the case sensitivity of letters when searching.</td>
</tr>
<tr>
<td>Find History</td>
<td>The list of strings recently used in searches.</td>
</tr>
<tr>
<td>Regular Expression Matching</td>
<td>Enables regular expression matching rather than plain text matching.</td>
</tr>
<tr>
<td>Replace History</td>
<td>The list of strings recently used in replaces.</td>
</tr>
<tr>
<td>Whole Word Matching</td>
<td>Enables or disables whole word matching when searching.</td>
</tr>
</tbody>
</table>

## Formatting
<table>
<thead>
<tr>
<th>Access Modifier Offset</th>
<th>Text Editor/Formatting/AccessModifierOffsetInteger</th>
<th>The extra indent or outdent of access modifiers, e.g. public:.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional Formatting Styles</td>
<td>Text Editor/Additional Formatting StylesStringList</td>
<td>Additional styles to pass to clang-format.</td>
</tr>
<tr>
<td>Align After Open Bracket</td>
<td>Text Editor/Formatting/AlignAfterOpenBracketBoolean</td>
<td>If enabled, horizontally aligns arguments after an open bracket.</td>
</tr>
<tr>
<td>Align Consecutive Assignments</td>
<td>Text Editor/Formatting/AlignConsecutiveAssignmentsBoolean</td>
<td>If enabled, aligns consecutive assignments.</td>
</tr>
<tr>
<td>Align Consecutive Declarations</td>
<td>Text Editor/Formatting/AlignConsecutiveDeclarationsBoolean</td>
<td>If enabled, aligns consecutive declarations.</td>
</tr>
<tr>
<td>Align Escaped Newlines Left</td>
<td>Text Editor/Formatting/AlignEscapedNewlinesLeftBoolean</td>
<td>If enabled, aligns escaped newlines as far left as possible otherwise puts them into the right-most column.</td>
</tr>
<tr>
<td>Align Operands</td>
<td>Text Editor/Formatting/AlignOperandsBoolean</td>
<td>If enabled, horizontally align operands of binary and ternary expressions.</td>
</tr>
<tr>
<td>Align Trailing Comments</td>
<td>Text Editor/Formatting/AlignTrailingCommentsBoolean</td>
<td>If enabled, aligns trailing comments.</td>
</tr>
<tr>
<td>Allow All Parameters Of Declaration On Next Line</td>
<td>Text Editor/Formatting/AllowAllParametersOfDeclarationOnNextLineBoolean</td>
<td>Allow putting all parameters of a function declaration onto the next line even if Bin-pack Parameters is disabled.</td>
</tr>
<tr>
<td>Allow Short 'if' Statements On A Single Line</td>
<td>Text Editor/Formatting/AllowShortIfStatementsOnASingleLineBoolean</td>
<td>If enabled, short 'if' statements are put on a single line.</td>
</tr>
<tr>
<td>Allow Short Blocks On A Single Line</td>
<td>Text Editor/Formatting/AllowShortBlocksOnASingleLineBoolean</td>
<td>If enabled, allows contracting simple braced statements to a single line.</td>
</tr>
<tr>
<td>Allow Short Case Labels On A Single Line</td>
<td>Text Editor/Formatting/AllowShortCaseLabelsOnASingleLineBoolean</td>
<td>If enabled, short case labels will be contracted to a single line.</td>
</tr>
<tr>
<td>Allow Short Functions On A Single Line</td>
<td>Text Editor/Formatting/AllowShortFunctionsOnASingleLineEnumeration</td>
<td>Optionally compress small functions to a single line.</td>
</tr>
<tr>
<td>Allow Short Loop Statements On A Single Line</td>
<td>Text Editor/Formatting/AllowShortLoopsOnASingleLineBoolean</td>
<td>If enabled, short loop statements are put on a single line.</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td>Setting</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Always Break After Return Type</strong></td>
<td>The function declaration return type breaking style to use.</td>
<td>Text Editor/Formatting/AlwaysBreakAfterReturnTypeEnumeration</td>
</tr>
<tr>
<td><strong>Always Break Before Multiline Strings</strong></td>
<td>If enabled, always break before multiline strings.</td>
<td>Text Editor/Formatting/AlwaysBreakBeforeMultilineStringsBoolean</td>
</tr>
<tr>
<td><strong>Always Break Template Declarations</strong></td>
<td>If enabled, always break after the 'template&lt;...&gt;' of a template declaration.</td>
<td>Text Editor/Formatting/AlwaysBreakTemplateDeclarationsBoolean</td>
</tr>
<tr>
<td><strong>Bin-Pack Arguments</strong></td>
<td>If disabled, a function calls arguments will either be all on the same line or will have one line each.</td>
<td>Text Editor/Formatting/BinPackArgumentsBoolean</td>
</tr>
<tr>
<td><strong>Bin-Pack Parameters</strong></td>
<td>If disabled, a function call's or function definition's parameters will either all be on the same line or will have one line each.</td>
<td>Text Editor/Formatting/BinPackParametersBoolean</td>
</tr>
<tr>
<td><strong>Break Before Binary Operators</strong></td>
<td>The way to wrap binary operators.</td>
<td>Text Editor/Formatting/BreakBeforeBinaryOperatorsBoolean</td>
</tr>
<tr>
<td><strong>Break Before Braces</strong></td>
<td>The brace breaking style to use.</td>
<td>Text Editor/Formatting/BreakBeforeBracesEnumeration</td>
</tr>
<tr>
<td><strong>Break Before Inheritance Comma</strong></td>
<td>If enabled, the class inheritance expression will break before : and , if there is multiple inheritance.</td>
<td>Text Editor/Formatting/BreakBeforeInheritanceCommaBoolean</td>
</tr>
<tr>
<td><strong>Break Before Ternary Operators</strong></td>
<td>If enabled, ternary operators will be placed after line breaks.</td>
<td>Text Editor/Formatting/BreakBeforeTernaryOperatorsBoolean</td>
</tr>
<tr>
<td><strong>Break Constructor Initializers Before Comma</strong></td>
<td>If enabled, always break constructor initializers before commas and align the commas with the colon.</td>
<td>Text Editor/Formatting/BreakConstructorInitializersBeforeCommaBoolean</td>
</tr>
<tr>
<td><strong>Break String Literals</strong></td>
<td>Allow breaking string literals when formatting.</td>
<td>Text Editor/Formatting/BreakStringLiteralsBoolean</td>
</tr>
<tr>
<td><strong>C++11 Braced List Style</strong></td>
<td>If enabled, format braced lists as best suited for C++11 braced lists.</td>
<td>Text Editor/Formatting/Cpp11BracedListStyleBoolean</td>
</tr>
<tr>
<td><strong>Column Limit</strong></td>
<td>The column limit which limits the width of formatted lines.</td>
<td>Text Editor/Formatting/ColumnLimitInteger</td>
</tr>
<tr>
<td><strong>Comment Pragmas</strong></td>
<td>A regular expression that describes comments with special meaning, which should not be split into lines or otherwise changed.</td>
<td>Text Editor/Formatting/CommentPragmasString</td>
</tr>
<tr>
<td>Feature</td>
<td>Text Editor/Formatting/</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------</td>
<td></td>
</tr>
<tr>
<td>Compact Namespaces</td>
<td>CompactNamespacesBoolean</td>
<td></td>
</tr>
<tr>
<td>Constructor Initializer All On One Line Or One Per Line</td>
<td>ConstructorInitializerAllOnOneLineOrOnePerLine</td>
<td></td>
</tr>
<tr>
<td>Constructor Initializer Indent Width</td>
<td>ConstructorInitializerIndentWidthInteger</td>
<td></td>
</tr>
<tr>
<td>Continuation Indent Width</td>
<td>ContinuationIndentWidthInteger</td>
<td></td>
</tr>
<tr>
<td>Derive Pointer Alignment</td>
<td>DerivePointerAlignmentBoolean</td>
<td></td>
</tr>
<tr>
<td>Empty Lines At End Of File</td>
<td>LinesAtEOFIntegerRange</td>
<td></td>
</tr>
<tr>
<td>Fix Namespace Comments</td>
<td>FixNamespaceCommentsBoolean</td>
<td></td>
</tr>
<tr>
<td>For-Each Macros</td>
<td>ForEachMacrosStringList</td>
<td></td>
</tr>
<tr>
<td>Formatting Indent Width</td>
<td>IndentWidthInteger</td>
<td></td>
</tr>
<tr>
<td>Formatting Style</td>
<td>FormattingStyleEnumeration</td>
<td></td>
</tr>
<tr>
<td>Formatting Tab Width</td>
<td>TabWidthIntegerRange</td>
<td></td>
</tr>
<tr>
<td>Include Is Main Regex</td>
<td>IncludeIsMainRegexString</td>
<td></td>
</tr>
<tr>
<td>Indent Case Labels</td>
<td>IndentCaseLabelsBoolean</td>
<td></td>
</tr>
<tr>
<td>Indent Wrapped Function Names</td>
<td>IndentWrappedFunctionNamesBoolean</td>
<td></td>
</tr>
</tbody>
</table>

- **Compact Namespaces**
  - If enabled, consecutive namespace declarations will be on the same line. If disabled, each namespace is declared on a new line.

- **Constructor Initializer All On One Line Or One Per Line**
  - If enabled and the constructor initializers don’t fit on a line, put each initializer on its own line.

- **Constructor Initializer Indent Width**
  - The number of characters to use for indentation of constructor initializer lists.

- **Continuation Indent Width**
  - Indent width for line continuations.

- **Derive Pointer Alignment**
  - If enabled, analyze the formatted file for the most common alignment of address of and dereference. PointerAlignment is then used only as fallback.

- **Empty Lines At End Of File**
  - The number of lines to add at the end of the file.

- **Fix Namespace Comments**
  - If enabled, add missing namespace end comments and fix invalid existing ones.

- **For-Each Macros**
  - A list of macros that should be interpreted as foreach loops rather than function calls.

- **Formatting Indent Width**
  - The number of columns the code formatter uses for indentation. Note that this is not the indent width used by the text editor, that value is specified in the 'Languages' environment option group.

- **Formatting Style**
  - Select a set of formatting options based on a named standard.

- **Formatting Tab Width**
  - The number of columns the code formatter uses for tab stops. Note that this is not the tab width used by the text editor, that value is specified in the 'Languages' environment option group.

- **Include Is Main Regex**
  - Specify a regular expression of suffixes that are allowed in the file-to-main-include mapping.

- **Indent Case Labels**
  - If enabled, indent case labels one level from the switch statement.

- **Indent Wrapped Function Names**
  - If enabled, Indent if a function definition or declaration is wrapped after the type.
<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keep Empty Lines At The Start Of Blocks</td>
<td>If enabled, empty lines at the start of blocks are kept.</td>
</tr>
<tr>
<td>Macro Block Begin</td>
<td>A regular expression matching macros that start a block.</td>
</tr>
<tr>
<td>Macro Block End</td>
<td>A regular expression matching macros that end a block.</td>
</tr>
<tr>
<td>Maximum Empty Lines To Keep</td>
<td>The maximum number of consecutive empty lines to keep.</td>
</tr>
<tr>
<td>Namespace Indentation</td>
<td>The indentation used for namespaces.</td>
</tr>
<tr>
<td>Penalty Break Assignment</td>
<td>The penalty for breaking around an assignment operator.</td>
</tr>
<tr>
<td>Penalty Break Before First Call Parameter</td>
<td>The penalty for breaking a function call after 'call('.</td>
</tr>
<tr>
<td>Penalty Break Before First Less-Less</td>
<td>The penalty for breaking before the first less-less.</td>
</tr>
<tr>
<td>Penalty Break Comment</td>
<td>The penalty for each line break introduced inside a comment.</td>
</tr>
<tr>
<td>Penalty Break String</td>
<td>The penalty for each line break introduced inside a string literal.</td>
</tr>
<tr>
<td>Penalty Excess Character</td>
<td>The penalty for each character outside of the column limit.</td>
</tr>
<tr>
<td>Penalty Return Type On Its Own Line</td>
<td>Penalty for putting the return type of a function onto its own line.</td>
</tr>
<tr>
<td>Pointer Alignment</td>
<td>Pointer and reference alignment style.</td>
</tr>
<tr>
<td>Reflow Comments</td>
<td>If enabled, clang-format will attempt to re-flow comments.</td>
</tr>
<tr>
<td>Sort Includes</td>
<td>If enabled, sort #includes.</td>
</tr>
<tr>
<td>Property</td>
<td>Category</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>Sort Using Declarations</td>
<td>Text Editor/Formatting/SortUsingDeclarations</td>
</tr>
<tr>
<td>Space After C Style Cast</td>
<td>Text Editor/Formatting/SpaceAfterCStyleCast</td>
</tr>
<tr>
<td>Space After Template Keyword</td>
<td>Text Editor/Formatting/SpaceAfterTemplateKeyword</td>
</tr>
<tr>
<td>Space Before Assignment Operators</td>
<td>Text Editor/Formatting/SpaceBeforeAssignmentOperators</td>
</tr>
<tr>
<td>Space Before Parentheses</td>
<td>Text Editor/Formatting/SpaceBeforeParentheses</td>
</tr>
<tr>
<td>Space In Empty Parentheses</td>
<td>Text Editor/Formatting/SpaceInEmptyParentheses</td>
</tr>
<tr>
<td>Spaces Before Trailing Comments</td>
<td>Text Editor/Formatting/SpacesBeforeTrailingComments</td>
</tr>
<tr>
<td>Spaces In Angles</td>
<td>Text Editor/Formatting/SpacesInAngles</td>
</tr>
<tr>
<td>Spaces In C-style Cast Parentheses</td>
<td>Text Editor/Formatting/SpacesInCStyleCastParentheses</td>
</tr>
<tr>
<td>Spaces In Container Literals</td>
<td>Text Editor/Formatting/SpacesInContainerLiterals</td>
</tr>
<tr>
<td>Spaces In Parentheses</td>
<td>Text Editor/Formatting/SpacesInParentheses</td>
</tr>
<tr>
<td>Spaces In Square Brackets</td>
<td>Text Editor/Formatting/SpacesInSquareBrackets</td>
</tr>
<tr>
<td>Standard</td>
<td>Text Editor/Formatting/StandardEnumeration</td>
</tr>
<tr>
<td>Tab Style</td>
<td>Text Editor/Formatting/UseTabEnumeration</td>
</tr>
</tbody>
</table>
### Use .clang-format File

<table>
<thead>
<tr>
<th>Text Editor/Use .clang-format File</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load code formatting style configuration from a .clang-format file located in one of the parent directories of the source file rather than use the formatting options.</td>
<td></td>
</tr>
</tbody>
</table>

### International

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto-Detect UTF-8</td>
<td>Auto-detect UTF-8 encoding without signature.</td>
</tr>
<tr>
<td>Default Text File Encoding</td>
<td>The encoding to use if not overridden by a project property or file is not in a known format.</td>
</tr>
</tbody>
</table>

### Mouse

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+Left Click Action</td>
<td>The action the editor performs on Alt+Left Click.</td>
</tr>
<tr>
<td>Ctrl+Left Click Action</td>
<td>The action the editor performs on Ctrl+Left Click.</td>
</tr>
<tr>
<td>Ctrl+Middle Click Action</td>
<td>The action the editor performs on Ctrl+Middle Click.</td>
</tr>
<tr>
<td>Ctrl+Right Click Action</td>
<td>The action the editor performs on Ctrl+Right Click.</td>
</tr>
<tr>
<td>Middle Click Action</td>
<td>The action the editor performs on Middle Click.</td>
</tr>
</tbody>
</table>
### Mouse Wheel Adjusts Font Size

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text Editor/Mouse Wheel Adjusts Font Size</td>
<td>Enable or disable resizing of font by mouse wheel when CTRL key pressed.</td>
</tr>
</tbody>
</table>

### Shift+Middle Click Action

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment/Project Explorer/Shift+Middle Click Action</td>
<td>The action the editor performs on Shift+Middle Click.</td>
</tr>
</tbody>
</table>

### Shift+Right Click Action

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment/Project Explorer/Shift+Right Click Action</td>
<td>The action the editor performs on Shift+Right Click.</td>
</tr>
</tbody>
</table>

## Programmer Assistance

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTENTION Tag List</td>
<td>Text Editor/ATTENTION Tags StringList</td>
</tr>
<tr>
<td>Ask For Index</td>
<td>Text Editor/Ask For Index Boolean</td>
</tr>
<tr>
<td>Auto-Comment Text</td>
<td>Text Editor/Auto Comment Boolean</td>
</tr>
<tr>
<td>Auto-Surround Text</td>
<td>Text Editor/Auto Surround Boolean</td>
</tr>
<tr>
<td>Check Spelling</td>
<td>Text Editor/Spell Checking Boolean</td>
</tr>
<tr>
<td>Code Completion Replaces Existing Word</td>
<td>Text Editor/Completion Replaces Existing Word Boolean</td>
</tr>
<tr>
<td>Code Completion Suggestion Selection Key</td>
<td>Text Editor/Suggestion Selection Key Enumeration</td>
</tr>
<tr>
<td>Display Code Completion Suggestions While Typing</td>
<td>Text Editor/Suggest Completion While Typing Boolean</td>
</tr>
<tr>
<td>Enable Popup Diagnostics</td>
<td>Text Editor/Enable Popup Diagnostics Boolean</td>
</tr>
<tr>
<td>FIXME Tag List</td>
<td>Text Editor/FIXME Tags StringList</td>
</tr>
<tr>
<td>Inactive Code Opacity</td>
<td>Text Editor/Inactive Code Opacity IntegerRange</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Include Preprocessor Definitions in Suggestions</strong></td>
<td>Include or exclude preprocessor definitions in code completion suggestions.</td>
</tr>
<tr>
<td>Text Editor/Preprocessor Definition Suggestions Boolean</td>
<td></td>
</tr>
<tr>
<td><strong>Include Templates in Suggestions</strong></td>
<td>Include or exclude templates in code completion suggestions.</td>
</tr>
<tr>
<td>Text Editor/Template Suggestions Boolean</td>
<td></td>
</tr>
<tr>
<td><strong>Lint Tag List</strong></td>
<td>Set the tags to display as Lint directives.</td>
</tr>
<tr>
<td>Text Editor/LINT Tags StringList</td>
<td></td>
</tr>
<tr>
<td><strong>Show Inactive Code</strong></td>
<td>Show code that has been conditionally excluded by the preprocessor.</td>
</tr>
<tr>
<td>Text Editor/Show Inactive Code Boolean</td>
<td></td>
</tr>
<tr>
<td><strong>Show Symbol Declaration Tooltips</strong></td>
<td>Show tooltips when hovering over symbols.</td>
</tr>
<tr>
<td>Text Editor/Show Tooltip Boolean</td>
<td></td>
</tr>
<tr>
<td><strong>Template Characters To Match</strong></td>
<td>The number of characters to match before suggesting a template.</td>
</tr>
<tr>
<td>Text Editor/Template Suggestions Characters IntegerRange</td>
<td></td>
</tr>
</tbody>
</table>

**Save**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Backup File History Depth</strong></td>
<td>The number of backup files to keep when saving an existing file.</td>
</tr>
<tr>
<td>Text Editor/Backup File Depth IntegerRange</td>
<td></td>
</tr>
<tr>
<td><strong>Default Line Endings</strong></td>
<td>The line ending format to use for a new file or a file where the existing line ending format cannot be determined.</td>
</tr>
<tr>
<td>Text Editor/Default Line Endings Enumeration</td>
<td></td>
</tr>
<tr>
<td><strong>Delete Trailing Space On Save</strong></td>
<td>Deletes trailing whitespace from each line when a file is saved.</td>
</tr>
<tr>
<td>Text Editor/Delete Trailing Space On Save Boolean</td>
<td></td>
</tr>
<tr>
<td><strong>Tab Cleanup On Save</strong></td>
<td>Cleans up tabs when a file is saved.</td>
</tr>
<tr>
<td>Text Editor/Cleanup Tabs On Save Enumeration</td>
<td></td>
</tr>
</tbody>
</table>

**Visual Appearance**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Font</strong></td>
<td>The font to use for text editors.</td>
</tr>
<tr>
<td>Text Editor/Font FixedPitch Font</td>
<td></td>
</tr>
<tr>
<td><strong>Font Rendering</strong></td>
<td>The font rendering scheme to use in text editors.</td>
</tr>
<tr>
<td>Text Editor/Font Rendering Enumeration</td>
<td></td>
</tr>
<tr>
<td><strong>Font Smoothing Threshold</strong></td>
<td>The minimum size for font smoothing: font sizes smaller than this will have antialiasing turned off.</td>
</tr>
<tr>
<td>Text Editor/Antialias Threshold IntegerRange</td>
<td></td>
</tr>
<tr>
<td><strong>Hide Cursor When Typing</strong></td>
<td>Hide or show the I-beam cursor when you start to type.</td>
</tr>
<tr>
<td>Text Editor/Hide Cursor When Typing Boolean</td>
<td></td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Highlight All Selected Text</td>
<td>Enable or disable visually highlighting all text that matches the current selection.</td>
</tr>
<tr>
<td>Highlight Cursor Line</td>
<td>Enable or disable visually highlighting the cursor line.</td>
</tr>
<tr>
<td>Horizontal Scroll Bar</td>
<td>Show or hide the horizontal scroll bar.</td>
</tr>
<tr>
<td>Insert Caret Style</td>
<td>How the caret is displayed with the editor in insert mode.</td>
</tr>
<tr>
<td>Line Numbers</td>
<td>How often line numbers are displayed in the margin.</td>
</tr>
<tr>
<td>Mate Match Off Screen</td>
<td>Specifies whether braces, brackets, and parentheses are matched when off screen.</td>
</tr>
<tr>
<td>Mate Matching Mode</td>
<td>Controls when braces, brackets, and parentheses are matched.</td>
</tr>
<tr>
<td>Overwrite Caret Style</td>
<td>How the caret is displayed with the editor in overwrite mode.</td>
</tr>
<tr>
<td>Show Diagnostic Icons In Gutter</td>
<td>Enables display of diagnostic icons in the icon gutter.</td>
</tr>
<tr>
<td>Show Icon Gutter</td>
<td>Show or hide the left-hand gutter containing breakpoint, bookmark, and optional diagnostic icons.</td>
</tr>
<tr>
<td>Show Mini Toolbar</td>
<td>Show the mini toolbar when selecting text with the mouse.</td>
</tr>
<tr>
<td>Show Toolbar</td>
<td>Show or hide the Editor toolbar in the dock window.</td>
</tr>
<tr>
<td>Use I-beam Cursor</td>
<td>Show an I-beam or arrow cursor in the text editor.</td>
</tr>
<tr>
<td>Vertical Scroll Bar</td>
<td>Show or hide the vertical scroll bar.</td>
</tr>
<tr>
<td>View Whitespace</td>
<td>Make whitespace characters visible in the text editor.</td>
</tr>
</tbody>
</table>
## Windows Environment Options

### Autos

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Show Member Functions</strong></td>
<td>Controls whether C++ class member functions are displayed.</td>
</tr>
<tr>
<td>Environment/AutosWindow/Show Member Functions</td>
<td></td>
</tr>
<tr>
<td><strong>Show Variable Address Column</strong></td>
<td>Controls whether the variable address column is displayed.</td>
</tr>
<tr>
<td>Environment/AutosWindow/Show Address Column</td>
<td></td>
</tr>
<tr>
<td><strong>Show Variable Size Column</strong></td>
<td>Controls whether the variable size column is displayed.</td>
</tr>
<tr>
<td>Environment/AutosWindow/Show Size Column</td>
<td></td>
</tr>
<tr>
<td><strong>Show Variable Type Column</strong></td>
<td>Controls whether the variable type column is displayed.</td>
</tr>
<tr>
<td>Environment/AutosWindow/Show Type Column</td>
<td></td>
</tr>
</tbody>
</table>

### Call Stack

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Execution Frame at Top</strong></td>
<td>Controls whether the most recent call is at the top or the bottom of the list.</td>
</tr>
<tr>
<td>Environment/Call Stack/Most Recent At Top</td>
<td></td>
</tr>
<tr>
<td><strong>Show Call Address</strong></td>
<td>Enables the display of the call address in the call stack.</td>
</tr>
<tr>
<td>Environment/Call Stack/Show Call Address</td>
<td></td>
</tr>
<tr>
<td><strong>Show Call Source Location</strong></td>
<td>Enables the display of the call source location in the call stack.</td>
</tr>
<tr>
<td>Environment/Call Stack/Show Call Location</td>
<td></td>
</tr>
<tr>
<td><strong>Show Frame Size</strong></td>
<td>Enables the display of the amount of stack used by the call.</td>
</tr>
<tr>
<td>Environment/Call Stack/Show Stack Usage</td>
<td></td>
</tr>
<tr>
<td><strong>Show Frame Size In Bytes</strong></td>
<td>Display the stack usage in bytes rather than words.</td>
</tr>
<tr>
<td>Environment/Call Stack/Show Stack Usage In</td>
<td></td>
</tr>
<tr>
<td><strong>Show Parameter Names</strong></td>
<td>Enables the display of parameter names in the call stack.</td>
</tr>
<tr>
<td>Environment/Call Stack/Show Parameter Names</td>
<td></td>
</tr>
</tbody>
</table>
### Show Parameter Types
- **Environment/Call Stack/Show Parameter Types**
  - **Boolean**: Enables the display of parameter types in the call stack.

### Show Parameter Values
- **Environment/Call Stack/Show Parameter Values**
  - **Boolean**: Enables the display of parameter values in the call stack.

### Show Stack Pointer
- **Environment/Call Stack/Show Stack Pointer**
  - **Boolean**: Enables the display of the stack pointer in the call stack.

### Show Stack Usage
- **Environment/Call Stack/Show Cumulative Stack Usage**
  - **Boolean**: Enables the display of the amount of stack used.

### Show Stack Usage In Bytes
- **Environment/Call Stack/Show Cumulative Stack Usage In Bytes**
  - **Boolean**: Display the stack usage in bytes rather than words.

---

### Clipboard Ring

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Maximum Items Held In Ring</strong></td>
<td>The maximum number of items held on the clipboard ring before they are recycled.</td>
</tr>
<tr>
<td><strong>Environment/Clipboard Ring/Max Entries</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Preserve Contents Between Runs</strong></td>
<td>Save the clipboard ring across SEGGER Embedded Studio runs.</td>
</tr>
<tr>
<td><strong>Environment/Clipboard Ring/Save</strong></td>
<td></td>
</tr>
</tbody>
</table>

---

### Debug Terminal

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Backscroll Buffer Lines</strong></td>
<td>The number of lines you can see when you scroll backward in the debug terminal window.</td>
</tr>
<tr>
<td><strong>Debug Terminal/Backscroll Buffer Lines</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Use Window System Colors</strong></td>
<td>Substitute window system colors for ANSI black background and white foreground in debug terminal.</td>
</tr>
<tr>
<td><strong>Debug Terminal/Use Window System Colors</strong></td>
<td></td>
</tr>
</tbody>
</table>

---

### Globals

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Show Member Functions</strong></td>
<td>Controls whether C++ class member functions are displayed.</td>
</tr>
<tr>
<td><strong>Environment/GlobalsWindow/Show Member Functions</strong></td>
<td></td>
</tr>
</tbody>
</table>
### Show Variable Address Column

**Environment/GlobalsWindow/Show Address Column**

Controls whether the variable address column is displayed.

### Show Variable Size Column

**Environment/GlobalsWindow/Show Size Column**

Controls whether the variable size column is displayed.

### Show Variable Type Column

**Environment/GlobalsWindow/Show Type Column**

Controls whether the variable type column is displayed.

### Locals

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Member Functions</td>
<td>Controls whether C++ class member functions are displayed.</td>
</tr>
<tr>
<td>Show Variable Address Column</td>
<td>Controls whether the variable address column is displayed.</td>
</tr>
<tr>
<td>Show Variable Size Column</td>
<td>Controls whether the variable size column is displayed.</td>
</tr>
<tr>
<td>Show Variable Type Column</td>
<td>Controls whether the variable type column is displayed.</td>
</tr>
</tbody>
</table>

### Memory

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confirm Large Download</td>
<td>Present a warning if you attempt to download a large amount of memory in the memory window.</td>
</tr>
<tr>
<td>Scroll Wheel Modifies Start Address</td>
<td>Selects whether the mouse scroll wheel can change the memory window start address.</td>
</tr>
</tbody>
</table>

### Outline

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group #define Directives</td>
<td>Group consecutive #define and #undef preprocessor directives.</td>
</tr>
</tbody>
</table>
### Group #include Directives
Windows/Outline/Group Includes Boolean

Group consecutive #include preprocessor directives.

### Group Top-Level Declarations
Windows/Outline/Group Top Level Items Boolean

Group consecutive top-level variable and type declarations.

### Refresh Outline and Preview
Windows/Outline/Preview Refresh Mode Enumeration

How the Preview pane refreshes its contexts.

## Project Explorer

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Filename Replace Macros Environment/Project Explorer/Filename Replace Macros StringList</td>
<td>Macros (system and global) used to replace the start of a filename on project file addition.</td>
</tr>
<tr>
<td>Check Solution Target Environment/Project Explorer/Check Solution Target Boolean</td>
<td>Specifies whether to check target is correct when loading a solution.</td>
</tr>
<tr>
<td>Color Project Nodes Environment/Project Explorer/Color Nodes Boolean</td>
<td>Show the project nodes colored for identification in the Project Explorer.</td>
</tr>
<tr>
<td>Confirm Configuration Folder Delete Project Explorer/Confirm Configuration Folder Delete Boolean</td>
<td>Display a confirmation prompt before deleting a configuration folder containing properties.</td>
</tr>
<tr>
<td>Confirm File Replacement Warning Project Explorer/Confirm File Replacement Warning Boolean</td>
<td>Display a confirmation prompt before replacing project files for import and creation.</td>
</tr>
<tr>
<td>Confirm Forget Modified Options Project Explorer/Confirm Reject Property Changes Boolean</td>
<td>Display a confirmation prompt before forgetting option modifications.</td>
</tr>
<tr>
<td>Context Menu Edit Options At Top Environment/Project Explorer/Context Menu Properties Position Boolean</td>
<td>Controls where Edit Options are displayed by the Project Explorer's context menu.</td>
</tr>
<tr>
<td>Context Menu Uses Common Folder Environment/Project Explorer/Context Menu Common Folder Boolean</td>
<td>Controls how common options are displayed by the Project Explorer's context menu.</td>
</tr>
<tr>
<td>External Editor Environment/Project Explorer/External Editor FileName</td>
<td>The file name of the application to use as the external text editor. The external editor is started by holding down the Shift key when opening files from the project explorer.</td>
</tr>
<tr>
<td>Setting</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Highlight Dynamic Items</td>
<td>Show an overlay on an item if it is populated from a dynamic folder.</td>
</tr>
<tr>
<td>Highlight External Items</td>
<td>Show an overlay on an item if it is not held within the project directory.</td>
</tr>
<tr>
<td>Output Files Folder</td>
<td>Show the build output files in an Output Files folder in the project explorer.</td>
</tr>
<tr>
<td>Read-Only Data In Code</td>
<td>Configures whether read-only data contributes to the Code or Data statistic.</td>
</tr>
<tr>
<td>Show Dependencies</td>
<td>Controls how the dependencies are displayed.</td>
</tr>
<tr>
<td>Show Favorite Properties</td>
<td>Controls if favorite properties are displayed by the Project Explorer's context menu.</td>
</tr>
<tr>
<td>Show File Count on Folder</td>
<td>Show the number of files contained in a folder as a badge in the Project Explorer.</td>
</tr>
<tr>
<td>Show Modified Options on Folder/File</td>
<td>Show if a folder or file has modified options as a badge in the Project Explorer.</td>
</tr>
<tr>
<td>Show Options</td>
<td>Controls how the options are displayed.</td>
</tr>
<tr>
<td>Show Project Count on Solution</td>
<td>Show the number of projects contained in a solution as a badge in the Project Explorer.</td>
</tr>
<tr>
<td>Show Source Control Annotation</td>
<td>Annotate items in the project explorer with their source control status.</td>
</tr>
<tr>
<td>Show Statistics Rounded</td>
<td>Show exact or rounded sizes in the project explorer.</td>
</tr>
<tr>
<td>Source Control Status Column</td>
<td>Show the source control status column in the project explorer.</td>
</tr>
<tr>
<td>Starred Files Names</td>
<td>The list of wildcard-matched file names that are highlighted with stars, to bring attention to themselves, in the Project Explorer.</td>
</tr>
</tbody>
</table>
### Source Navigator

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Definitions Only</td>
<td>Show definitions only. When set to Yes only symbols that are defined will be included in the source navigator display. When set to No declarations of symbols will also be included in the source navigator display.</td>
</tr>
</tbody>
</table>

### Symbol Browser

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Field</td>
<td>Selects whether the Code field is displayed.</td>
</tr>
<tr>
<td>Const Field</td>
<td>Selects whether the Const field is displayed.</td>
</tr>
<tr>
<td>Data Field</td>
<td>Selects whether the Data field is displayed.</td>
</tr>
<tr>
<td>Frame Size Field</td>
<td>Selects whether the Frame Size field is displayed.</td>
</tr>
<tr>
<td>Range Field</td>
<td>Selects whether the Range field is displayed.</td>
</tr>
<tr>
<td>Section Field</td>
<td>Selects whether the Section field is displayed.</td>
</tr>
</tbody>
</table>
### Size Field

**Environment/Symbol Browser/Display**

**Size**

Boolean

Selects whether the Size field is displayed.

### Sort Criteria

**Environment/Symbol Browser/**

**Grouping**

Enumeration

Selects how to sort or group the symbols displayed.

### Type Field

**Environment/Symbol Browser/Display**

**Type**

Boolean

Selects whether the Type field is displayed.

### Value Field

**Environment/Symbol Browser/Display**

**Value**

Boolean

Selects whether the Value field is displayed.

---

### Terminal Emulator

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backscroll Buffer Lines</td>
<td>The number of lines you can see when you scroll backward in the terminal emulator window.</td>
</tr>
<tr>
<td>Baud Rate</td>
<td>Baud rate used when transmitting and receiving data.</td>
</tr>
<tr>
<td>Data Bits</td>
<td>Number of data bits to use when transmitting and receiving data.</td>
</tr>
<tr>
<td>Flow Control</td>
<td>The flow control method to use.</td>
</tr>
<tr>
<td>Line Feed On Carriage Return</td>
<td>Append a line feed character when a carriage return character is received.</td>
</tr>
<tr>
<td>Local Echo</td>
<td>Displays every character typed before sending to the remote computer.</td>
</tr>
<tr>
<td>Maximum Input Block Size</td>
<td>The maximum number of bytes to read at a time.</td>
</tr>
<tr>
<td>Parity</td>
<td>Parity used when transmitting and receiving data.</td>
</tr>
<tr>
<td>Port</td>
<td>The communications port to use, e.g. /dev/ttyS0, /dev/ttyS1, etc.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------</td>
</tr>
<tr>
<td><strong>Port Used By Target Interface</strong>&lt;br&gt;Terminal Emulator/Communications/Port Used By Target Interface</td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Set DTR</strong>&lt;br&gt;Terminal Emulator/Communications/DTR</td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Stop Bits</strong>&lt;br&gt;Terminal Emulator/Communications/Stop Bits</td>
<td>Enumeration</td>
</tr>
</tbody>
</table>

### Watch 1

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Show Member Functions</strong>&lt;br&gt;Environment/Watch1Window/Show Member Functions</td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Show Variable Address Column</strong>&lt;br&gt;Environment/Watch1Window/Show Address Column</td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Show Variable Size Column</strong>&lt;br&gt;Environment/Watch1Window/Show Size Column</td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Show Variable Type Column</strong>&lt;br&gt;Environment/Watch1Window/Show Type Column</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

### Watch 2

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Show Member Functions</strong>&lt;br&gt;Environment/Watch2Window/Show Member Functions</td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Show Variable Address Column</strong>&lt;br&gt;Environment/Watch2Window/Show Address Column</td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Show Variable Size Column</strong>&lt;br&gt;Environment/Watch2Window/Show Size Column</td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Show Variable Type Column</strong>&lt;br&gt;Environment/Watch2Window/Show Type Column</td>
<td>Boolean</td>
</tr>
</tbody>
</table>
# Watch 3

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Member Functions</td>
<td>Controls whether C++ class member functions are displayed.</td>
</tr>
<tr>
<td>Show Variable Address Column</td>
<td>Controls whether the variable address column is displayed.</td>
</tr>
<tr>
<td>Show Variable Size Column</td>
<td>Controls whether the variable size column is displayed.</td>
</tr>
<tr>
<td>Show Variable Type Column</td>
<td>Controls whether the variable type column is displayed.</td>
</tr>
</tbody>
</table>

# Watch 4

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Member Functions</td>
<td>Controls whether C++ class member functions are displayed.</td>
</tr>
<tr>
<td>Show Variable Address Column</td>
<td>Controls whether the variable address column is displayed.</td>
</tr>
<tr>
<td>Show Variable Size Column</td>
<td>Controls whether the variable size column is displayed.</td>
</tr>
<tr>
<td>Show Variable Type Column</td>
<td>Controls whether the variable type column is displayed.</td>
</tr>
</tbody>
</table>

# Windows

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer Grouping</td>
<td>How the files are grouped or listed in the Windows window.</td>
</tr>
<tr>
<td>Show File Path as Tooltip</td>
<td>Show the full file name as a tooltip when hovering over files in the Windows window.</td>
</tr>
<tr>
<td>Show Line Count and File Size</td>
<td>Environment/Windows/Show Sizes Boolean</td>
</tr>
</tbody>
</table>
# Code Options

## Assembler

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional Assembler Options</td>
<td>Enables additional options to be supplied to the assembler. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>asm_additional_options StringList</td>
<td></td>
</tr>
<tr>
<td>Additional Assembler Options From File</td>
<td>Enables additional options to be supplied to the assembler from a file. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>asm_additional_options_from_file ProjFileName</td>
<td></td>
</tr>
<tr>
<td>Assembler</td>
<td>Specifies which assembler to use.</td>
</tr>
<tr>
<td>arm_assembler_variant Enumeration</td>
<td></td>
</tr>
<tr>
<td>Backup Additional Assembler Options</td>
<td>Value of additional assembler options prior to generic options processing.</td>
</tr>
<tr>
<td>asm_additional_options_backup String</td>
<td></td>
</tr>
<tr>
<td>Generate Assembler Listing File</td>
<td>An assembler listing file is generated which can be found in the Output Files folder.</td>
</tr>
<tr>
<td>asm_generate_listing_file Boolean</td>
<td></td>
</tr>
</tbody>
</table>

## Build

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always Rebuild</td>
<td>Specifies whether or not to always rebuild the project/folder/file.</td>
</tr>
<tr>
<td>build_always_rebuild Boolean</td>
<td></td>
</tr>
<tr>
<td>Batch Build Configurations</td>
<td>The set of configurations to batch build.</td>
</tr>
<tr>
<td>batch_build_configurations StringList</td>
<td></td>
</tr>
<tr>
<td>Build Options Generic File Name</td>
<td>The file name containing the generic options.</td>
</tr>
<tr>
<td>build_generic_options_file_name ProjFileName</td>
<td></td>
</tr>
<tr>
<td>Build Quietly</td>
<td>Suppress the display of startup banners and information messages.</td>
</tr>
<tr>
<td>build_quietly Boolean</td>
<td></td>
</tr>
<tr>
<td>Dependency File Name</td>
<td>The file name to contain the dependencies.</td>
</tr>
<tr>
<td>build_dependency_file_name FileName</td>
<td></td>
</tr>
<tr>
<td>Enable Unused Symbol Removal</td>
<td>Enable the removal of unused symbols from the executable.</td>
</tr>
<tr>
<td>build_remove_unused_symbols Boolean</td>
<td></td>
</tr>
<tr>
<td>Exclude From Build</td>
<td>Specifies whether or not to exclude the project/folder/file from the build.</td>
</tr>
<tr>
<td>build_exclude_from_build Boolean</td>
<td></td>
</tr>
<tr>
<td>Include Debug Information</td>
<td>Specifies whether symbolic debug information is generated.</td>
</tr>
<tr>
<td>build_debug_information Boolean</td>
<td></td>
</tr>
<tr>
<td>Intermediate Directory</td>
<td>Specifies a relative path to the intermediate file directory. This property will have macro expansion applied to it. The macro $(IntDir) is set to this value.</td>
</tr>
<tr>
<td>build_intermediate_directory DirPath</td>
<td></td>
</tr>
<tr>
<td>Property Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Memory Map File</td>
<td>The name of the file containing the memory map description.</td>
</tr>
<tr>
<td>linker_memory_map_file</td>
<td>ProjFileName</td>
</tr>
<tr>
<td>Memory Map Macros</td>
<td>Macro values to substitute in memory map nodes. Each macro is defined as name=value and are separated by ;.</td>
</tr>
<tr>
<td>linker_memory_map_macros</td>
<td>StringList</td>
</tr>
<tr>
<td>Memory Segments</td>
<td>The start, access and size of named segments in the target, these are used when no memory map file is available. Each segment is specified by NAME RWX HEXSTART HEXSIZE for example FLASH RX 0x08000000 0x00010000</td>
</tr>
<tr>
<td>linker_section_placements_segments</td>
<td>String</td>
</tr>
<tr>
<td>Output Directory</td>
<td>Specifies a relative path to the output file directory. This property will have macro expansion applied to it. The macro $(OutDir) is set to this value. The macro $(RootRelativeOutDir) is set relative to the Root Output Directory if specified.</td>
</tr>
<tr>
<td>build_output_directory</td>
<td>DirPath</td>
</tr>
<tr>
<td>Project Can Build In Parallel</td>
<td>Specifies that dependent projects can be built in parallel. Default is No for Staging and Combining project types, Yes for all other project types.</td>
</tr>
<tr>
<td>project_can_build_in_parallel</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Project Dependencies</td>
<td>Specifies the projects the current project depends upon.</td>
</tr>
<tr>
<td>project_dependencies</td>
<td>StringList</td>
</tr>
<tr>
<td>Project Directory</td>
<td>Path of the project directory relative to the directory containing the project file. The macro $(ProjectDir) is set to the absolute path of this property.</td>
</tr>
<tr>
<td>project_directory</td>
<td>String</td>
</tr>
<tr>
<td>Project Macros</td>
<td>Specifies macro values which are expanded in project properties and for file names in Common configuration only. Each macro is defined as name=value and are separated by ;.</td>
</tr>
<tr>
<td>macros</td>
<td>StringList</td>
</tr>
<tr>
<td>Project Type</td>
<td>Specifies the type of project to build. The options are Executable, Library, Object file, Staging, Combining, Externally Built Executable, Externally Built Library, Externally Built Object file.</td>
</tr>
<tr>
<td>project_type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Property Groups File</td>
<td>The file containing the property groups for this project. This is applicable to Executable and Externally Built Executable project types only.</td>
</tr>
<tr>
<td>property_groups_file_path</td>
<td>ProjFileName</td>
</tr>
<tr>
<td>Root Output Directory</td>
<td>Allows a common root output directory to be specified that can be referenced using the $(RootOutDir) macro.</td>
</tr>
<tr>
<td>build_root_output_directory</td>
<td>DirPath</td>
</tr>
<tr>
<td>Suppress Warnings</td>
<td>Don't report warnings.</td>
</tr>
<tr>
<td>build_suppress_warnings</td>
<td>Boolean</td>
</tr>
<tr>
<td>Tool Chain Directory</td>
<td>Specify the root of the toolchain directory. This property will have macro expansion applied to it. The macro $(ToolChainDir) is set to this value.</td>
</tr>
<tr>
<td>build_toolchain_directory</td>
<td>DirPath</td>
</tr>
<tr>
<td>Treat Warnings as Errors</td>
<td>Treat all warnings as errors.</td>
</tr>
<tr>
<td>build_treat_warnings_as_errors</td>
<td>Boolean</td>
</tr>
</tbody>
</table>
### Use Compiler Driver

<table>
<thead>
<tr>
<th>use_compiler_driver</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The build will issue cc commands.</td>
</tr>
</tbody>
</table>

## Code Generation

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| ARM Advanced SIMD Auto Vectorize  
arm_advanced_SIMD_auto_vectorize | Enable automatic code generation for Advanced SIMD. |
| ARM Advanced SIMD Type  
arm_advanced_SIMD_type | Specifies the Advanced SIMD type to generate code for. The options are:  
- NEON - Cortex-A based processors |
| ARM Architecture  
arm_architecture | Specifies the version of the instruction set to generate code for. The options are:  
- v4T - ARM7TDMI and ARM920T processors  
v5TE - ARM9E, Feroceon and XScale processors  
v6 - ARM11 processors  
v6M - Cortex-M0 and Cortex-M1 processors  
v7M - Cortex-M3 processors  
v7EM - Cortex-M4 and Cortex-M7 processors  
v7R - Cortex-R processors  
v7A - Cortex-A processors  
v8R - Cortex-R AArch32 processors  
v8A - Cortex-A AArch32 processors  
v8M_Baseline - Cortex M23 processor  
v8M_Mainline - Cortex M33 processor  
None |
<p>| The corresponding preprocessor definitions: | |
| <em><strong>ARM_ARCH_4T</strong></em> | |
| <em><strong>ARM_ARCH_5TE</strong></em> | |
| <em><strong>ARM_ARCH_6</strong></em> | |
| <em><strong>ARM_ARCH_6M</strong></em> | |
| <em><strong>ARM_ARCH_7M</strong></em> | |
| <em><strong>ARM_ARCH_7EM</strong></em> | |
| <em><strong>ARM_ARCH_7R</strong></em> | |
| <em><strong>ARM_ARCH_7A</strong></em> | |
| <em><strong>ARM_ARCH_8R</strong></em> | |
| <em><strong>ARM_ARCH_8A</strong></em> | |
| <em><strong>ARM_ARCH_8M_BASELINE</strong></em> | |
| <em><strong>ARM_ARCH_8M_MAINLINE</strong></em> | |
| are defined. | |</p>
<table>
<thead>
<tr>
<th>ARM Core Type</th>
<th>Specifies the core to generate code for. The options are:</th>
</tr>
</thead>
<tbody>
<tr>
<td>arm_core_type</td>
<td>ARM7TDMI, ARM7TDMI-S, ARM720T</td>
</tr>
<tr>
<td></td>
<td>ARM920T, ARM946E-S, ARM966E-S, ARM968E-S, ARM926EJ-S</td>
</tr>
<tr>
<td></td>
<td>ARM1136J-S, ARM1136JF-S, ARM1176JZ-S, ARM1176JZF-S</td>
</tr>
<tr>
<td></td>
<td>Cortex-M0, Cortex-M0+, Cortex-M1, Cortex-M23, cortex-M3,</td>
</tr>
<tr>
<td></td>
<td>cortex-M33, Cortex-M4, Cortex-M7</td>
</tr>
<tr>
<td></td>
<td>Cortex-R4, Cortex-R4F, Cortex-R5, Cortex-R7, Cortex-R8</td>
</tr>
<tr>
<td></td>
<td>Cortex-R52, Cortex-A5, Cortex-A7, Cortex-A8, Cortex-A9,</td>
</tr>
<tr>
<td></td>
<td>Cortex-A15, Cortex-A17</td>
</tr>
<tr>
<td></td>
<td>Cortex-A32, Cortex-A35, Cortex-A53, Cortex-A57, Cortex-A72, Cortex-A73, Cortex-A75</td>
</tr>
<tr>
<td></td>
<td>XScale</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>IF this property is set to None then the architecture property is used</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ARM FP ABI Type</th>
<th>Specifies the FP ABI type to generate code for. The options are:</th>
</tr>
</thead>
<tbody>
<tr>
<td>arm_fp_abi</td>
<td><strong>Soft</strong> generate calls to the C library to implement floating point operations.</td>
</tr>
<tr>
<td></td>
<td><strong>SoftFP</strong> generate VFP code to implement floating point operations.</td>
</tr>
<tr>
<td></td>
<td><strong>Hard</strong> generate VFP code to implement floating point operations and use VFP registers to pass floating point parameters on function calls.</td>
</tr>
<tr>
<td></td>
<td><strong>None</strong> will not specify the FP ABI or the FPU.</td>
</tr>
</tbody>
</table>
### ARM FPU Type

**arm_fpu_type**<br>**Enumeration**<br>Specifies the FPU type to generate code for. The options are:<br>- VFP - ARM9/ARM11 based processors<br>- VFP9 - the same as VFP<br>- VFPv3-D32 - Cortex-A/Cortex-R based processors<br>- VFPv3-D16 - Cortex-A/Cortex-R based processors<br>- VFPv4-D32 - Cortex-A/Cortex-R based processors<br>- VFPv4-D16 - Cortex-A/Cortex-R based processors<br>- FPv4-SP-D16 - Cortex-M4 processors<br>- FPv5-SP-D16 - Cortex-M7 processors<br>- FPv5-D16 - Cortex-M7 processors<br>- FP-ARMv8 - Cortex-A AArch32 processors<br>---<br>The corresponding preprocessor definitions:<br>```
__ARM_ARCH_VFP__
__ARM_ARCH_VFP3_D32__
__ARM_ARCH_VFP3_D16__
__ARM_ARCH_VFP4_D32__
__ARM_ARCH_VFP4_D16__
__ARM_ARCH_FPV4_SP_D16__
__ARM_ARCH_FPV5_SP_D16__
__ARM_ARCH_FPV5_D16__
__ARM_ARCH_FP_ARMv8__
```
are defined.

### ARM/Thumb Interworking

**arm_interwork**<br>**Enumeration**<br>Specifies whether ARM/Thumb interworking code should be generated. Setting this property to **No** may result in smaller code sizes when compiling for architecture **v4T**.

### Byte Order

**arm_endian**<br>**Enumeration**<br>Specify the byte order of the target processor. The options are:<br>- **Little** little endian code and data.<br>- **Big** big endian code and data.<br>- **BE-8** little endian code and big endian data.

### CM0/CM0+/CM1 Has Small Multiplier

**arm_cm0_has_small_multiplier**<br>**Boolean**<br>The CM0/CM0+/CM1 core has the small multiplier.

### Debugging Level

**gcc_debugging_level**<br>**Enumeration**<br>Specifies the level of debugging information to generate. The options are:<br>- **None** - no debugging information<br>- Level 1 - backtrace and line number debugging information<br>- Level 2 - Level 1 and variable display debugging information<br>- Level 3 - Level 2 and macro display debugging information
<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dwarf Version</td>
<td></td>
<td>Specifies the version of Dwarf debugging information to generate.</td>
</tr>
<tr>
<td>Emit Assembler CFI</td>
<td></td>
<td>Emit DWARF 2 unwind info using GAS .cfi_* directives rather than a compiler generated .eh_frame section.</td>
</tr>
<tr>
<td>Enable Exception Support</td>
<td></td>
<td>Specifies whether exception support is enabled for C++ programs.</td>
</tr>
<tr>
<td>Enable RTTI Support</td>
<td></td>
<td>Specifies whether RTTI support is enabled for C++ programs.</td>
</tr>
<tr>
<td>Enumeration Size</td>
<td></td>
<td>Select between minimal container sized enumerations and int sized enumerations.</td>
</tr>
<tr>
<td>Instruction Set</td>
<td></td>
<td>Specifies the instruction set to generate code for.</td>
</tr>
<tr>
<td>Instrument Functions</td>
<td></td>
<td>Specifies whether instrumentation calls are generated for function entry and exit.</td>
</tr>
<tr>
<td>Keep Link Time Optimization Intermediate Files</td>
<td></td>
<td>Specifies whether to keep the link time optimization resolution and object files.</td>
</tr>
<tr>
<td>Link Time Optimization</td>
<td></td>
<td>Specifies whether the project should be built for optimization at link time.</td>
</tr>
<tr>
<td>Long Calls</td>
<td></td>
<td>Specifies whether function calls are made using absolute addresses.</td>
</tr>
<tr>
<td>Math Errno</td>
<td></td>
<td>Set errno after calling math functions that are executed with a single instruction, e.g., sqrt.</td>
</tr>
<tr>
<td>Merge Globals [clang]</td>
<td></td>
<td>Select whether global declarations are merged. This may reduce code size and increase execution speed for some applications. However, if functions are not used in an application and are eliminated by the linker, merged globals may increase the data size requirement of an application.</td>
</tr>
<tr>
<td>No COMMON</td>
<td></td>
<td>Don’t put globals in the common section</td>
</tr>
<tr>
<td>Omit Frame Pointer</td>
<td></td>
<td>Specifies whether a frame pointer register is omitted if not required.</td>
</tr>
<tr>
<td>Optimization Level</td>
<td></td>
<td>Specifies the optimization level to use. The options are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- None - don’t specify an optimization level</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Level 0 - no optimization, fastest compilation and best debug experience.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Level 1 - optimize minimally.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Level 2 for speed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Level 2 balanced</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Level 2 for size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Level 3 - optimize even more, will take longer to compile and may produce much larger code.</td>
</tr>
</tbody>
</table>
### Unaligned Access Support

**arm_unaligned_accessEnumeration**

Unaligned word and half-words can be accessed. The option are:

- **Yes** enable unaligned word and half-words.
- **No** disable unaligned word and half-words.
- **Auto** disable unaligned word and half-word access for v4T/v5TE/v6M/v8M_Baseline architectures, enable for others.

### Unwind Tables

**arm_unwind_tablesBoolean**

Generate unwind tables for C code.

### Use Builtins

**arm_use_builtinsBoolean**

Use built-in library functions e.g. scanf.

### Wide Character Size

**gcc_wchar_sizeEnumeration**

Select between standard 32-bit or shorter 16-bit size for wide characters and wchar_t.

- **v7A/v7R Has Integer Divide Instructions**

  **arm_v7_has_divide_instructionsBoolean**

  The v7A/v7R architecture has integer divide instructions in both ARM and Thumb instruction sets.

- **v8A Has CRC Instructions**

  **arm_v8A_has_crcBoolean**

  The v8A architecture has CRC instructions.

- **v8A Has Crypto Instructions**

  **arm_v8A_has_cryptoBoolean**

  The v8A architecture has crypto instructions.

- **v8M Has CMSE Instructions**

  **arm_v8M_has_cmseBoolean**

  The v8M architecture has CMSE instructions.

- **v8M Has DSP Instructions**

  **arm_v8M_has_dspBoolean**

  The v8M architecture has DSP instructions.

### Combining

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Combine Command</strong></td>
<td>The command to execute. This property will have macro expansion applied to it with the macro $(CombiningOutputFilePath) set to the output filepath of the combine command and the macro $(CombiningRelInputPaths) is set to the (project relative) names of all of the files in the project.</td>
</tr>
<tr>
<td><strong>Combine Command Working Directory</strong></td>
<td>The working directory in which the combine command is run. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td><strong>Output File Path</strong></td>
<td>The output file path the stage command will create. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td><strong>Set To Read-only</strong></td>
<td>Set the output file to read only or read/write.</td>
</tr>
</tbody>
</table>

1020
## Compiler

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Additional C Compiler Only Options</strong>&lt;br&gt;`c_only_additional_options`&lt;br&gt;`StringList`</td>
<td>Enables additional options to be supplied to the C compiler only. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td><strong>Additional C Compiler Only Options From File</strong>&lt;br&gt;`c_only_additional_options_from_file`&lt;br&gt;`ProjFileName`</td>
<td>Enables additional options to be supplied to the C compiler only from a file. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td><strong>Additional C++ Compiler Only Options</strong>&lt;br&gt;`cpp_only_additional_options`&lt;br&gt;`StringList`</td>
<td>Enables additional options to be supplied to the C++ compiler only. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td><strong>Additional C++ Compiler Only Options From File</strong>&lt;br&gt;`cpp_only_additional_options_from_file`&lt;br&gt;`ProjFileName`</td>
<td>Enables additional options to be supplied to the C++ compiler only from a file. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td><strong>Additional C/C++ Compiler Options</strong>&lt;br&gt;`c_additional_options`&lt;br&gt;`StringList`</td>
<td>Enables additional options to be supplied to the C/C++ compiler. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td><strong>Additional C/C++ Compiler Options From File</strong>&lt;br&gt;`c_additional_options_from_file`&lt;br&gt;`ProjFileName`</td>
<td>Enables additional options to be supplied to the C/C++ compiler from a file. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td><strong>Backup Additional C Compiler Only Options</strong>&lt;br&gt;`c_only_additional_options_backup`&lt;br&gt;`String`</td>
<td>Value of additional C compiler options prior to generic options processing.</td>
</tr>
<tr>
<td><strong>Backup Additional C++ Compiler Only Options</strong>&lt;br&gt;`cpp_only_additional_options_backup`&lt;br&gt;`String`</td>
<td>Value of additional C++ compiler options prior to generic options processing.</td>
</tr>
<tr>
<td><strong>Backup Additional Compiler Options</strong>&lt;br&gt;`c_additional_options_backup`&lt;br&gt;`String`</td>
<td>Value of additional compiler options prior to generic options processing.</td>
</tr>
<tr>
<td><strong>C Language Standard</strong>&lt;br&gt;`gcc_c_language_standard`&lt;br&gt;`Enumeration`</td>
<td>Specifies the language standard to use when compiling C files. The options are: None - don’t specify a language standard&lt;br&gt;c89/gnu89&lt;br&gt;c90/gnu90&lt;br&gt;c99/gnu99&lt;br&gt;c11/gnu11</td>
</tr>
<tr>
<td><strong>C++ Language Standard</strong>&lt;br&gt;`gcc_cplusplus_language_standard`&lt;br&gt;`Enumeration`</td>
<td>Specifies the language standard to use when compiling C files. The options are: None - don’t specify a language standard&lt;br&gt;++89/gnu++89&lt;br&gt;++11/gnu++11&lt;br&gt;++14/gnu++14&lt;br&gt;++17/gnu++17</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Compiler</td>
<td>Specifies which compiler to use.</td>
</tr>
<tr>
<td>Enable All Warnings</td>
<td>Enables all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros.</td>
</tr>
<tr>
<td>Enable All Warnings C Compiler Only Command Line Options</td>
<td>The command line options supplied to the C compiler when Enable All Warnings is enabled.</td>
</tr>
<tr>
<td>Enable All Warnings C++ Compiler Only Command Line Options</td>
<td>The command line options supplied to the C++ compiler when Enable All Warnings is enabled.</td>
</tr>
<tr>
<td>Enable All Warnings Command Line Options</td>
<td>The command line options supplied to the compiler when Enable All Warnings is enabled.</td>
</tr>
<tr>
<td>Enforce ANSI Checking</td>
<td>Perform additional checks for ensure strict conformance to the selected ISO (ANSI) C or C++ standard.</td>
</tr>
<tr>
<td>Enforce ANSI Checking C Command Line Options</td>
<td>The command line options supplied to the C compiler when Enforce ANSI Checking is enabled.</td>
</tr>
<tr>
<td>Enforce ANSI Checking C++ Command Line Options</td>
<td>The command line options supplied to the C++ compiler when Enforce ANSI Checking is enabled.</td>
</tr>
<tr>
<td>Enforce ANSI Checking Command Line Options</td>
<td>The command line options supplied to the compiler when Enforce ANSI Checking is enabled.</td>
</tr>
<tr>
<td>GNU Version</td>
<td>Specifies value of <strong>GNU</strong> and related macros.</td>
</tr>
<tr>
<td>Keep Assembly Source</td>
<td>Specifies whether assembly code generated by the compiler is kept.</td>
</tr>
<tr>
<td>Keep Preprocessor Output</td>
<td>Specifies whether preprocessor output generated by the compiler is kept.</td>
</tr>
<tr>
<td>Object File Name</td>
<td>Specifies a name to override the default object file name.</td>
</tr>
<tr>
<td>Supply Absolute File Path</td>
<td>Specifies whether absolute file paths are supplied to the compiler.</td>
</tr>
</tbody>
</table>

### External Build

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>External Command</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Archive Command</td>
<td>external_archive_command</td>
</tr>
<tr>
<td></td>
<td>The command line to archive object files. This property will have macro expansion applied to it with the additional macros:</td>
</tr>
<tr>
<td></td>
<td>$(TargetPath) contains the full file name of the Library File Name property.</td>
</tr>
<tr>
<td></td>
<td>$(RelTargetPath) contains the project directory relative file name of the Object File Name property.</td>
</tr>
<tr>
<td></td>
<td>$(Objects) a space separated list of files to archive, generated from the source files of the project OR.</td>
</tr>
<tr>
<td></td>
<td>$(ObjectsFilePath) contains the full file name of the file containing the list of files to archive.</td>
</tr>
<tr>
<td></td>
<td>$(RelObjectsFilePath) contains the project directory relative file name of the file containing the list of files to link.</td>
</tr>
<tr>
<td>Assemble Command</td>
<td>external_assemble_command</td>
</tr>
<tr>
<td></td>
<td>The command line to assemble an assembly source file. This property will have macro expansion applied to it with the additional macros:</td>
</tr>
<tr>
<td></td>
<td>$(TargetPath) contains the full file name of the Object File Name property.</td>
</tr>
<tr>
<td></td>
<td>$(RelTargetPath) contains the project directory relative file name of the Object File Name property.</td>
</tr>
<tr>
<td></td>
<td>$(AsmOptions) contains a space seperated list of options as set in the Additional Assembler Options property.</td>
</tr>
<tr>
<td></td>
<td>$(DependencyPath) contains the filename of the .d file that is required to be output by the compilation for dependency support.</td>
</tr>
<tr>
<td></td>
<td>$(RelDependencyPath) contains the relative filename of the .d file that is required to be output by the compilation for dependency support.</td>
</tr>
<tr>
<td></td>
<td>$(Defines) contains a space seperated list of preprocessor definitions as set in the Preprocessor Definitions property.</td>
</tr>
<tr>
<td></td>
<td>$(Undefines) contains a space seperated list of preprocessor undefinitions as set in the Preprocessor Definitions property.</td>
</tr>
<tr>
<td></td>
<td>$(Includes) contains a space seperated list of user include directories as set in the User Include Directories property.</td>
</tr>
<tr>
<td>Build Command</td>
<td>external_build_command</td>
</tr>
<tr>
<td></td>
<td>The command line to build the executable e.g. make. This property will have macro expansion applied to it.</td>
</tr>
</tbody>
</table>
The command line to compile a C source file. This property will have macro expansion applied to it with the additional macros:

- $(TargetPath) contains the full file name of the **Object File Name** property.
- $(RelTargetPath) contains the project directory relative file name of the **Object File Name** property.
- $(COptions) contains a space separated list of options as set in the **C Additional C/C++ Compiler Options** property.
- $(COnlyOptions) contains a space separated list of options as set in the **C Additional C Compiler Only Options** property.
- $(DependencyPath) contains the filename of the .d file that is required to be output by the compilation for dependency support.
- $(RelDependencyPath) contains the relative filename of the .d file that is required to be output by the compilation for dependency support.
- $(Defines) contains a space separated list of preprocessor definitions as set in the **Preprocessor Definitions** property.
- $(Undefines) contains a space separated list of preprocessor undefinitions as set in the **Preprocessor Definitions** property.
- $(Includes) contains a space separated list of user include directories as set in the **User Include Directories** property.
<table>
<thead>
<tr>
<th>C++ Compile Command</th>
<th>The command line to compile a C++ source file. This property will have macro expansion applied to it with the additional macros:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>external_cpp_compile_command</code></td>
<td>- <code>${TargetPath}</code> contains the full file name of the <strong>Object File Name</strong> property.</td>
</tr>
<tr>
<td></td>
<td>- <code>${RelTargetPath}</code> contains the project directory relative file name of the <strong>Object File Name</strong> property.</td>
</tr>
<tr>
<td></td>
<td>- <code>${COptions}</code> contains a space separated list of options as set in the <strong>C Additional C/C++ Compiler Options</strong> property.</td>
</tr>
<tr>
<td></td>
<td>- <code>${CppOnlyOptions}</code> contains a space separated list of options as set in the <strong>C Additional C++ Compiler Only Options</strong> property.</td>
</tr>
<tr>
<td></td>
<td>- <code>${DependencyPath}</code> contains the filename of the .d file that is required to output by the compilation for dependency support.</td>
</tr>
<tr>
<td></td>
<td>- <code>${RelDependencyPath}</code> contains the relative filename of the .d file that is required to output by the compilation for dependency support.</td>
</tr>
<tr>
<td></td>
<td>- <code>${Defines}</code> contains a space separated list of preprocessor definitions as set in the <strong>Preprocessor Definitions</strong> property.</td>
</tr>
<tr>
<td></td>
<td>- <code>${Undefines}</code> contains a space separated list of preprocessor undefinitions as set in the <strong>Preprocessor Definitions</strong> property.</td>
</tr>
<tr>
<td></td>
<td>- <code>${Includes}</code> contains a space separated list of user include directories as set in the <strong>User Include Directories</strong> property.</td>
</tr>
</tbody>
</table>
The command line to link an executable. This property will have macro expansion applied to it with the additional macros:

- `${TargetPath}` contains the full file name of the `Executable File Name` property.
- `${RelTargetPath}` contains the project directory relative file name of the `Executable File Name` property.
- `${LinkOptions}` contains a space separated list of options as set in the `Additional Linker Options` property.
- `${Objects}` contains a space separated list of files to link, generated from the source files of the project and the outputs of any dependent projects OR.
- `${ObjectsFilePath}` contains the full file name of the file containing the list of files to link.
- `${RelObjectsFilePath}` contains the project directory relative file name of the file containing the list of files to link.
- `${LinkerScriptPath}` contains the full file name of the `Linker Script File` property.
- `${RelLinkerScriptPath}` contains the project directory relative file name of the `Linker Script File` property.
- `${MapPath}` contains the full file name of the required map file.
- `${RelMapPath}` contains the project directory relative file name of the required map file.

The command line to clean the executable e.g. `make clean`. This property will have macro expansion applied to it.
### Link Command

**external_link_command**: Unknown

The command line to link an executable. This property will have macro expansion applied to it with the additional macros:

- `${TargetPath}` contains the full file name of the **Executable File Name** property.
- `${RelTargetPath}` contains the project directory relative file name of the **Executable File Name** property.
- `${LinkOptions}` contains a space separated list of options as set in the **Additional Linker Options** property.
-`${Objects}` a space separated list of files to link, generated from the source files of the project and the outputs of any dependent projects OR.
- `${ObjectsFilePath}` contains the full file name of the file containing the list of files to link
-`${RelObjectsFilePath}` contains the project directory relative file name of the file containing the list of files to link
-`${LinkerScriptPath}` contains the full file name of the **Linker Script File** property.
-`${RelLinkerScriptPath}` contains the project directory relative file name of the **Linker Script File** property.
-`${MapPath}` contains the full file name of the required map file.
-`${RelMapPath}` contains the project directory relative file name of the required map file.

### Objects File

**external_objects_file_name**: Unknown

The name of the file containing the list of files to archive or link, generated from the source files of the project. This property will have macro expansion applied to it. The macro `${ObjectsFilePath}` is set to this value.

### File

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File Encoding</strong></td>
<td><strong>file_codec</strong> Enumeration</td>
</tr>
<tr>
<td></td>
<td>Specifies the encoding to use when reading and writing the file.</td>
</tr>
</tbody>
</table>
**File Name**

**file_name**

String

The name of the file. This property will have global macro expansion applied to it. The following macros are set based on the value: $(InputDir) relative directory of file, $(InputName) file name without directory or extension, $(InputFileName) file name, $(InputExt) file name extension, $(InputPath) absolute path to the file name, $(RelInputPath) relative path from project directory to the file name.

**File Open Action**

**file_open_with**

Enumeration

Specifies how to open the file when it is double clicked.

**File Type**

**file_type**

Enumeration

The type of file. Default setting uses the file extension to determine file type.

**Flag**

**file_flag**

Enumeration

Flag which you can use to draw attention to important files in your project.

---

### Folder

**Property**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Folder Directory</td>
</tr>
<tr>
<td><strong>pathDirPath</strong></td>
</tr>
<tr>
<td>Dynamic folder directory specification.</td>
</tr>
<tr>
<td>Dynamic Folder Exclude</td>
</tr>
<tr>
<td><strong>exclude</strong> StringList</td>
</tr>
<tr>
<td>Dynamic folder exclude specification - ; seperated wildcards.</td>
</tr>
<tr>
<td>Dynamic Folder Filter</td>
</tr>
<tr>
<td><strong>filter</strong> String</td>
</tr>
<tr>
<td>Dynamic folder filter specification - ; seperated wildcards.</td>
</tr>
<tr>
<td>Dynamic Folder Recurse</td>
</tr>
<tr>
<td><strong>recurse</strong> Boolean</td>
</tr>
<tr>
<td>Dynamic folder recurse into subdirectories.</td>
</tr>
<tr>
<td>Unity Build Exclude Filter</td>
</tr>
<tr>
<td><strong>unity_build_exclude_filter</strong> String</td>
</tr>
<tr>
<td>The filter specification to exclude from the unity build - ; seperated wildcards.</td>
</tr>
<tr>
<td>Unity Build File Name</td>
</tr>
<tr>
<td><strong>unity_build_file_name</strong> FileName</td>
</tr>
<tr>
<td>The file name created that #includes all files in the folder for the unity build.</td>
</tr>
</tbody>
</table>

---

### General

**Property**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment Variables</td>
</tr>
<tr>
<td><strong>environment_variables</strong> StringList</td>
</tr>
<tr>
<td>Environment variables to set on solution load.</td>
</tr>
<tr>
<td>Inherited Configurations</td>
</tr>
<tr>
<td><strong>inherited_configurations</strong> StringList</td>
</tr>
<tr>
<td>The list of configurations that are inherited by this configuration.</td>
</tr>
</tbody>
</table>
## Library

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exclude Default Library Helper Functions</td>
<td>Specifies whether to exclude default library helper functions.</td>
</tr>
<tr>
<td><code>link_use_multi_threaded_libraries</code> String</td>
<td>Boolean</td>
</tr>
<tr>
<td>Include Standard Libraries</td>
<td>Specifies whether the standard libraries should be linked into your application.</td>
</tr>
<tr>
<td><code>link_include_standard_libraries</code> String</td>
<td>Boolean</td>
</tr>
<tr>
<td>Library ARM Architecture</td>
<td>Specifies the architecture variant of the library to link with. The default uses the ARM Architecture value</td>
</tr>
<tr>
<td><code>arm_library_architecture</code> String</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Library File Name</td>
<td>Specifies a name to override the default library file name.</td>
</tr>
<tr>
<td><code>build_output_file_name</code> String</td>
<td>FileName</td>
</tr>
<tr>
<td>Library Instruction Set</td>
<td>Specifies the instruction set variant of the libraries to link with.</td>
</tr>
<tr>
<td><code>arm_library_instruction_set</code> String</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Library Optimization</td>
<td>Specifies whether to link with libraries optimized for speed or size.</td>
</tr>
<tr>
<td><code>arm_library_optimization</code> String</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Standard Libraries Directory</td>
<td>Specifies where to find the standard libraries</td>
</tr>
<tr>
<td><code>link_standard_libraries_directory</code> String</td>
<td>String</td>
</tr>
</tbody>
</table>

## Linker

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional Input Files</td>
<td>Enables additional object and library files to be supplied to the linker.</td>
</tr>
<tr>
<td><code>linker_additional_files</code> String</td>
<td>StringList</td>
</tr>
<tr>
<td>Additional Linker Options</td>
<td>Enables additional options to be supplied to the linker.</td>
</tr>
<tr>
<td><code>linker_additional_options</code> String</td>
<td>StringList</td>
</tr>
<tr>
<td>Additional Linker Options From File</td>
<td>Enables additional options to be supplied to the linker from a file.</td>
</tr>
<tr>
<td><code>linker_additional_options_from_file</code> String</td>
<td>ProjFileName</td>
</tr>
<tr>
<td>Additional Linker Script Generator Options</td>
<td>Enables additional options to be supplied to the linker script generator</td>
</tr>
<tr>
<td><code>arm_additional_mkld_options</code> String</td>
<td>StringList</td>
</tr>
<tr>
<td>Additional Output File Gap Fill Value</td>
<td>The value to fill gaps between sections in additional output file.</td>
</tr>
<tr>
<td><code>arm_linker_additional_output_file_gap_fill</code> String</td>
<td></td>
</tr>
<tr>
<td>Additional Output Format</td>
<td>The format used when creating an additional linked output file. The options are:</td>
</tr>
<tr>
<td><code>linker_output_format</code> String</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Allow Multiple Symbol Definition</td>
<td>Do not report error if the same symbol is defined more than once in object files/libraries.</td>
</tr>
<tr>
<td><code>arm_linker_allow_multiple_definition</code> String</td>
<td>Boolean</td>
</tr>
<tr>
<td>Backup Additional Linker Options</td>
<td>Value of additional linker options prior to generic options processing</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>link_additional_options_backup_string</code></td>
<td></td>
</tr>
<tr>
<td>CMSE Import Library File</td>
<td>Specifies the name of the CMSE import library to generate.</td>
</tr>
<tr>
<td><code>arm_linker_cmse_import_library_file_name</code></td>
<td></td>
</tr>
<tr>
<td>Check CMSE Import Library File</td>
<td>Specifies the name of the file to check the generated CMSE import library with.</td>
</tr>
<tr>
<td><code>arm_linker_check_cmse_import_library_file</code></td>
<td></td>
</tr>
<tr>
<td>Check For Memory Segment Overflow</td>
<td>Specifies whether the linker should check whether program sections fit in their memory segments.</td>
</tr>
<tr>
<td><code>arm_library_check_memory_segment_overflow</code></td>
<td></td>
</tr>
<tr>
<td>DebugIO Implementation</td>
<td>Specifies which DebugIO mechanism to link in. Options are <strong>Breakpoint</strong> (hardware breakpoint instruction and memory locations are used, not available on v4t architecture), <strong>DCC</strong> (ARM debug communication channel is used), and <strong>Memory Poll</strong> (memory locations are polled).</td>
</tr>
<tr>
<td><code>arm_link_debugio_type</code></td>
<td></td>
</tr>
<tr>
<td>Deduplicate Code Sections [segger-ld]</td>
<td>Specifies whether the linker finds readonly code sections that are identical and discard duplicates.</td>
</tr>
<tr>
<td><code>link_dedupe_code</code></td>
<td></td>
</tr>
<tr>
<td>Deduplicate Data Sections [segger-ld]</td>
<td>Specifies whether the linker finds readonly data sections that are identical and discard duplicates.</td>
</tr>
<tr>
<td><code>link_dedupe_data</code></td>
<td></td>
</tr>
<tr>
<td>Default Fill Pattern</td>
<td>Specifies the default pattern used to fill unspecified regions of memory in a generated linker script. This pattern may be overridden by the <code>fill</code> attribute of a program section in the section placement file.</td>
</tr>
<tr>
<td><code>arm_linker_script_generator_default_fill_pattern</code></td>
<td></td>
</tr>
<tr>
<td>Entry Point</td>
<td>Specifies the entry point of the program.</td>
</tr>
<tr>
<td><code>gcc_entry_point</code></td>
<td></td>
</tr>
<tr>
<td>Generate Log File [segger-ld]</td>
<td>Specifies whether to generate a linkage log file.</td>
</tr>
<tr>
<td><code>linker_log_file</code></td>
<td></td>
</tr>
<tr>
<td>Generate Map File</td>
<td>Specifies whether to generate a linkage map file.</td>
</tr>
<tr>
<td><code>linker_map_file</code></td>
<td></td>
</tr>
<tr>
<td>Inline Small Functions [segger-ld]</td>
<td>Specifies whether the linker inlines small functions at the call site rather than calling the function.</td>
</tr>
<tr>
<td><code>link_inline</code></td>
<td></td>
</tr>
<tr>
<td>Keep Symbols</td>
<td>Specifies the symbols that should be kept by the linker even if they are not reachable.</td>
</tr>
<tr>
<td><code>linker_keep_symbols</code></td>
<td></td>
</tr>
<tr>
<td>Link Dependent Projects</td>
<td>Specifies whether to link the output of dependent library projects.</td>
</tr>
<tr>
<td><code>link_dependent_projects</code></td>
<td></td>
</tr>
<tr>
<td>Linker</td>
<td>Specifies which linker to use.</td>
</tr>
<tr>
<td><code>arm_linker_variant</code></td>
<td></td>
</tr>
<tr>
<td>Linker Script File</td>
<td>The name of the manual linker script file.</td>
</tr>
<tr>
<td><code>link_linker_script_file</code></td>
<td></td>
</tr>
<tr>
<td>Linker Symbol Definitions</td>
<td>Specifies one or more linker symbol definitions.</td>
</tr>
<tr>
<td><code>link_symbol_definitions</code></td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Merge String Constants [segger-id]</td>
<td>Specifies whether the linker merges duplicate string constants.</td>
</tr>
<tr>
<td>link_merge_stringsBoolean</td>
<td></td>
</tr>
<tr>
<td>No Enum Size Warning</td>
<td>Do not generate warnings when object files have different ARM EABI enum size attributes.</td>
</tr>
<tr>
<td>arm_linker_no_enum_size_warningsBoolean</td>
<td></td>
</tr>
<tr>
<td>No Wide Char Size Warning</td>
<td>Do not generate warnings when object files have different ARM EABI wide character size attributes.</td>
</tr>
<tr>
<td>arm_linker_no_wchar_size_warningsBoolean</td>
<td></td>
</tr>
<tr>
<td>Section Placement File</td>
<td>The name of the file containing section placement description.</td>
</tr>
<tr>
<td>linker_section_placement_fileProjFileName</td>
<td></td>
</tr>
<tr>
<td>Section Placement Macros</td>
<td>Macro values to substitute in section placement nodes - MACRO1=value1;MACRO2=value2.</td>
</tr>
<tr>
<td>linker_section_placement_macrosStringList</td>
<td></td>
</tr>
<tr>
<td>Strip Debug Information</td>
<td>Specifies whether debug information should be stripped from the linked image.</td>
</tr>
<tr>
<td>linker_strip_debug_informationBoolean</td>
<td></td>
</tr>
<tr>
<td>Strip Symbols</td>
<td>Specifies whether symbols should be stripped.</td>
</tr>
<tr>
<td>gcc_strip_symbolsBoolean</td>
<td></td>
</tr>
<tr>
<td>Supply Memory Segments To Linker [segger-id]</td>
<td>Specifies whether to supply memory segments on the linker command line.</td>
</tr>
<tr>
<td>linker_supply_memory_segmentsBoolean</td>
<td></td>
</tr>
<tr>
<td>Suppress Warning on Mismatch</td>
<td>No warning on mismatched object files/libraries.</td>
</tr>
<tr>
<td>arm_linker_no_warn_on_mismatchBoolean</td>
<td></td>
</tr>
<tr>
<td>Symbols File</td>
<td>Specify the name of a symbols file to link.</td>
</tr>
<tr>
<td>arm_linker_symbols_filesFileName</td>
<td></td>
</tr>
<tr>
<td>Treat Linker Warnings as Errors</td>
<td>Treat linker warnings as errors.</td>
</tr>
<tr>
<td>arm_linker_treat_warnings_as_errorsBoolean</td>
<td></td>
</tr>
<tr>
<td>Use Manual Linker Script</td>
<td>Specifies whether to use a manual linker script.</td>
</tr>
<tr>
<td>link_use_linker_script_fileBoolean</td>
<td></td>
</tr>
</tbody>
</table>

**Package**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package Dependencies</td>
<td>Specifies the packages the current project depends upon.</td>
</tr>
<tr>
<td>package_dependenciesStringList</td>
<td></td>
</tr>
</tbody>
</table>

**Preprocessor**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ignore Includes</td>
<td>Ignore the include directories properties.</td>
</tr>
<tr>
<td>c_ignore_includesBoolean</td>
<td></td>
</tr>
<tr>
<td>Preprocessor Definitions</td>
<td>Specifies one or more preprocessor definitions. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>c_preprocessor_definitionsStringList</td>
<td></td>
</tr>
</tbody>
</table>
Preprocessor Undefinitions
\textit{c\_preprocessor\_unndefinitions}\texttt{StringList}
Specifies one or more preprocessor undefinitions. This property will have macro expansion applied to it.

System Include Directories
\textit{c\_system\_include\_directories}\texttt{StringList}
Specifies the system include path. This property will have macro expansion applied to it.

Undefine All Preprocessor Definitions
\textit{c\_undefine\_all\_preprocessor\_definitions}\texttt{Boolean}
Does not define any standard preprocessor definitions.

User Include Directories
\textit{c\_user\_include\_directories}\texttt{StringList}
Specifies the user include path. This property will have macro expansion applied to it.

### Printf/Scanf

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{linker_printf_fp_enabled}\texttt{Enumeration}</td>
<td>Are floating point numbers supported by the printf function group.</td>
</tr>
<tr>
<td>\texttt{linker_printf_fmt_level}\texttt{Enumeration}</td>
<td>The largest integer type supported by the printf function group.</td>
</tr>
<tr>
<td>\texttt{linker_printf_width_precision_supported}\texttt{Boolean}</td>
<td>Enables support for width and precision specification in the printf function group.</td>
</tr>
<tr>
<td>\texttt{linker_scanf_character_group_matching_enabled}\texttt{Boolean}</td>
<td>Enables support for %[...] and %[^...] character class matching in the scanf functions.</td>
</tr>
<tr>
<td>\texttt{linker_scanf_fp_enabled}\texttt{Boolean}</td>
<td>Are floating point numbers supported by the scanf function group.</td>
</tr>
<tr>
<td>\texttt{linker_scanf_fmt_level}\texttt{Enumeration}</td>
<td>The largest integer type supported by the scanf function group.</td>
</tr>
<tr>
<td>\texttt{linker_printf_wchar_enabled}\texttt{Boolean}</td>
<td>Are wide characters supported by the printf function group.</td>
</tr>
</tbody>
</table>

### Project

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{project_flag}\texttt{Enumeration}</td>
<td>Flag which you can use to draw attention to important projects in your solution.</td>
</tr>
</tbody>
</table>

### Runtime Memory Area

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{arm_linker_heap_size}\texttt{IntegerRange}</td>
<td>The size of the heap in bytes. The preprocessor define _HEAP_SIZE_ is set to this value.</td>
</tr>
</tbody>
</table>
### Main Stack Size

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>arm_linker_stack_size</code></td>
<td>IntegerRange</td>
</tr>
<tr>
<td></td>
<td>The size of the main stack in bytes.</td>
</tr>
</tbody>
</table>

### Process Stack Size

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>arm_linker_process_stack_size</code></td>
<td>IntegerRange</td>
</tr>
<tr>
<td></td>
<td>The size of the process stack in bytes.</td>
</tr>
</tbody>
</table>

### Stack Size (Abort Mode)

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>arm_linker_abt_stack_size</code></td>
<td>IntegerRange</td>
</tr>
<tr>
<td></td>
<td>The size of the Abort mode stack in bytes.</td>
</tr>
</tbody>
</table>

### Stack Size (FIQ Mode)

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>arm_linker_fiq_stack_size</code></td>
<td>IntegerRange</td>
</tr>
<tr>
<td></td>
<td>The size of the FIQ mode stack in bytes.</td>
</tr>
</tbody>
</table>

### Stack Size (IRQ Mode)

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>arm_linker_irq_stack_size</code></td>
<td>IntegerRange</td>
</tr>
<tr>
<td></td>
<td>The size of the IRQ mode stack in bytes.</td>
</tr>
</tbody>
</table>

### Stack Size (Supervisor Mode)

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>arm_linker_svc_stack_size</code></td>
<td>IntegerRange</td>
</tr>
<tr>
<td></td>
<td>The size of the Supervisor mode stack in bytes.</td>
</tr>
</tbody>
</table>

### Stack Size (Undefined Mode)

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>arm_linker_und_stack_size</code></td>
<td>IntegerRange</td>
</tr>
<tr>
<td></td>
<td>The size of the Undefined mode stack in bytes.</td>
</tr>
</tbody>
</table>

### Section

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Section Name</td>
<td><code>default_code_section</code></td>
</tr>
<tr>
<td></td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Specifies the default name to use for the program code section.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant Section Name</td>
<td><code>default_const_section</code></td>
</tr>
<tr>
<td></td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Specifies the default name to use for the read-only constant section.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Section Name</td>
<td><code>default_data_section</code></td>
</tr>
<tr>
<td></td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Specifies the default name to use for the initialized, writable data section.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISR Section Name</td>
<td><code>default_isr_section</code></td>
</tr>
<tr>
<td></td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Specifies the default name to use for the ISR code.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector Section Name</td>
<td><code>default_vector_section</code></td>
</tr>
<tr>
<td></td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Specifies the default name to use for the interrupt vector section.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zeroed Section Name</td>
<td><code>default_zeroed_section</code></td>
</tr>
<tr>
<td></td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Specifies the default name to use for the zero-initialized, writable data section.</td>
</tr>
</tbody>
</table>

### Solution

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag</td>
<td><code>solution_flag</code></td>
</tr>
<tr>
<td></td>
<td>Enumeration</td>
</tr>
<tr>
<td></td>
<td>Flag which you can use to draw attention to important projects in your solution.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties Filter</td>
<td><code>properties_filter</code></td>
</tr>
<tr>
<td></td>
<td>StringList</td>
</tr>
<tr>
<td></td>
<td>The names of project properties that can be displayed at the solution.</td>
</tr>
</tbody>
</table>
## Source Code

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional Code Completion Compiler Options</td>
<td>Additional source indexing and code completion compiler options.</td>
</tr>
<tr>
<td>code_completion_options</td>
<td>StringList</td>
</tr>
<tr>
<td>Inhibit Source Indexing</td>
<td>Disable source indexing and code completion for files/folders/projects that would normally be indexed (C/C++ files in executable and library projects).</td>
</tr>
<tr>
<td>project_inhibit_indexing</td>
<td>Boolean</td>
</tr>
<tr>
<td>Source Code Control Directory</td>
<td>Source code control directory root.</td>
</tr>
<tr>
<td>source_code_control_directory</td>
<td>DirPath</td>
</tr>
</tbody>
</table>

## Staging

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output File Path</td>
<td>The output file path the stage command will create. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>stage_output_filepath</td>
<td>String</td>
</tr>
<tr>
<td>Set To Read-only</td>
<td>Set the output file permissions to read only or read/write.</td>
</tr>
<tr>
<td>stage_set_readonly</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Stage Command</td>
<td>The command to execute. This property will have macro expansion applied to it with the additional $(StageOutputFilePath) macro set to the output filepath of the stage command.</td>
</tr>
<tr>
<td>stage_command</td>
<td>Unknown</td>
</tr>
<tr>
<td>Stage Command Working Directory</td>
<td>The working directory in which the stage command is run. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>stage_command_wd</td>
<td>String</td>
</tr>
<tr>
<td>Stage Project Command</td>
<td>The command to execute after staging commands have executed. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>stage_post_build_command</td>
<td>Unknown</td>
</tr>
<tr>
<td>Stage Project Command Working Directory</td>
<td>The working directory where the post build command runs. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>stage_post_build_command_wd</td>
<td>String</td>
</tr>
</tbody>
</table>

## User Build Step

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link Patch Command</td>
<td>A command to run after the link but prior to additional binary file generation. This property will have macro expansion applied to it with the additional $(TargetPath) macro set to the output filepath of the linker command.</td>
</tr>
<tr>
<td>linker_patch_build_command</td>
<td>Unknown</td>
</tr>
<tr>
<td>Property Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Link Patch Working Directory</td>
<td>The working directory where the link patch command is run. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>linker_patch_build_command_wdDirPath</td>
<td></td>
</tr>
<tr>
<td>Post-Archive Command</td>
<td>A command to run after the archive command has completed. This property will have macro expansion applied to it with the additional $(TargetPath) macro set to the output filepath of the archive command.</td>
</tr>
<tr>
<td>archive_post_build_command</td>
<td>Unknown</td>
</tr>
<tr>
<td>Post-Archive Working Directory</td>
<td>The working directory where the post-archive command is run. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>archive_post_build_command_wdDirPath</td>
<td></td>
</tr>
<tr>
<td>Post-Build Command</td>
<td>The command to execute after a project build. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>post_build_command</td>
<td>Unknown</td>
</tr>
<tr>
<td>Post-Build Command Control</td>
<td>Controls when the post-build command is run, either Always Run or when Run When Build Has Occurred.</td>
</tr>
<tr>
<td>post_build_command_control</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Post-Build Command Working Directory</td>
<td>The working directory in which the post-build command is run. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>post_build_command_wdString</td>
<td></td>
</tr>
<tr>
<td>Post-Compile Command</td>
<td>A command to run after the compile command has completed. This property will have macro expansion applied to it with the additional $(TargetPath) macro set to the output filepath of the compiler command.</td>
</tr>
<tr>
<td>compile_post_build_command</td>
<td>Unknown</td>
</tr>
<tr>
<td>Post-Compile Working Directory</td>
<td>The working directory where the post-compile command is run. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>compile_post_build_command_wdDirPath</td>
<td></td>
</tr>
<tr>
<td>Post-Link Command</td>
<td>A command to run after the link command has completed. This property will have macro expansion applied to it with the additional $(TargetPath) macro set to the output filepath of the linker command and $(PostLinkOutputFilePath) set to the value of the output filepath of the post link command.</td>
</tr>
<tr>
<td>linker_post_build_command</td>
<td>Unknown</td>
</tr>
<tr>
<td>Post-Link Output File</td>
<td>The name of the file created by the post-link command. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>linker_post_build_command_output_file</td>
<td></td>
</tr>
<tr>
<td>Post-Link Working Directory</td>
<td>The working directory where the post-link command is run. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>linker_post_build_command_wdDirPath</td>
<td></td>
</tr>
<tr>
<td>Pre-Build Command</td>
<td>The command to execute before a project build. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>pre_build_command</td>
<td>Unknown</td>
</tr>
<tr>
<td>Pre-Build Command Control</td>
<td>Controls when the pre-build command is run, either Always Run or when Run When Build Required.</td>
</tr>
<tr>
<td>pre_build_command_control</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Pre-Build Command Working Directory</td>
<td>The working directory in which the pre-build command is run. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>pre_build_command_wdString</td>
<td></td>
</tr>
<tr>
<td>Pre-Compile Command</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>compile_pre_build_command</td>
<td>A command to run before the compile command. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>Unknown</td>
<td></td>
</tr>
<tr>
<td>Pre-Compile Command Output File Path</td>
<td>The pre-compile generated file name. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>compile_pre_build_command_output_file_name</td>
<td></td>
</tr>
<tr>
<td>Pre-Compile Working Directory</td>
<td>The working directory where the pre-compile command is run. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>compile_pre_build_command_wd</td>
<td></td>
</tr>
<tr>
<td>linker_pre_build_command</td>
<td>A command to run before the link command. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>linker_pre_build_command_wd_dir_path</td>
<td></td>
</tr>
<tr>
<td>Pre-Link Working Directory</td>
<td>The working directory where the pre-link command is run. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>linker_pre_build_command_wd_dir_path</td>
<td></td>
</tr>
</tbody>
</table>
## Debug Options

### Debugger

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Arguments</td>
<td>The command arguments passed to the executable. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td><code>debug_command_arguments</code> String</td>
<td></td>
</tr>
<tr>
<td>DABORT Handler Name</td>
<td>The name of the dabort handler symbol. Used for backtracing out of exception handlers.</td>
</tr>
<tr>
<td><code>dabortHandler_name</code> String</td>
<td></td>
</tr>
<tr>
<td>Debug Additional Projects</td>
<td>The debugger will load (if not already loaded by Load Additional Projects) and debug the specified additional projects.</td>
</tr>
<tr>
<td><code>debug_dependent_projects</code> StringList</td>
<td></td>
</tr>
<tr>
<td>Debug Symbols File[0]</td>
<td>The name of the debug symbols file. This property will have macro expansion applied to it. If it is not defined then the main load file is used.</td>
</tr>
<tr>
<td><code>external_debug_symbols_file_name</code> ProjFileName</td>
<td></td>
</tr>
<tr>
<td>Debug Symbols File[1]</td>
<td>The name of the debug symbols file. This property will have macro expansion applied to it. If it is not defined then the main load file is used.</td>
</tr>
<tr>
<td><code>external_debug_symbols_file_name1</code> ProjFileName</td>
<td></td>
</tr>
<tr>
<td>Debug Symbols File[2]</td>
<td>The name of the debug symbols file. This property will have macro expansion applied to it. If it is not defined then the main load file is used.</td>
</tr>
<tr>
<td><code>external_debug_symbols_file_name2</code> ProjFileName</td>
<td></td>
</tr>
<tr>
<td>Debug Symbols File[3]</td>
<td>The name of the debug symbols file. This property will have macro expansion applied to it. If it is not defined then the main load file is used.</td>
</tr>
<tr>
<td><code>external_debug_symbols_file_name3</code> ProjFileName</td>
<td></td>
</tr>
<tr>
<td>Debug Symbols Load Address[0]</td>
<td>The (code) address to be added to the debug symbol (code) addresses.</td>
</tr>
<tr>
<td><code>external_debug_symbols_load_address</code> String</td>
<td></td>
</tr>
<tr>
<td>Debug Symbols Load Address[1]</td>
<td>The (code) address to be added to the debug symbol (code) addresses.</td>
</tr>
<tr>
<td><code>external_debug_symbols_load_address1</code> String</td>
<td></td>
</tr>
<tr>
<td>Debug Symbols Load Address[2]</td>
<td>The (code) address to be added to the debug symbol (code) addresses.</td>
</tr>
<tr>
<td><code>external_debug_symbols_load_address2</code> String</td>
<td></td>
</tr>
<tr>
<td>Debug Symbols Load Address[3]</td>
<td>The (code) address to be added to the debug symbol (code) addresses.</td>
</tr>
<tr>
<td><code>external_debug_symbols_load_address3</code> String</td>
<td></td>
</tr>
<tr>
<td>Debug Terminal Log File</td>
<td>A file to write the output from the debug terminal to.</td>
</tr>
<tr>
<td><code>debug_terminal_log_file</code> Unknown</td>
<td></td>
</tr>
<tr>
<td>Default debugIO implementation</td>
<td>The default debugIO implementation.</td>
</tr>
<tr>
<td><code>arm_debugIO_Implementation</code> Enumeration</td>
<td></td>
</tr>
<tr>
<td>Display DCC data</td>
<td>The debugger will display data that is written to the DCC when debugIO is not used.</td>
</tr>
<tr>
<td><code>arm_display_DCC</code> Boolean</td>
<td></td>
</tr>
<tr>
<td>Entry Point Symbol</td>
<td>Debugger will start execution at symbol if defined.</td>
</tr>
<tr>
<td><code>debug_entry_point_symbol</code> String</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FIQ Handler Name</td>
<td>The name of the fiq handler symbol. Used for backtracing out of exception handlers.</td>
</tr>
<tr>
<td>IRQ Handler Name</td>
<td>The name of the irq handler symbol. Used for backtracing out of exception handlers.</td>
</tr>
<tr>
<td>Ignore .debug_aranges Section</td>
<td>The debugger will not use the .debug_aranges section.</td>
</tr>
<tr>
<td>Ignore .debug_frame Section</td>
<td>The debugger will not use the .debug_frame section.</td>
</tr>
<tr>
<td>Load Additional Projects</td>
<td>The debugger will load the outputs of the specified additional projects.</td>
</tr>
<tr>
<td>Load Offset</td>
<td>The offset to add to the load address of the load file. This offset is added to any absolute relocations of symbols (whose address is less than Load Offset Symbol Limit) if the load file contains relocation sections.</td>
</tr>
<tr>
<td>Load Offset Symbol Limit</td>
<td>If set apply the Load Offset logic to only those symbols that have addresses less than the specified limit.</td>
</tr>
<tr>
<td>Memory Upload Page Size</td>
<td>The aligned page size the debugger uses when uploading address ranges.</td>
</tr>
<tr>
<td>PABORT Handler Name</td>
<td>The name of the pabort handler symbol. Used for backtracing out of exception handlers.</td>
</tr>
<tr>
<td>RTT Control Block Address</td>
<td>The symbol or 0x prefixed address of the RTT control block.</td>
</tr>
<tr>
<td>RTT Enable</td>
<td>If enabled the debugger will service RTT input/output in the debug terminal.</td>
</tr>
<tr>
<td>Register Definition File</td>
<td>The name of the file containing register definitions.</td>
</tr>
<tr>
<td>Reserved Member Name</td>
<td>The struct reserved member name. Struct members that contain the (case insensitive) string will not be displayed.</td>
</tr>
<tr>
<td>Restrict Memory Access</td>
<td>If enabled the debugger will only display variables located in the address ranges of the memory map or the sections in the elf file.</td>
</tr>
<tr>
<td>Run To</td>
<td>The initial breakpoint to set</td>
</tr>
<tr>
<td>Run To Control</td>
<td>Specify when the initial breakpoint should be set</td>
</tr>
<tr>
<td>SWI Handler Name</td>
<td>The name of the swi handler symbol. Used for backtracing out of exception handlers.</td>
</tr>
<tr>
<td>Start Address</td>
<td>The address to start the externally built executable running from.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Start From Entry Point Symbol</strong></td>
<td>If yes the debugger will start execution from the entry point symbol. If no the debugger will start execution from the core specific location.</td>
</tr>
<tr>
<td><code>debug_start_from_entry_point_symbol</code></td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Starting Stack Pointer Value</strong></td>
<td>The symbol or 0x prefixed value to set the stack pointer on start debugging.</td>
</tr>
<tr>
<td><code>debug_stack_pointer_start</code></td>
<td>String</td>
</tr>
<tr>
<td><strong>Startup Completion Point</strong></td>
<td>Specifies the point in the program where startup is complete. Software breakpoints and debugIO will be enabled after this point has been reached.</td>
</tr>
<tr>
<td><code>debug_startup_completion_point</code></td>
<td>String</td>
</tr>
<tr>
<td><strong>Target Connection</strong></td>
<td>Specifies the target to connect to for debugging actions.</td>
</tr>
<tr>
<td><code>debug_target_connection</code></td>
<td>Enumeration</td>
</tr>
<tr>
<td><strong>Target Device</strong></td>
<td>The name of the device to connect to. The macro <code>$(Target)</code> is substituted with the Target Processor project property value.</td>
</tr>
<tr>
<td><code>arm_target_device_name</code></td>
<td>String</td>
</tr>
<tr>
<td><strong>Thread Maximum</strong></td>
<td>The maximum number of threads to display.</td>
</tr>
<tr>
<td><code>debug_threads_max</code></td>
<td>IntegerRange</td>
</tr>
<tr>
<td><strong>Threads Script File</strong></td>
<td>The threads script used by the debugger.</td>
</tr>
<tr>
<td><code>debug_threads_script</code></td>
<td>ProjFileName</td>
</tr>
<tr>
<td><strong>Type Interpretation File</strong></td>
<td>Specifies the type interpretation file to use.</td>
</tr>
<tr>
<td><code>debug_type_file</code></td>
<td>FileName</td>
</tr>
<tr>
<td><strong>UNDEF handler name</strong></td>
<td>The name of the undef handler symbol. Used for backtracing out of exception handlers.</td>
</tr>
<tr>
<td><code>undefHandler_name</code></td>
<td>String</td>
</tr>
<tr>
<td><strong>Working Directory</strong></td>
<td>The working directory for a debug session. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td><code>debug_working_directory</code></td>
<td>DirPath</td>
</tr>
</tbody>
</table>

**GDB Server**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Allow Memory Access During Execution</strong></td>
<td>Specifies whether memory can be access while target is running. If set to No, target will be stopped each time memory is accessed.</td>
</tr>
<tr>
<td><code>gdb_server_allow_memory_access_during_execution</code></td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Auto Start GDB Server</strong></td>
<td>Specifies whether a GDB server should be started on connect.</td>
</tr>
<tr>
<td><code>gdb_server_autostart_server</code></td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Connect Timeout</strong></td>
<td>The length of time in seconds to attempt to connect to server before failing.</td>
</tr>
<tr>
<td><code>gdb_server_connect_timeout</code></td>
<td>Integer</td>
</tr>
<tr>
<td><strong>GDB Server Command Line</strong></td>
<td>The command line to start the gdb server</td>
</tr>
<tr>
<td><code>gdb_server_command_line</code></td>
<td>Unknown</td>
</tr>
<tr>
<td><strong>Host</strong></td>
<td>The hostname of the GDB server to connect to.</td>
</tr>
<tr>
<td><code>gdb_server_hostname</code></td>
<td>String</td>
</tr>
<tr>
<td><strong>Ignore Checksum Errors</strong></td>
<td>Specifies whether an incorrect GDB server checksum causes and error.</td>
</tr>
<tr>
<td><code>gdb_server_ignore_checksum_errors</code></td>
<td>Boolean</td>
</tr>
</tbody>
</table>
## Log File

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdb_server_log_fileFileName</td>
<td>Specifies a file to output a log of GDB server transactions to.</td>
</tr>
</tbody>
</table>

## Port

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdb_server_port:Integer</td>
<td>The port number to use to connect to the GDB server.</td>
</tr>
</tbody>
</table>

## Read Timeout

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdb_server_read_timeout:Integer</td>
<td>The length of time in seconds to attempt to read from server before failing.</td>
</tr>
</tbody>
</table>

## Register Access

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdb_server_register_access:Enumeration</td>
<td>Specifies how registers are accessed</td>
</tr>
</tbody>
</table>

## Reset and Stop Command

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdb_server_reset_command:String</td>
<td>The remote GDB server command to use to reset and stop the target.</td>
</tr>
</tbody>
</table>

## Target XML File

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdb_server_target_xml_file:FileName</td>
<td>If specified, points to a target.xml file to use in place of file returned by the GDB server.</td>
</tr>
</tbody>
</table>

## Type

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdb_server_type:Enumeration</td>
<td>Specifies the type of GDB server being connected to. J-Link, OpenOCD, ST-LINK and pyOCD gdb server implementations are currently supported.</td>
</tr>
</tbody>
</table>

## Write Timeout

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdb_server_write_timeout:Integer</td>
<td>The length of time in seconds to attempt to write to server before failing.</td>
</tr>
</tbody>
</table>

## J-Link

### Property

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional J-Link Options:JLinkExecuteCommand:StringLengthList</td>
<td>Specify additional J-Link options to allow enabling or disabling advanced features and fine tuning. For more information see J-Link Command Strings</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Adaptive Clocking:adaptive:Enumeration</td>
<td>Adaptive clocking is enabled.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exclude Flash Cache Range:JLinkExcludeFlashCacheRange:String</td>
<td>Define a memory range that should not be cached by J-Link. Per default, all areas that J-Link knows to be Flash memory, are cached. This means that it is assumed that the contents of this areas do not change during program execution. If this assumption does not hold true, typically because the target program modifies the flash content for data storage, then the affected area should be excluded from the cache. This may slightly reduce the debugging speed. Syntax: either 'start_address-end_address' or 'address,size'. For example: 0x08000000,0x1000.</td>
</tr>
</tbody>
</table>
### Host Connection

**Connection Enumeration**

Defines how to connect the host to the J-Link:

- "USB": Connect to J-Link via USB
- "USB S/N": Connect to J-Link with specified serial number via USB e.g. USB 174300001
- "IP S/N": Connect to J-Link with specified serial number via IP e.g. IP 174300001
- "IP n.n.n.n port": Connect to J-Link with specified IP address and optional port number e.g. IP 192.168.20.20 19020

### JTAG Instruction Register Size Before Target

**arm_linker_jtag_pad_post_ir IntegerRange**

Specifies the number of bits in the instruction register before the target (as seen from TDI), which is the number of bits to pad the JTAG instruction register with the BYPASS instruction after the target instruction.

### JTAG Number Of Devices Before Target

**arm_linker_jtag_pad_post_dr IntegerRange**

Specifies the number of devices before the target (as seen from TDI), which is the number of bits to pad the JTAG data register.

### Log File

**JLinkLogFileName FileName**

The file to output the J-Link log to.

### Script File

**JLinkScriptFileName FileName**

The file path of the optional J-Link script file to use.

### Show Log Messages In Output Window

**showLog Enumeration**

Display the J-Link log messages to the output window.

### Speed

**speed IntegerRange**

The required JTAG/SWD clock frequency in kHz (0 to auto-detect best possible).

### Supply Power

**supplyPower Enumeration**

The J-Link supplies power to the target.

### Target Interface Type

**arm_target_interface_type Enumeration**

Specifies the type of interface the target has. The options are:

- JTAG - Use JTAG interface
- SWD - Use SWD interface
- cJTAG - Use cJTAG interface

## Loader

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional Load File Address[0]</td>
<td>The address to load the additional load file.</td>
</tr>
<tr>
<td>debug_additional_load_file_addressString</td>
<td></td>
</tr>
<tr>
<td>Additional Load File Address[1]</td>
<td>The address to load the additional load file.</td>
</tr>
<tr>
<td>debug_additional_load_file_address1String</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Additional Load File Address[2]</td>
<td>The address to load the additional load file.</td>
</tr>
<tr>
<td>debug_additional_load_file_address2String</td>
<td></td>
</tr>
<tr>
<td>Additional Load File Address[3]</td>
<td>The address to load the additional load file.</td>
</tr>
<tr>
<td>debug_additional_load_file_address3String</td>
<td></td>
</tr>
<tr>
<td>Additional Load File Type[0]</td>
<td>The file type of the additional load file. The options are Detect, elf, bin, ihex, hex, tihex, srec.</td>
</tr>
<tr>
<td>debug_additional_load_file_type Enumeration</td>
<td></td>
</tr>
<tr>
<td>Additional Load File Type[1]</td>
<td>The file type of the additional load file. The options are Detect, elf, bin, ihex, hex, tihex, srec.</td>
</tr>
<tr>
<td>debug_additional_load_file_type1 Enumeration</td>
<td></td>
</tr>
<tr>
<td>Additional Load File Type[2]</td>
<td>The file type of the additional load file. The options are Detect, elf, bin, ihex, hex, tihex, srec.</td>
</tr>
<tr>
<td>debug_additional_load_file_type2 Enumeration</td>
<td></td>
</tr>
<tr>
<td>Additional Load File Type[3]</td>
<td>The file type of the additional load file. The options are Detect, elf, bin, ihex, hex, tihex, srec.</td>
</tr>
<tr>
<td>debug_additional_load_file_type3 Enumeration</td>
<td></td>
</tr>
<tr>
<td>Additional Load File[0]</td>
<td>Additional file to load on debug load. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>debug_additional_load_fileProjFileName</td>
<td></td>
</tr>
<tr>
<td>Additional Load File[1]</td>
<td>Additional file to load on debug load. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>debug_additional_load_file1ProjFileName</td>
<td></td>
</tr>
<tr>
<td>Additional Load File[2]</td>
<td>Additional file to load on debug load. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>debug_additional_load_file2ProjFileName</td>
<td></td>
</tr>
<tr>
<td>Additional Load File[3]</td>
<td>Additional file to load on debug load. This property will have macro expansion applied to it.</td>
</tr>
<tr>
<td>debug_additional_load_file3ProjFileName</td>
<td></td>
</tr>
<tr>
<td>Check Load Sections Fit Target Description</td>
<td>Specifies whether load sections in the program match the memory segments described in the memory map.</td>
</tr>
<tr>
<td>target_check_load_sections_fitBoolean</td>
<td></td>
</tr>
<tr>
<td>Load ELF Sections</td>
<td>The debugger will load ELF sections rather than ELF programs.</td>
</tr>
<tr>
<td>debug_load_sectionsEnumeration</td>
<td></td>
</tr>
<tr>
<td>Load File</td>
<td>The name of the main load file. This property will have macro expansion applied to it. If it is not defined then the output filepath of the linker command is used.</td>
</tr>
<tr>
<td>external_build_file_nameProjFileName</td>
<td></td>
</tr>
<tr>
<td>Load File Address</td>
<td>The address to download the main load file to.</td>
</tr>
<tr>
<td>external_load_addressString</td>
<td></td>
</tr>
<tr>
<td>Load File Type</td>
<td>The file type of the main load file. The options are Detect, elf, bin, ihex, hex, tihex, srec.</td>
</tr>
<tr>
<td>external_load_file_typeEnumeration</td>
<td></td>
</tr>
<tr>
<td>No Load Sections</td>
<td>Names of (loadable) sections not to load.</td>
</tr>
<tr>
<td>target_loader_no_load_sectionsStringList</td>
<td></td>
</tr>
</tbody>
</table>

## Simulator

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Simulation File</td>
<td>Specifies the dll that simulates the memory system.</td>
</tr>
<tr>
<td>arm_simulator_memory_simulation_filenamePrc</td>
<td>This property will have macro expansion applied to it.</td>
</tr>
</tbody>
</table>
### Memory Simulation Parameter

**arm_simulator_memory_simulation_parameter**

Parameter passed to the memory simulation. The format of this is specific to the memory simulation. The default memory simulation takes a list of **RX|RWX 'hex start address', 'hex size in bytes', 'default hex word value'** for example RX 00000000, 10000000, FFFFFFFF;RWX 10000000, 10000000, CDDCDDCD

---

### Memory Simulation Parameter Macros

**arm_simulator_memory_simulation_parameter_macros**

Macros to apply to the parameter passed to the memory simulation on creation.

---

### Stop On Branch

**arm_simulator_stop_on_branch_dot**

Boolean stop when the simulator executes a b . instruction.

---

### Stop On Memory Error

**arm_simulator_stop_on_read_write**

Enumeration specifies the simulator behaviour when a memory error occurs.

---

### Trace Buffer Size

**arm_simulator_num_trace_entries**

Integer the number of trace entries to store.

---

## Target Script

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attach Script</td>
<td>The script that is executed when the target is attached to.</td>
</tr>
<tr>
<td>target_attach_scriptJavaScript</td>
<td></td>
</tr>
<tr>
<td>Connect Script</td>
<td>The script that is executed when the target is connected to.</td>
</tr>
<tr>
<td>target_connect_scriptJavaScript</td>
<td></td>
</tr>
<tr>
<td>Debug Begin Script</td>
<td>The script that is executed when the debugger begins a debug session.</td>
</tr>
<tr>
<td>target_debug_begin_scriptJavaScript</td>
<td></td>
</tr>
<tr>
<td>Debug End Script</td>
<td>The script that is executed when the debugger ends a debug session.</td>
</tr>
<tr>
<td>target_debug_end_scriptJavaScript</td>
<td></td>
</tr>
<tr>
<td>Disconnect Script</td>
<td>The script that is executed when the target is disconnected from.</td>
</tr>
<tr>
<td>target_disconnect_scriptJavaScript</td>
<td></td>
</tr>
<tr>
<td>Load Begin Script</td>
<td>The script that is executed when the debugger begins a load.</td>
</tr>
<tr>
<td>target_load_begin_scriptJavaScript</td>
<td></td>
</tr>
<tr>
<td>Load End Script</td>
<td>The script that is executed when the debugger ends a load.</td>
</tr>
<tr>
<td>target_load_end_scriptJavaScript</td>
<td></td>
</tr>
<tr>
<td>Reset Script</td>
<td>The script that is executed when the target is reset.</td>
</tr>
<tr>
<td>target_reset_scriptJavaScript</td>
<td></td>
</tr>
<tr>
<td>Target Script File</td>
<td>The target script file, the contents of this file are prepended to script project properties before they are executed.</td>
</tr>
<tr>
<td>target_script_fileFileName</td>
<td></td>
</tr>
</tbody>
</table>

---

## Target Trace

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ITM Stimulus Port To Display</strong></td>
<td>Specifies the ITM Stimulus port to display in the debug terminal -1 disables this.</td>
</tr>
<tr>
<td>arm_target_itm_stimulus_port_display</td>
<td>IntegerRange</td>
</tr>
<tr>
<td><strong>ITM Stimulus Ports Enable</strong></td>
<td>Specifies the ITM Stimulus ports to enable.</td>
</tr>
<tr>
<td>arm_target_itm_stimulus_port_enable</td>
<td>IntegerHex</td>
</tr>
<tr>
<td><strong>ITM Stimulus Ports Privilege</strong></td>
<td>Specifies the ITM Stimulus ports to enable.</td>
</tr>
<tr>
<td>arm_target_itm_stimulus_port_privilege</td>
<td>IntegerHex</td>
</tr>
<tr>
<td><strong>ITM Timestamping</strong></td>
<td>Specifies ITM timestamping. The options are:</td>
</tr>
<tr>
<td>arm_target_itm_timestamping_enable</td>
<td>Enumeration</td>
</tr>
<tr>
<td><strong>ITM/DWT Data Trace PC</strong></td>
<td>Specifies whether to trace the PC on data trace.</td>
</tr>
<tr>
<td>arm_target_dwt_data_trace_PC</td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>ITM/DWT PC Sampling</strong></td>
<td>Specifies the DWT PC sampling rate.</td>
</tr>
<tr>
<td>arm_target_dwt_PC_sampling_enable</td>
<td>Enumeration</td>
</tr>
<tr>
<td><strong>ITM/DWT Trace Exceptions</strong></td>
<td>Specifies whether to trace exception entry and return.</td>
</tr>
<tr>
<td>arm_target_dwt_trace_exceptions</td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>MTB RAM Address</strong></td>
<td>Specifies the MTB RAM Address - note that this must be aligned to the MTB RAM size.</td>
</tr>
<tr>
<td>arm_target_mtb_ram_address</td>
<td>IntegerHex</td>
</tr>
<tr>
<td><strong>MTB RAM Size</strong></td>
<td>Specifies the MTB RAM size in bytes.</td>
</tr>
<tr>
<td>arm_target_mtb_ram_size</td>
<td>Enumeration</td>
</tr>
<tr>
<td><strong>SWO Baud Rate</strong></td>
<td>Specifies the baud rate of the SWO - zero selects auto detection.</td>
</tr>
<tr>
<td>arm_target_trace_SWO_speed</td>
<td>IntegerRange</td>
</tr>
<tr>
<td><strong>Trace Clock Speed</strong></td>
<td>The speed of the trace clock. This is usually the same as the CPU clock and is used to program the prescaler for the SWO.</td>
</tr>
<tr>
<td>arm_target_trace_clock_speed</td>
<td>IntegerRange</td>
</tr>
<tr>
<td><strong>Trace Initialize Script</strong></td>
<td>The script that is executed to initialize the target trace hardware. When executed this script has the macro $(TraceInterfaceType) expanded with value of the Trace Interface Type property, typically it is EnableTrace(&quot;$(TraceInterfaceType)&quot;).</td>
</tr>
<tr>
<td>target_trace_initialize_script</td>
<td>JavaScript</td>
</tr>
<tr>
<td><strong>Trace Interface Type</strong></td>
<td>Specifies the type of trace interface the target has. The options are:</td>
</tr>
<tr>
<td>arm_target_trace_interface_type</td>
<td>Enumeration</td>
</tr>
<tr>
<td></td>
<td>SWO - Use asynchronous SWO trace interface.</td>
</tr>
<tr>
<td></td>
<td>TracePort - Use synchronous parallel trace interface.</td>
</tr>
<tr>
<td></td>
<td>ETB - Use on-chip embedded trace buffer.</td>
</tr>
<tr>
<td></td>
<td>MTB - Use on-chip MTB - Cortex-M0+ only.</td>
</tr>
<tr>
<td></td>
<td>PC Sampling - sample the PC.</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Trace Port Size</td>
<td>Specifies the trace port size the target has. The options are:</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| arm_target_trace_port_size | 1-bit
|                         | 2-bit
|                         | 4-bit
|                         | 8-bit
|                         | 16-bit
|                         | 24-bit
|                         | 32-bit
## System Macros

### System Macro Values

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(Date) String</td>
<td>Day Month Year e.g. 21 June 2011.</td>
</tr>
<tr>
<td>$(DateDay) String</td>
<td>Day e.g. 21.</td>
</tr>
<tr>
<td>$(DateMonth) String</td>
<td>Month e.g. June.</td>
</tr>
<tr>
<td>$(DateYear) String</td>
<td>Year e.g. 2011.</td>
</tr>
<tr>
<td>$(DesktopDir) String</td>
<td>Path to users desktop directory.</td>
</tr>
<tr>
<td>$(DocumentsDir) String</td>
<td>Path to users documents directory.</td>
</tr>
<tr>
<td>$(EmbeddedStudioVersionNumber) String</td>
<td>The release version number.</td>
</tr>
<tr>
<td>$(HomeDir) String</td>
<td>Path to users home directory.</td>
</tr>
<tr>
<td>$(HostArch) String</td>
<td>The CPU architecture that SEGGER Embedded Studio is running on e.g. x86.</td>
</tr>
<tr>
<td>$(HostDLL) String</td>
<td>The file extension for dynamic link libraries on the CPU that SEGGER Embedded Studio is running on e.g. .dll.</td>
</tr>
<tr>
<td>$(HostDLLExt) String</td>
<td>The file extension for dynamic link libraries used by the operating system that SEGGER Embedded Studio is running on e.g. .dll, .so, .dylib.</td>
</tr>
<tr>
<td>$(HostEXE) String</td>
<td>The file extension for executables on the CPU that SEGGER Embedded Studio is running on e.g. .exe.</td>
</tr>
<tr>
<td>$(HostOS) String</td>
<td>The name of the operating system that SEGGER Embedded Studio is running on e.g. win.</td>
</tr>
<tr>
<td>$(JLinkDir) String</td>
<td>Path to J-Link software.</td>
</tr>
<tr>
<td>$(Micro) String</td>
<td>The SEGGER Embedded Studio target e.g. ARM.</td>
</tr>
<tr>
<td>$(PackagesDir) String</td>
<td>Path to the users packages directory.</td>
</tr>
<tr>
<td>$(Platform) String</td>
<td>The target platform.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>$(ProductNameShort)</td>
<td>The product name.</td>
</tr>
<tr>
<td>$(StudioArchiveFileExt)</td>
<td>The filename extension of a studio archive file.</td>
</tr>
<tr>
<td>$(StudioBuildToolExeName)</td>
<td>The filename of the build tool executable.</td>
</tr>
<tr>
<td>$(StudioBuildToolName)</td>
<td>The name of the build tool executable.</td>
</tr>
<tr>
<td>$(StudioDir)</td>
<td>The install directory of the product.</td>
</tr>
<tr>
<td>$(StudioExeName)</td>
<td>The filename of the studio executable.</td>
</tr>
<tr>
<td>$(StudioMajorVersion)</td>
<td>The major release version of software.</td>
</tr>
<tr>
<td>$(StudioMinorVersion)</td>
<td>The minor release version of software.</td>
</tr>
<tr>
<td>$(StudioName)</td>
<td>The full name of studio.</td>
</tr>
<tr>
<td>$(StudioNameShort)</td>
<td>The short name of studio.</td>
</tr>
<tr>
<td>$(StudioPackageFileExt)</td>
<td>The filename extension of a studio package file.</td>
</tr>
<tr>
<td>$(StudioProjectFileExt)</td>
<td>The filename extension of a studio project file.</td>
</tr>
<tr>
<td>$(StudioScriptToolExeName)</td>
<td>The filename of the script tool executable.</td>
</tr>
<tr>
<td>$(StudioScriptToolName)</td>
<td>The name of the script tool executable.</td>
</tr>
<tr>
<td>$(StudioSessionFileExt)</td>
<td>The filename extension of a studio session file.</td>
</tr>
<tr>
<td>$(StudioSimulatorExeName)</td>
<td>The filename of the simulator executable.</td>
</tr>
<tr>
<td>$(StudioSimulatorName)</td>
<td>The name of the simulator executable.</td>
</tr>
<tr>
<td>$(StudioUserDir)</td>
<td>The directory containing the user data.</td>
</tr>
<tr>
<td>$(TargetID)</td>
<td>ID number representing the SEGGER Embedded Studio target.</td>
</tr>
<tr>
<td>$(Time)</td>
<td>Hour:Minutes:Seconds e.g. 15:34:03.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>$(TimeHour) $ (TimeHour) String</td>
<td>Hour e.g. 15.</td>
</tr>
<tr>
<td>$(TimeMinute) $ (TimeMinute) String</td>
<td>Minute e.g. 34.</td>
</tr>
<tr>
<td>$(TimeSecond) $ (TimeSecond) String</td>
<td>Seconds e.g. 03.</td>
</tr>
<tr>
<td>$(UnixTime) $ (UnixTime) String</td>
<td>Seconds since 00:00, Jan 1 1970 UTC</td>
</tr>
</tbody>
</table>
## Build Macros

### (Build Macro Values)

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(Arch)</td>
<td>The lower case value of the ARM Architecture project property.</td>
</tr>
<tr>
<td>$(AsmOptions)</td>
<td>A space separated list of assembler options for the external assemble command.</td>
</tr>
<tr>
<td>$(COnlyOptions)</td>
<td>A space separated list of compiler options for the external c compile command.</td>
</tr>
<tr>
<td>$(COptions)</td>
<td>A space separated list of compiler options for the external c and c++ compile commands.</td>
</tr>
<tr>
<td>$(CombiningOutputFilePath)</td>
<td>The full path of the output file of the combining command.</td>
</tr>
<tr>
<td>$(CombiningRelInputPaths)</td>
<td>The relative inputs to the combining command.</td>
</tr>
<tr>
<td>$(Configuration)</td>
<td>The build configuration e.g. ARM Flash Debug.</td>
</tr>
<tr>
<td>$(CoreType)</td>
<td>The lower case value of the ARM Core Type project property.</td>
</tr>
<tr>
<td>$(Defines)</td>
<td>The preprocessor defines property value for the external compile command.</td>
</tr>
<tr>
<td>$(DependencyPath)</td>
<td>The path of the dependency file for the external compile command.</td>
</tr>
<tr>
<td>$(EXE)</td>
<td>The default file extension for an executable file including the dot e.g. .elf.</td>
</tr>
<tr>
<td>$(Endian)</td>
<td>The lower case value of the Byte Order project property.</td>
</tr>
<tr>
<td>$(FPABI)</td>
<td>The value of the ARM FP ABI Type project property.</td>
</tr>
<tr>
<td>$(FPU)</td>
<td>The lower case value of the ARM FPU Type project property.</td>
</tr>
<tr>
<td>$(FPU2)</td>
<td>Alternative value of the ARM FPU Type project property.</td>
</tr>
<tr>
<td>$(FPU3)</td>
<td>Alternative value of the ARM FPU Type project property.</td>
</tr>
<tr>
<td>$(FolderName)</td>
<td>The folder name of the containing folder.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>$(FolderPath)$</td>
<td>The folder path of the containing folders.</td>
</tr>
<tr>
<td>$(GCCPrefix)$</td>
<td>The macro-expanded value of the <strong>GCC Prefix</strong> project property.</td>
</tr>
<tr>
<td>$(GCCTarget)$</td>
<td>The value of the <strong>GCC Target</strong> project property.</td>
</tr>
<tr>
<td>$(Includes)$</td>
<td>The user includes property value for the external compile command.</td>
</tr>
<tr>
<td>$(InputDir)$</td>
<td>The absolute directory of the input file.</td>
</tr>
<tr>
<td>$(InputExt)$</td>
<td>The extension of an input file not including the dot e.g. cpp.</td>
</tr>
<tr>
<td>$(InputFileName)$</td>
<td>The name of an input file relative to the project directory.</td>
</tr>
<tr>
<td>$(InputName)$</td>
<td>The name of an input file relative to the project directory without the extension.</td>
</tr>
<tr>
<td>$(InputPath)$</td>
<td>The absolute name of an input file including the extension.</td>
</tr>
<tr>
<td>$(IntDir)$</td>
<td>The macro-expanded value of the <strong>Intermediate Directory</strong> project property.</td>
</tr>
<tr>
<td>$(LIB)$</td>
<td>The default file extension for a library file including the dot e.g. .lib.</td>
</tr>
<tr>
<td>$(LibExt)$</td>
<td>The architecture and build specific library extension.</td>
</tr>
<tr>
<td>$(LinkOptions)$</td>
<td>A space seperated list of compiler options for the external link command.</td>
</tr>
<tr>
<td>$(LinkerScriptPath)$</td>
<td>The full path of the linker script file for the link command.</td>
</tr>
<tr>
<td>$(MapPath)$</td>
<td>The full path of the map file of the external link command.</td>
</tr>
<tr>
<td>$(OBJ)$</td>
<td>The default file extension for an object file including the dot e.g. .o.</td>
</tr>
<tr>
<td>$(Objects)$</td>
<td>A space seperated list of files for the external archive or link command.</td>
</tr>
<tr>
<td>$(ObjectsFilePath)$</td>
<td>The full path containing the files for the external archive or link command.</td>
</tr>
<tr>
<td>$(OutDir)$</td>
<td>The macro-expanded value of the <strong>Output Directory</strong> project property.</td>
</tr>
<tr>
<td>$(PackageExt)$</td>
<td>The file extension of a package file e.g. emPackage.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>$(PostLinkOutputFilePath)</td>
<td>The full path of the output file of the post link command.</td>
</tr>
<tr>
<td>$(ProjectDir)</td>
<td>The absolute value of the <strong>Project Directory</strong> project property of the current project. If this isn't set then the directory containing the solution file.</td>
</tr>
<tr>
<td>$(ProjectName)</td>
<td>The project name of the current project.</td>
</tr>
<tr>
<td>$(ProjectNodeName)</td>
<td>The name of the selected project node.</td>
</tr>
<tr>
<td>$(RelDependencyPath)</td>
<td>The relative path of the dependency file for the external compile command.</td>
</tr>
<tr>
<td>$(RelInputDir)</td>
<td>The relative path to the directory containing the input file from the project directory.</td>
</tr>
<tr>
<td>$(RelInputPath)</td>
<td>The relative path to the input file from the project directory.</td>
</tr>
<tr>
<td>$(RelLinkerScriptPath)</td>
<td>The relative path of the linker script file for the link command.</td>
</tr>
<tr>
<td>$(RelMapPath)</td>
<td>The relative path of the map file of the external link command.</td>
</tr>
<tr>
<td>$(RelObjectsFilePath)</td>
<td>The relative path containing the files for the external archive or link command.</td>
</tr>
<tr>
<td>$(RelTargetPath)</td>
<td>The project directory relative path of the output file of the link or compile command.</td>
</tr>
<tr>
<td>$(RootOutDir)</td>
<td>The macro-expanded value of the <strong>Root Output Directory</strong> project property.</td>
</tr>
<tr>
<td>$(RootRelativeOutDir)</td>
<td>The relative path to get from the path specified by the <strong>Output Directory</strong> project property to the path specified by the <strong>Root Output Directory</strong> project property.</td>
</tr>
<tr>
<td>$(SolutionDir)</td>
<td>The absolute path of the directory containing the solution file.</td>
</tr>
<tr>
<td>$(SolutionExt)</td>
<td>The extension of the solution file without the dot.</td>
</tr>
<tr>
<td>$(SolutionFileName)</td>
<td>The filename of the solution file.</td>
</tr>
<tr>
<td>$(SolutionName)</td>
<td>The basename of the solution file.</td>
</tr>
<tr>
<td>$(SolutionPath)</td>
<td>The absolute path of the solution file.</td>
</tr>
<tr>
<td>$(StageOutputFilePath)</td>
<td>The full path of the output file of the stage command.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>$(TargetPath)</code></td>
<td>The full path of the output file of the link or compile command.</td>
</tr>
<tr>
<td><code>$(ToolChainDir)</code></td>
<td>The macro-expanded value of the Tool Chain Directory project property.</td>
</tr>
<tr>
<td><code>$(Undefines)</code></td>
<td>The preprocessor undefines property value for the external compile command.</td>
</tr>
</tbody>
</table>
**BinaryFile**

The following table lists the BinaryFile object's member functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BinaryFile.crc32(offset, length)</code></td>
<td>returns the CRC-32 checksum of an address range <code>length</code> bytes long, starting at <code>offset</code>. This function computes a CRC-32 checksum on a block of data using the standard CRC-32 polynomial (0x04C11DB7) with an initial value of 0xFFFFFFFF. Note that this implementation doesn't reflect the input or the output and the result is inverted.</td>
</tr>
<tr>
<td><code>BinaryFile.length()</code></td>
<td>returns the length of the binary file in bytes.</td>
</tr>
<tr>
<td><code>BinaryFile.load(path)</code></td>
<td>loads binary file from <code>path</code>.</td>
</tr>
<tr>
<td><code>BinaryFile.peekBytes(offset, length)</code></td>
<td>returns byte array containing <code>length</code> bytes peeked from <code>offset</code>.</td>
</tr>
<tr>
<td><code>BinaryFile.peekUint32(offset, littleEndian)</code></td>
<td>returns a 32-bit word peeked from <code>offset</code>. The <code>littleEndian</code> argument specifies the endianness of the access, if true or undefined it will be little endian, otherwise it will be big endian.</td>
</tr>
<tr>
<td><code>BinaryFile.pokeBytes(offset, byteArray)</code></td>
<td>poke byte array <code>byteArray</code> to <code>offset</code>.</td>
</tr>
<tr>
<td><code>BinaryFile.pokeUint32(offset, value, littleEndian)</code></td>
<td>poke a <code>value</code> to 32-bit word located at <code>offset</code>. The <code>littleEndian</code> argument specifies the endianness of the access, if true or undefined it will be little endian, otherwise it will be big endian.</td>
</tr>
<tr>
<td><code>BinaryFile.resize(length, fill)</code></td>
<td>resizes the binary image to <code>length</code> bytes. If the operation extends the size, the binary image will be padded with bytes of value <code>fill</code>.</td>
</tr>
<tr>
<td><code>BinaryFile.save(path)</code></td>
<td>saves binary file to <code>path</code>.</td>
</tr>
<tr>
<td><code>BinaryFile.saveRange(path, offset, length)</code></td>
<td>saves part of the binary file to <code>path</code>. The <code>offset</code> argument specifies the byte offset to start from. The <code>length</code> argument specifies the maximum number of bytes that should be saved.</td>
</tr>
</tbody>
</table>
CWSys

The following table lists the CWSys object's member functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWSys appendStringToFile(path, string)</td>
<td>appends string to the end of the file path.</td>
</tr>
<tr>
<td>CWSys.copyFile(srcPath, destPath)</td>
<td>copies file srcPath to destPath.</td>
</tr>
<tr>
<td>CWSys.crc32(array)</td>
<td>returns the CRC-32 checksum of the byte array array. This function computes a CRC-32 checksum on a block of data using the standard CRC-32 polynomial (0x04C11DB7) with an initial value of 0xFFFFFFFF. Note that this implementation doesn't reflect the input or the output and the result is inverted.</td>
</tr>
<tr>
<td>CWSys.fileExists(path)</td>
<td>returns true if file path exists.</td>
</tr>
<tr>
<td>CWSys.fileSize(path)</td>
<td>return the number of bytes in file path.</td>
</tr>
<tr>
<td>CWSys.getRunStderr()</td>
<td>returns the stderr output from the last CWSys.run() call.</td>
</tr>
<tr>
<td>CWSys.getRunStdout()</td>
<td>returns the stdout output from the last CWSys.run() call.</td>
</tr>
<tr>
<td>CWSys.makeDirectory(path)</td>
<td>create the directory path.</td>
</tr>
<tr>
<td>CWSys.packU32(array, offset, number, le)</td>
<td>packs number into the array at offset.</td>
</tr>
<tr>
<td>CWSys.popup(text, caption)</td>
<td>prompt the user with text and return true for yes and false for no.</td>
</tr>
<tr>
<td>CWSys.readByteArrayFromFile(path)</td>
<td>returns the byte array contained in the file path.</td>
</tr>
<tr>
<td>CWSys.readStringFromFile(path)</td>
<td>returns the string contained in the file path.</td>
</tr>
<tr>
<td>CWSys.removeDirectory(path)</td>
<td>remove the directory path.</td>
</tr>
<tr>
<td>CWSys.removeFile(path)</td>
<td>deletes file path.</td>
</tr>
<tr>
<td>CWSys.renameFile(oldPath, newPath)</td>
<td>renames file oldPath to be newPath.</td>
</tr>
<tr>
<td>CWSys.run(cmd, wait)</td>
<td>runs command line cmd optionally waits for it to complete if wait is true.</td>
</tr>
<tr>
<td>CWSys.unpackU32(array, offset, le)</td>
<td>returns the number unpacked from the array at offset.</td>
</tr>
<tr>
<td>CWSys.writeByteArrayToFile(path, array)</td>
<td>creates a file path containing the byte array array.</td>
</tr>
<tr>
<td>CWSys.writeStringToFile(path, string)</td>
<td>creates a file path containing string.</td>
</tr>
</tbody>
</table>
Debug

The following table lists the Debug object's member functions.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug.evaluate(expression)</td>
<td>Evaluates debug expression and returns it as a JavaScript value.</td>
</tr>
<tr>
<td>Debug.getfunction(address)</td>
<td>Returns function name containing address.</td>
</tr>
</tbody>
</table>
ElfFile

The following table lists the ElfFile object’s member functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ElfFile.crc32(address, length, virtualNotPhysical, padding, programNotSection)</td>
<td>returns the CRC-32 checksum of an address range length bytes long, located at address. If virtualNotPhysical is true or undefined, address is a virtual address otherwise it is a physical address. If padding is defined, it specifies the byte value used to fill gaps in the program. If programNotSection is true or undefined, data is read using program headers rather than section headers. This function computes a CRC-32 checksum on a block of data using the standard CRC-32 polynomial (0x04C11DB7) with an initial value of 0xFFFFFFFF. Note that this implementation doesn’t reflect the input or the output and the result is inverted.</td>
</tr>
<tr>
<td>ElfFile.findProgram(address)</td>
<td>returns an object with start, the data and the size to allocate of the Elf program that contains address.</td>
</tr>
<tr>
<td>ElfFile.getEntryPoint()</td>
<td>returns the entry point in the ELF file.</td>
</tr>
<tr>
<td>ElfFile.getSection(name)</td>
<td>returns an object with start and the data of the Elf section corresponding to the name.</td>
</tr>
<tr>
<td>ElfFile.isLittleEndian()</td>
<td>returns true if the Elf file has numbers encoded as little endian.</td>
</tr>
<tr>
<td>ElfFile.load(path)</td>
<td>loads Elf file from path.</td>
</tr>
<tr>
<td>ElfFile.peekBytes(address, length, virtualNotPhysical, padding, programNotSection)</td>
<td>returns byte array containing length bytes peeked from address. If virtualNotPhysical is true or undefined, address is a virtual address otherwise it is a physical address. If padding is defined, it specifies the byte value used to fill gaps in the program. If programNotSection is true or undefined, data is read using program headers rather than section headers.</td>
</tr>
<tr>
<td>ElfFile.peekUint32(address, virtualNotPhysical)</td>
<td>returns a 32-bit word peeked from address. If virtualNotPhysical is true or undefined, address is a virtual address otherwise it is a physical address.</td>
</tr>
<tr>
<td>ElfFile.pokeBytes(address, byteArray, virtualNotPhysical)</td>
<td>poke byte array byteArray to address. If virtualNotPhysical is true or undefined, address is a virtual address otherwise it is a physical address.</td>
</tr>
<tr>
<td>ElfFile.pokeUint32(address, value, virtualNotPhysical)</td>
<td>poke a value to 32-bit word located at address. If virtualNotPhysical is true or undefined, address is a virtual address otherwise it is a physical address.</td>
</tr>
<tr>
<td>ElfFile.save(path)</td>
<td>saves Elf file to path.</td>
</tr>
<tr>
<td>ElfFile.symbolValue(symbol)</td>
<td>returns the value of symbol in Elf file.</td>
</tr>
</tbody>
</table>
# TargetInterface

The following table lists the TargetInterface object’s member functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TargetInterface.crc32(address, length)</code></td>
<td>reads a block of bytes from target memory starting at address for length bytes, generates a crc32 on the block of bytes and returns it.</td>
</tr>
<tr>
<td><code>TargetInterface.delay(ms)</code></td>
<td>waits for ms milliseconds</td>
</tr>
<tr>
<td><code>TargetInterface.error(message)</code></td>
<td>terminates execution of the script and outputs a target interface error message to the target log.</td>
</tr>
<tr>
<td><code>TargetInterface.executeFunction(address, parameter, timeout)</code></td>
<td>calls a function at address with the parameter and returns the function result. The timeout is in milliseconds.</td>
</tr>
<tr>
<td><code>TargetInterface.expandMacro(string)</code></td>
<td>returns the string with macros expanded.</td>
</tr>
<tr>
<td><code>TargetInterface.findByte(address, length, byte)</code></td>
<td>returns the index of the byte in the specified target memory range.</td>
</tr>
<tr>
<td><code>TargetInterface.findNotByte(address, length, byte)</code></td>
<td>returns the index of the byte that isn't in the specified target memory range.</td>
</tr>
<tr>
<td><code>TargetInterface.getProjectProperty(savename)</code></td>
<td>returns the value of the savename project property.</td>
</tr>
<tr>
<td><code>TargetInterface.getTargetProperty(savename)</code></td>
<td>returns the value of the savename target property.</td>
</tr>
<tr>
<td><code>TargetInterface.go()</code></td>
<td>allows the target to run.</td>
</tr>
<tr>
<td><code>TargetInterface.isStopped()</code></td>
<td>returns true if the target is stopped.</td>
</tr>
<tr>
<td><code>TargetInterface.message(message)</code></td>
<td>outputs a target interface message to the target log.</td>
</tr>
<tr>
<td><code>TargetInterface.peekBinary(address, length, filename)</code></td>
<td>reads a block of bytes from target memory starting at address for length bytes and writes them to filename.</td>
</tr>
<tr>
<td><code>TargetInterface.peekByte(address)</code></td>
<td>reads a byte of target memory from address and returns it.</td>
</tr>
<tr>
<td><code>TargetInterface.peekBytes(address, length)</code></td>
<td>reads a block of bytes from target memory starting at address for length bytes and returns the result as an array containing the bytes read.</td>
</tr>
<tr>
<td><code>TargetInterface.peekMultUint16(address, length)</code></td>
<td>reads length unsigned 16-bit integers from target memory starting at address and returns them as an array.</td>
</tr>
<tr>
<td><code>TargetInterface.peekMultUint32(address, length)</code></td>
<td>reads length unsigned 32-bit integers from target memory starting at address and returns them as an array.</td>
</tr>
<tr>
<td><code>TargetInterface.peek Uint16(address)</code></td>
<td>reads a 16-bit unsigned integer from target memory from address and returns it.</td>
</tr>
<tr>
<td><code>TargetInterface.peek Uint32(address)</code></td>
<td>reads a 32-bit unsigned integer from target memory from address and returns it.</td>
</tr>
<tr>
<td><code>TargetInterface.peekWord(address)</code></td>
<td>reads a word as an unsigned integer from target memory from address and returns it.</td>
</tr>
<tr>
<td><code>TargetInterface.pokeBinary(address, filename)</code></td>
<td>reads a block of bytes from filename and writes them to target memory starting at address.</td>
</tr>
<tr>
<td><code>TargetInterface.pokeByte(address, data)</code></td>
<td>writes the byte data to address in target memory.</td>
</tr>
</tbody>
</table>
**TargetInterface.pokeBytes(address, data)** writes the array data containing 8-bit data to target memory at address.

**TargetInterface.pokeMultUint16(address, data)** writes the array data containing 16-bit data to target memory at address.

**TargetInterface.pokeMultUint32(address, data)** writes the array data containing 32-bit data to target memory at address.

**TargetInterface.pokeUint16(address, data)** writes data as a 16-bit value to address in target memory.

**TargetInterface.pokeUint32(address, data)** writes data as a 32-bit value to address in target memory.

**TargetInterface.pokeWord(address, data)** writes data as a word value to address in target memory.

**TargetInterface.readBinary(filename)** reads a block of bytes from filename and returns them in an array.

**TargetInterface.reset()** resets the target.

**TargetInterface.resetAndStop()** resets and stops the target.

**TargetInterface.runFromAddress(address, timeout)** start the target executing at address and waits for a breakpoint to be hit. The timeout is in milliseconds.

**TargetInterface.runFromToAddress(from, to, timeout)** start the target executing at address from and waits for the breakpoint to be hit. The timeout is in milliseconds.

**TargetInterface.runToAddress(address, timeout)** sets a breakpoint at address, starts the target executing and waits for the breakpoint to be hit. The timeout is in milliseconds.

**TargetInterface.setTargetProperty(savename)** set the value of the savename target property.

**TargetInterface.stop()** stops the target.

**TargetInterface.writeBinary(array, filename)** write the bytes in array to filename.
## WScript

The following table lists the WScript object’s member functions.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WScript.Echo(s)</strong></td>
<td>echos string s to the output terminal.</td>
</tr>
</tbody>
</table>