Migration guide

Migrating from the Renesas High-performance Embedded Workshop and e2studio toolchains for RX to IAR Embedded Workbench® for RX

Use this guide as a guideline when converting project files and source code written for Renesas toolchains for RX to IAR Embedded Workbench® for RX.

	Product	Version number
Migrating from	Renesas HEW or e2studio for RX (CCRX / HEW / e2studio)	V.1.x / 4.x / 2.x
Migrating to	IAR Embedded Workbench for RX (EWRX)	V.2.42 and newer

Migration overview

Migrating an existing project from Renesas toolchain for RX requires that you collect information about your current project and then apply this information to the new IAR EWRX project. In addition, you need to make some changes in the actual source code. The information in this document is intended to simplify this process.

Note: If you are new to using IAR Embedded Workbench, we suggest that you first look at the user guides and tutorials which you can find in the IAR Information Center.

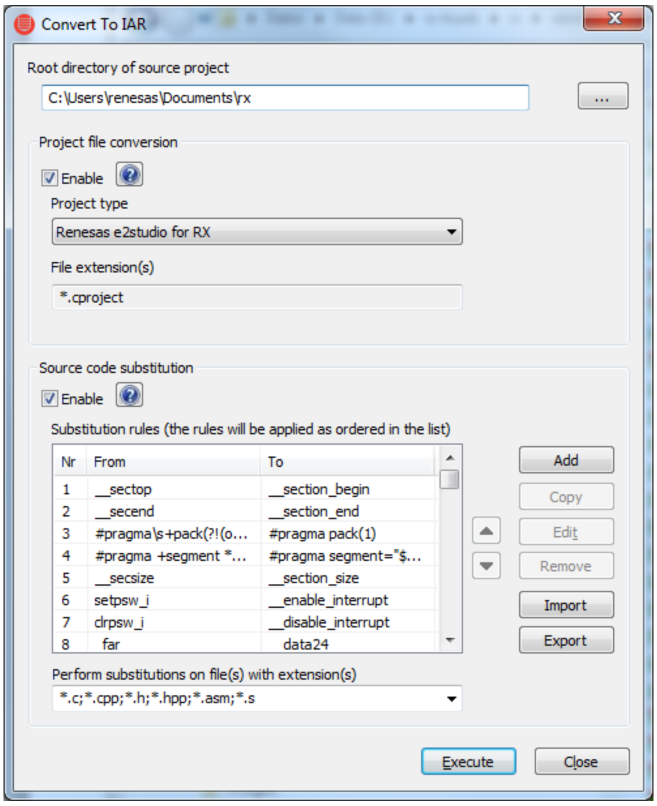
Project conversion

To migrate existing Renesas HEW or e2studio applications to IAR EWRX there is a tool called **Convert To IAR**. This is a GUI application included with IAR Embedded Workbench, available via the **Tools** menu.

The **Convert To IAR** tool converts HEW as well as Renesas e2studio project files into EWRX project files without changing the original project file. Information about source files, include paths, defined symbols and build configuration is transferred. As an option, also source code text substitutions are performed and you can add your own substitution rules including support for regular expressions.

Procedure

1. Start EWRX.
2. Start **Convert To IAR** available in the **Tools** menu.
3. Navigate to the HEW or e2studio project to convert by clicking the browse button.
4. Click the **Execute** button and a new EWRX project file will be created.
5. Add the new project to a EWRX workspace by choosing **Project>Add Existing Project…**.
6. Set the relevant project options by choosing **Project>Options…**.
   Hint: Open the original project in HEW/e2studio, walk through the options and set the corresponding options in EWRX as suggested in the section *Important tool settings* below.



Root directory of source project: C:\Users\renesas\Documents\rx

Project file conversion
[✓] Enable
Project type: Renesas e2studio for RX
File extension(s): *.cproject

Source code substitution
[✓] Enable
Substitution rules (the rules will be applied as ordered in the list)

Nr	From	To
1	__sectop	__section_begin
2	__secend	__section_end
3	#pragma\s+pack(?!(o…	#pragma pack(1)
4	#pragma +segment *…	#pragma segment="$…
5	__secsize	__section_size
6	setpsw_i	__enable_interrupt
7	clrpsw_i	__disable_interrupt
8	far	data24

Perform substitutions on file(s) with extension(s): *.c;*.cpp;*.h;*.hpp;*.asm;*.s

Basic code differences

This table shows some of the basic differences between code written for HEW/e2studio and EWRX that you need to handle before building your converted project.

Renesas HEW/e2studio	IAR Embedded Workbench
Initialization code	

The following files contain startup code that you normally don't need to port as the functionality is covered by the EWRX `cstartup.s` or in the linker configuration files:
• `resetprg.c` Startup code
• `dbsct.c` Data initialization
• `sbrk.c` Configures the MCU heap memory
• `intprg.c` Empty interrupt handler functions
• `vecttbl.c` Vector table initialization
• `id_code.c` MCU ID code handling
 	
Startup code that you normally shall port:
• `HardwareSetup()` Customized HW initialization
	`cstartup.s`
	System startup executed after reset. Data and segment initialization. Part of the runtime library but can be overridden by including this assembler file in your project. You find the file in sub folder `rx\src\lib\rx`.
	
	`int __low_level_init(void);`
	Called from `cstartup.s` before initializing segments and calling `main()`. You may include your own version of this routine in you project. Suitable for HW initialization. This function shall return `1` for the data sections to be initialized. Otherwise, `0`.

SFR I/O files	
The SFR header file is created by the HEW/e2studio project generator: Naming: `iodefine.h`	One file per device family located in sub folder `rx/inc` Naming: `io<device family>.h` Example: `iorx63n.h`
Interrupt declarations	

`#pragma interrupt func_name (interrupt specification)`
 	
Example:
`#pragma interrupt _timer_a0(vect=12)`
`void _timer_a0(void)`
`{`
`}`
	`#pragma vector=<OFFSET>`
	`__interrupt [__nested] void func_name (void)`
	`{`
	`}`
	Example:
	`#pragma vector=TIMER_A0`
	`__interrupt __nested void _timer_a0 (void)`
	`{`
	`}`
	(The vector offset symbol is declared in the SFR header file)

Building your project

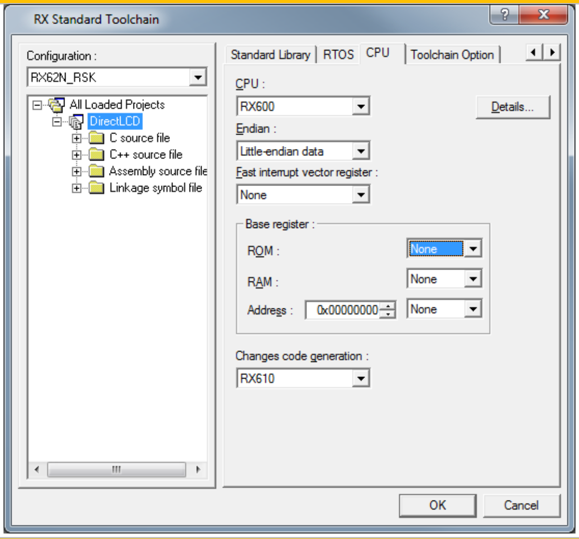
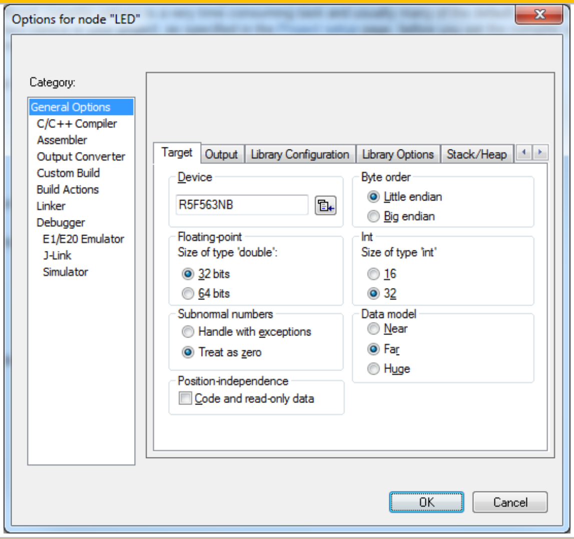
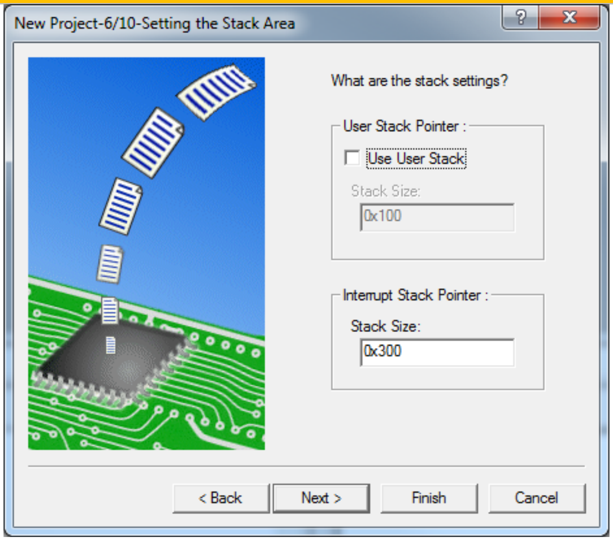
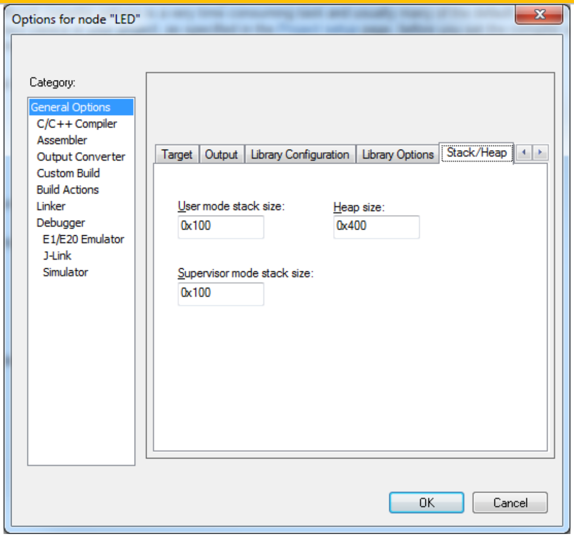
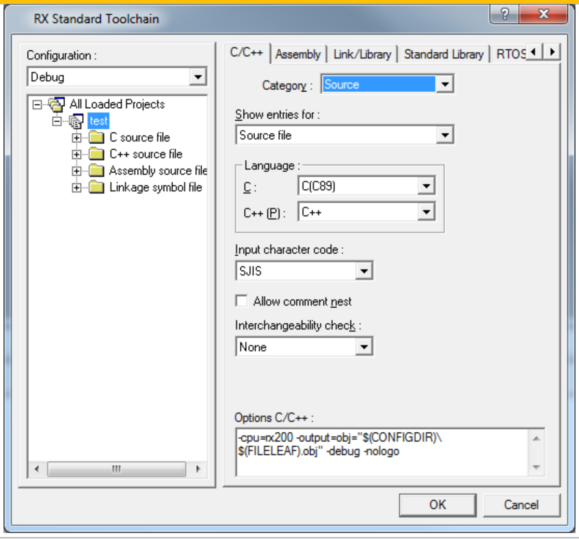
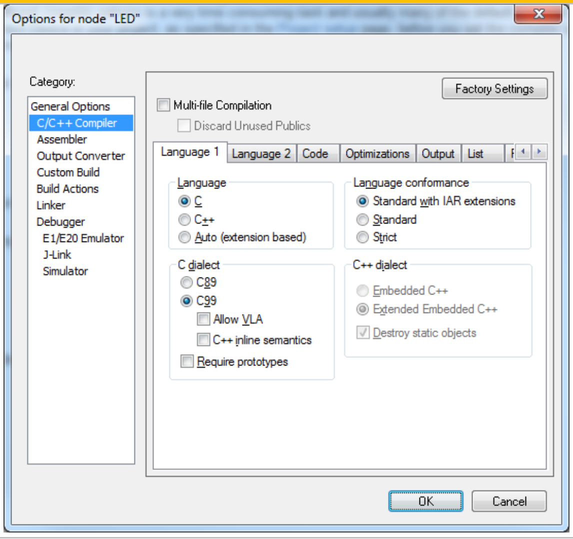
After successfully converting the Renesas project and considered the basic code differences described above, you will still most likely need to fine-tune parts of the source code so that it follows the EWRX syntax.

1. Select your device under **Project>Options>General Options**.
2. Choose **Project>Make**.
3. To find the different errors/warnings, press **F4** (Next Error/Tag).
   This will bring you to the location in the source code that generated this error/warning.
4. For each error/warning, modify the source code to match the EWRX syntax.
   Note: See the **Reference information** section below for this step.
5. After correcting one or more errors/warnings, repeat the procedure.

Note: It is always a good idea to correct the first couple of errors/warnings in different source files first.
This is because errors and warnings later in the source code might just be effects of faulty syntax at the beginning of the source.

Important tool settings

This is an overview of the most important tool settings. Make sure that they match your original HEW project.

Renesas HEW	IAR Embedded Workbench
Device selection and Byte-order	
	
Stack/Heap size	
 (Setup in project wizard which generates `stacksct.h`)	 (Sets symbols used in the linker configuration file (`.icf`)
Language settings	
	

RX Standard Toolchain

Configuration : RX62N_RSK

All Loaded Projects / DirectLCD / C source file / C++ source file / Assembly source file / Linkage symbol file

Standard Library | RTOS | CPU | Toolchain Option

CPU : RX600 — Details...

Endian : Little-endian data

Fast interrupt vector register : None

Base register : ROM : None; RAM : None; Address : 0x00000000 None

Changes code generation : RX610

OK Cancel

Options for node "LED"

Category: General Options, C/C++ Compiler, Assembler, Output Converter, Custom Build, Build Actions, Linker, Debugger, E1/E20 Emulator, J-Link, Simulator

Target | Output | Library Configuration | Library Options | Stack/Heap

Device: R5F563NB

Byte order: (●) Little endian ( ) Big endian

Floating-point — Size of type 'double': (●) 32 bits ( ) 64 bits

Int — Size of type 'int': ( ) 16 (●) 32

Subnormal numbers: ( ) Handle with exceptions (●) Treat as zero

Data model: ( ) Near (●) Far ( ) Huge

Position-independence: ☐ Code and read-only data

OK Cancel

New Project-6/10-Setting the Stack Area

What are the stack settings?

User Stack Pointer : ☐ Use User Stack — Stack Size: 0x100

Interrupt Stack Pointer : Stack Size: 0x300

< Back | Next > | Finish | Cancel

Options for node "LED"

Category: General Options, C/C++ Compiler, Assembler, Output Converter, Custom Build, Build Actions, Linker, Debugger, E1/E20 Emulator, J-Link, Simulator

Target | Output | Library Configuration | Library Options | Stack/Heap

User mode stack size: 0x100

Heap size: 0x400

Supervisor mode stack size: 0x100

OK Cancel

RX Standard Toolchain

Configuration : Debug

All Loaded Projects / test / C source file / C++ source file / Assembly source file / Linkage symbol file

C/C++ | Assembly | Link/Library | Standard Library | RTOS

Category : Source

Show entries for : Source file

Language : C : C(C89); C++ (P) : C++

Input character code : SJIS

☐ Allow comment nest

Interchangeability check : None

Options C/C++ :
```
-cpu=rx200 -output=obj="$(CONFIGDIR)\
$(FILELEAF).obj" -debug -nologo
```

OK Cancel

Options for node "LED"

Category: General Options, C/C++ Compiler, Assembler, Output Converter, Custom Build, Build Actions, Linker, Debugger, E1/E20 Emulator, J-Link, Simulator

Factory Settings

☐ Multi-file Compilation

☐ Discard Unused Publics

Language 1 | Language 2 | Code | Optimizations | Output | List

Language: (●) C ( ) C++ ( ) Auto (extension based)

Language conformance: (●) Standard with IAR extensions ( ) Standard ( ) Strict

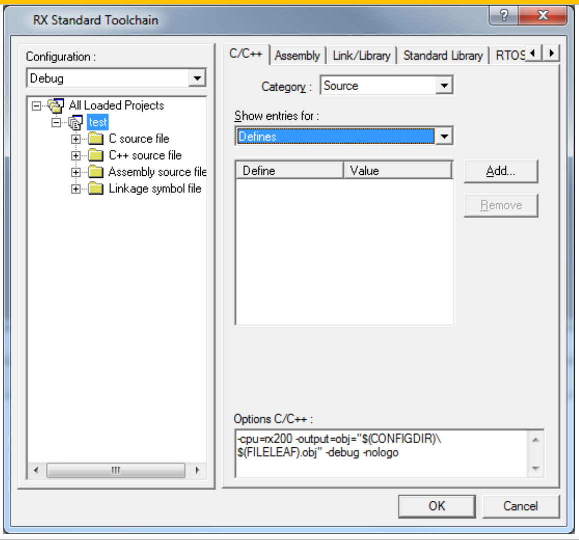
C dialect: ( ) C89 (●) C99 ☐ Allow VLA ☐ C++ inline semantics

☐ Require prototypes

C++ dialect: ( ) Embedded C++ (●) Extended Embedded C++ ☑ Destroy static objects

OK Cancel

Defined symbols and include directories



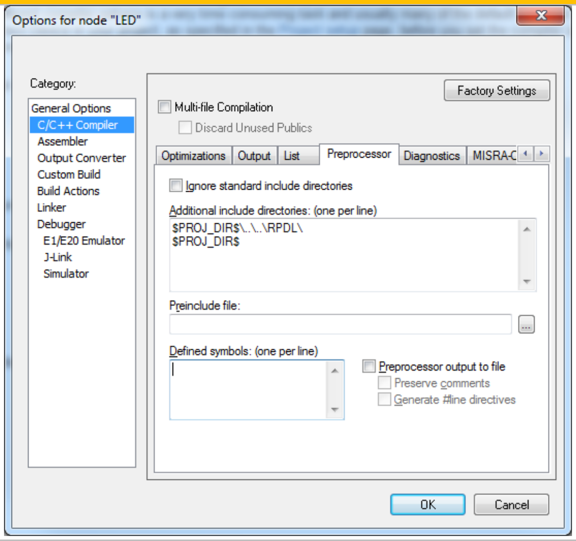
Configuration : Debug

Category : Source

Show entries for : Defines

Options C/C++ :

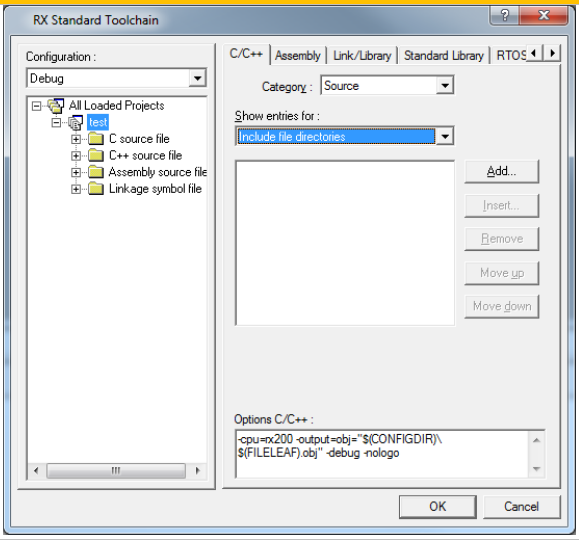
```
-cpu=rx200 -output=obj="$(CONFIGDIR)\
$(FILELEAF).obj" -debug -nologo
```



Additional include directories: (one per line)

```
$PROJ_DIR$\..\..\RPDL\
$PROJ_DIR$
```

Include directories



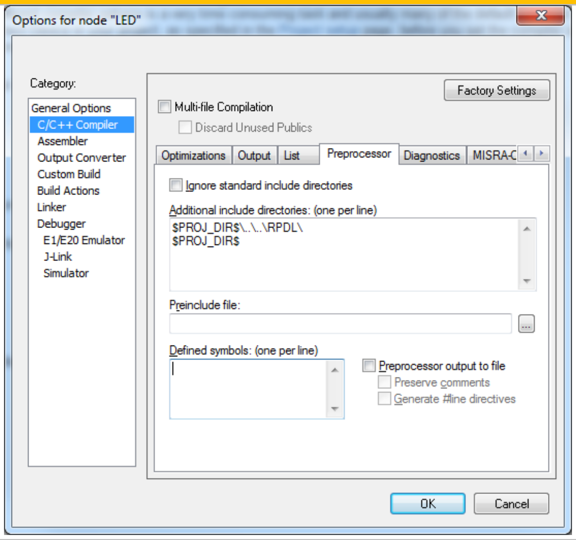
Configuration : Debug

Category : Source

Show entries for : Include file directories

Options C/C++ :

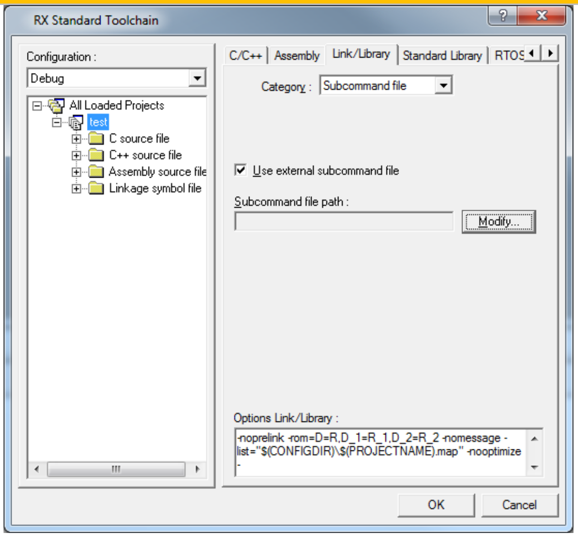
```
-cpu=rx200 -output=obj="$(CONFIGDIR)\
$(FILELEAF).obj" -debug -nologo
```



Additional include directories: (one per line)

```
$PROJ_DIR$\..\..\RPDL\
$PROJ_DIR$
```

Linker configuration file



Configuration : Debug

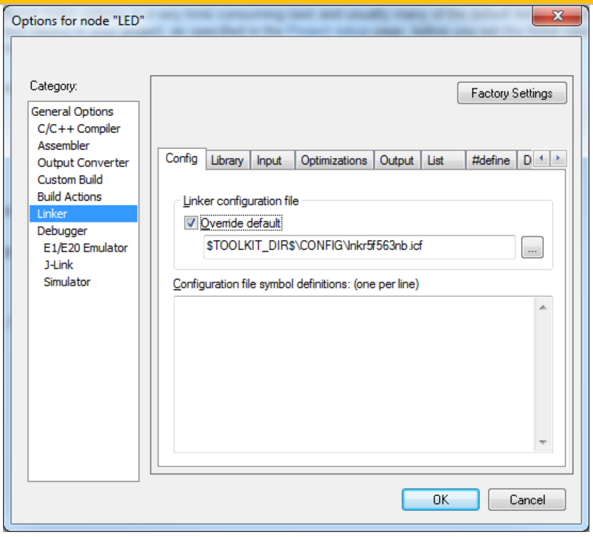
Category : Subcommand file

☑ Use external subcommand file

Subcommand file path :

Options Link/Library :

```
-noprelink -rom=D=R,D_1=R_1,D_2=R_2 -nomessage -
list="$(CONFIGDIR)\$(PROJECTNAME).map" -nooptimize
-
```

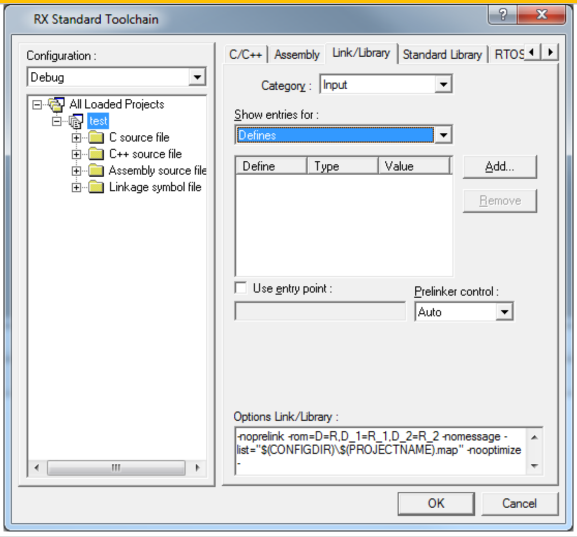


Linker configuration file

☑ Override default

```
$TOOLKIT_DIR$\CONFIG\lnkr5f563nb.icf
```

Linker symbols


RX Standard Toolchain

Configuration : Debug

All Loaded Projects
- test
  - C source file
  - C++ source file
  - Assembly source file
  - Linkage symbol file

Tabs: C/C++ | Assembly | Link/Library | Standard Library | RTOS

Category : Input

Show entries for : Defines

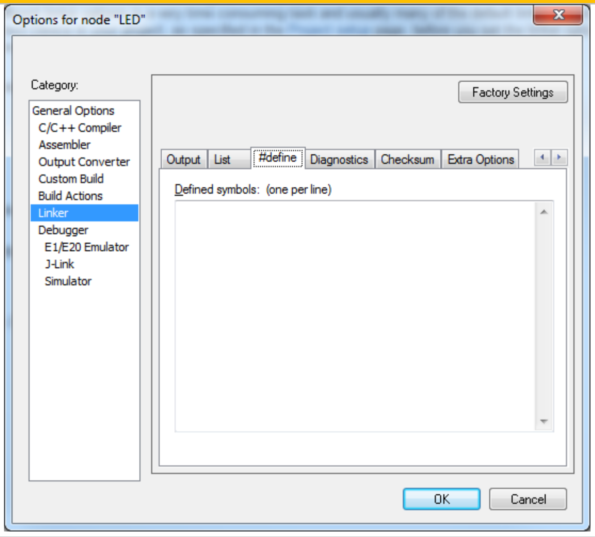
Define	Type	Value

Add... Remove

Use entry point :

Prelinker control : Auto

Options Link/Library :
```
-noprelink -rom=D=R,D_1=R_1,D_2=R_2 -nomessage -
list="$(CONFIGDIR)\$(PROJECTNAME).map" -nooptimize
-
```
OK Cancel


Options for node "LED"

Category:
- General Options
- C/C++ Compiler
- Assembler
- Output Converter
- Custom Build
- Build Actions
- Linker
- Debugger
  - E1/E20 Emulator
  - J-Link
  - Simulator

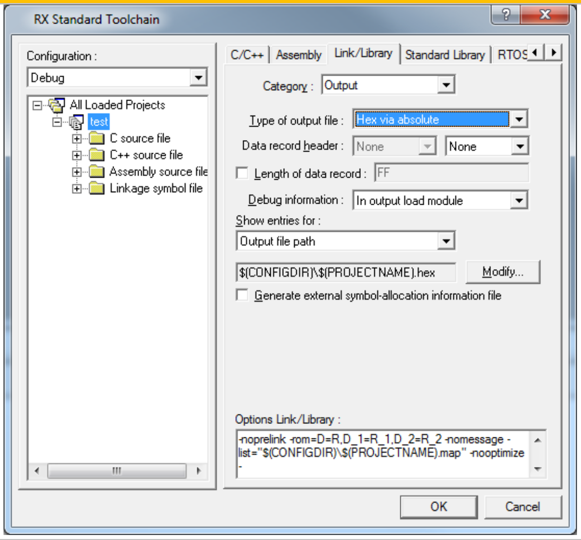
Factory Settings

Tabs: Output | List | #define | Diagnostics | Checksum | Extra Options

Defined symbols: (one per line)

OK Cancel

Additional output format


RX Standard Toolchain

Configuration : Debug

All Loaded Projects
- test
  - C source file
  - C++ source file
  - Assembly source file
  - Linkage symbol file

Tabs: C/C++ | Assembly | Link/Library | Standard Library | RTOS

Category : Output

Type of output file : Hex via absolute

Data record header : None None

Length of data record : FF

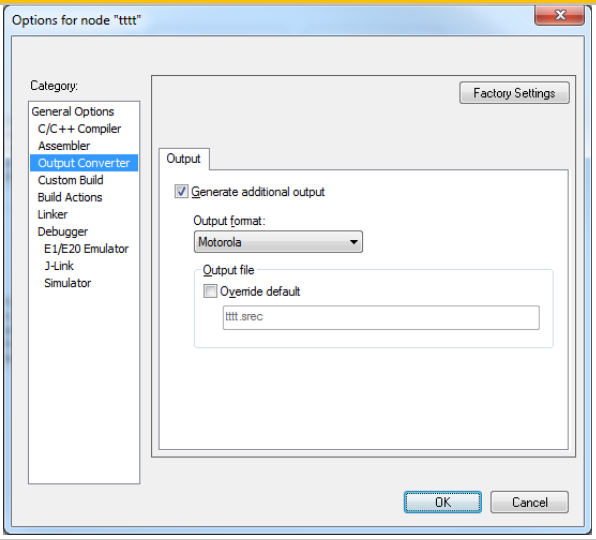
Debug information : In output load module

Show entries for : Output file path

$(CONFIGDIR)\$(PROJECTNAME).hex Modify...

Generate external symbol-allocation information file

Options Link/Library :
```
-noprelink -rom=D=R,D_1=R_1,D_2=R_2 -nomessage -
list="$(CONFIGDIR)\$(PROJECTNAME).map" -nooptimize
-
```
OK Cancel


Options for node "tttt"

Category:
- General Options
- C/C++ Compiler
- Assembler
- Output Converter
- Custom Build
- Build Actions
- Linker
- Debugger
  - E1/E20 Emulator
  - J-Link
  - Simulator

Factory Settings

Tabs: Output

☑ Generate additional output

Output format: Motorola

Output file

☐ Override default

tttt.srec

OK Cancel

Note: We recommend that you verify all settings to make sure they match your project needs.

Reference information

Locate a feature in the left-hand column; then you can find the IAR Systems counterpart to the right. For detailed information about this feature specific to IAR Embedded Workbench®, see the relevant documentation. For a complete list of guides, see IAR Information Center in the IDE.

Compiler-specific details

Renesas HEW/e2studio	IAR Embedded Workbench
Programming languages	
Assembler, C(C89/C99), C++, EC++	Supported programming languages: assembler, C, Embedded C++, Extended Embedded C++, and C++. For C, the C99 standard is default, but C89 can optionally be used. C99 is supported by the library.
Processor configuration	
- CPU type RX600 (incl. FPU) or RX200 (no FPU) - Big endian or little endian - Bit order (in bitfields) left or right	-CPU type RX100, RX200, RX600, or RX610 -Big endian or little endian -Bit order (in bit fields) left or right
Memory models/Data models/Code models	
None	Supported data models (option `--data_model`): Near: Low 32 Kbytes or high 32 Kbytes Far (default): Low 8 Mbytes or high 8 Mbytes Huge: The entire 4 Gbytes of memory
Overriding default placement of given code/data model	
Segment names for both code and data segments can be modified using the `#pragma section` command.	To place a variable or function in a named section, use: `#pragma location="FLASH"`

	To override default placement of the selected data model, use any of these memory attributes:
	`__data16`
	`__data24`
	`__data32`

Absolute placement of variables	

`#pragma ADDRESS variable_name = absolute_address`	`__no_init char a @0x80;`
	
	or
	
	`#pragma location=0x80`
	`__no_init const int a;`

Absolute placement of functions	

`#pragma section P MyFunction`
`void foo(void);`
	`void foo(void) @ 0x2000;`
	or
	`void foo(void) @ "MyFunctions"`
	or
	`#pragma location="MyFunctions"`
	`void foo(void);`

The section `MyFunction` must be defined in the linker options.	The section `MyFunction` must be placed by customizing the linker configuration file. See *Customizing the linker configuration file* in the development guide. To place a function at a specific location, the section must first be created in the linker configuration file (`.icf`). This can be achieved with: `place at address Mem:[0] {readonly section MyFunction};` Where the `MyFunction` section will be placed at address 0 in `Mem`.
Constants in ROM	
`Const unsigned char c_char[] = [0x1234, 0x5678};`	`const unsigned short constants[] = {0x1234, 0x5678}`
Interrupt functions	

`#pragma interrupt function_name (interrupt specification)`
Interrupt Specifications
1. Vector table
`vect= <vector number>` Specifies the vector number for which the interrupt function address is stored.
2 Fast interrupt
`fint` Specifies the function used for fast interrupts. This `RTFI` instruction is used to return from the function.
3 Limitation on registers in interrupt function
`save` Limits the number of registers used in the interrupt function to reduce save and restore operations.
4 Nested interrupt enable
`enable` Sets the I flag in PSW to 1 at the beginning of the function to enable nested interrupts.
5 ACC saving
`acc` None Saves and restores ACC in the interrupt function.
6 ACC non-saving
`no_acc` Does not save and restore ACC in the interrupt function.
	`#pragma vector =`
	`__interrupt [__nested] void MyInterruptRoutine(void)`
	`{`
	`  /* Do something here.*/`
	`}`
	or
	`#pragma vector = /* Symbol from I/O header file */`
	`__interrupt void MyInterruptRoutine(void)`
	`{`
	`  /* Do something here. */`
	`}`
	The `__nested` keyword modifies the enter and exit code of an interrupt function to allow for nested interrupts.
	
	Note that an interrupt function must have the return type `void`, and it cannot specify any parameters.

Inline assembler	

`#pragma inline_asm[(]<function name>[,...][)]`
 	
Example:
`#pragma inline_asm Add`
 	
`static int Add(int a, int b){`
`ADD R2,R1 ; Assembly-language description`
`}`
	`asm [volatile]( string [assembler-interface])`
	`string` can contain one or more valid assembler instructions or data definition assembler directives, separated by `\n`.
	Example:
	`asm("movw ax, sp");`
	`asm("mov a, 0xff");`
	
	Example:
	`int Add(int term1, int term2)`
	`{`
	`int sum;`
	`asm("add %2,%1,%0 \n"`
	`: "=r"(sum)`
	`: "r"(term1), "r"(term2));`
	`return sum;`
	`}`

Renesas HEW/e2studio		IAR Embedded Workbench
Sizes on integers and floating-point		
8 bits	`char`	8 bits
32 bits	`int`	16 or 32 bits (--int={16|32})
16 bits	`short`	16 bits
32 bits	`float`	32 bits
32 bits	`long`	32 bits
64 bits	`long long`	64 bits
32 or 64 bits (when option `dbl_size=8` is specified double is 64 bits)	`double`	32 or 64 bits (`--double={32|64}`)
32 bits	`size_t`	-
32 bits	`ptr_diff_t`	-
32 bits	`enum`	-
32 bits	`Pointer`	-
8 bits	`bool`	8 bits
Extended keywords		
`_far, far`		`__data24`
`_near, near`		`__data32`
`__evenaccess <type specifier> <variable name>`	This extension guarantees access in the size of the target variable.	__sfr

`<type specifier> __evenaccess <variable name>`		
Pragma directives		
`#pragma section [<section type>] [ <new section name>]`	Switches sections	`#pragma section`
`#pragma stacksize {si=<constant> | su=<constant>}`	Creates a stack section	–
`#pragma interrupt [(]<function name> [(<interrupt specification> [,...])][,...][)]`	Creates an interrupt function	`#pragma vector=[interrupt] __interrupt [__nested] void <function>`
`#pragma inline [(]<function name>[,...][)]` `#pragma noinline [(]<function name>[,...][)]`	Performs inline expansion of a function or disables inlining of a function	`#pragma inline`
`#pragma inline_asm[(]<function name> [,...][)]`	Performs inline expansion of an assembly-language function	–
`#pragma entry[(]<function name>[)]`	Creates an entry function	–
`#pragma option [<option string>]`	Specifies options for a function	–
`#pragma bit_order [{left | right}]`	Switches the order of bit assignment	`#pragma bitfield=reversed`
`#pragma pack` `#pragma unpack` `#pragma packoption`	Specifies the boundary alignment value for structure members and class members	`#pragma pack(1)` `#pragma pack()`
`#pragma address [(]<variable name>=<absolute address> [,...][)]`	Specifies an absolute address for a variable	`#pragma location={address|NAME}`
`#pragma endian [{big | little}]`	Specifies the byte order for initial values	–
`#pragma instalign4 [(]<function name>[(<branch destination type>)][,...][)]` `#pragma instalign8 [(]<function name>[(<branch destination type>)][,...][)]` `#pragma noinstalign [(]<function name>[,...][)]`	Specifies the function in which instructions at branch destinations are aligned for execution	–
Intrinsic functions		
`signed long max(signed long data1, signed long data2)`	Selects the maximum value.	–
`signed long min(signed long data1, signed long data2)`	Selects the minimum value.	–
`unsigned long revl(unsigned long data)`	Reverses the byte order in longword data.	–
`unsigned long revw(unsigned long data)`	Reverses the byte order in longword data in word units.	–
`void xchg(signed long *data1, signed long *data2)`	Exchanges data.	–
`long long rmpab(long long init, unsigned long count, signed char *addr1, signed char *add2)`	Multiply-and-accumulate operation (byte).	`void __RMPA_B(signed char * v1, signed char * v2, unsigned long n, rmpa_t * acc)`
`long long rmpaw(long long init, unsigned long count, short *addr1, short *add2)`	Multiply-and-accumulate operation (word).	`void __RMPA_W(signed short * v1, signed short * v2, unsigned long n, rmpa_t * acc);`
`long long rmpal(long long init, unsigned long count, long *addr1, long *add2)`	Multiply-and-accumulate operation (longword).	`void __RMPA_L(signed long * v1, signed long * v2, unsigned long n, rmpa_t * acc)`
`unsigned long rolc(unsigned long data)`	Rotates data including the carry to left by one bit.	–
`unsigned long rorc(unsigned long data)`	Rotates data including the carry to right by one bit.	–
`unsigned long rotl(unsigned`	Rotates data to the left.	–

`long data)`		
`unsigned long rotr(unsigned long data)`	Rotates data to the right.	–
`void brk(void)`	`BRK` instruction exception.	`void __break(void)`
`void int_exception(signed long num)`	`INT` instruction exception	`void __software_interrupt(void)`
`void wait(void)`	Stops program execution	`void __wait_for_interrupt(void)`
`void nop(void)`	Expanded to a `NOP` instruction	`void __no_operation(void)`
`void set_ipl(signed long level)`	Sets the interrupt priority level.	`void __set_interrupt_level( __ilevel_t)`
`unsigned char get_ipl(void)`	Refers to the interrupt priority level.	`__ilevel_t __get_interrupt_level(void)`
`void set_psw(unsigned long data)`	Sets data to `PSW`.	–
`unsigned long get_psw(void)`	Refers to `PSW` value.	–
`void set_fpsw(unsigned long data)`	Sets data to `FPSW`.	–
`unsigned long get_fpsw (void)`	Refers to `FPSW` value.	–
`void set_usp(void * data)`	Sets data to `USP`.	–
`void * get_usp(void)`	Refers to `USP` value.	–
`void set_isp(void * data)`	Sets data to `ISP`.	`void __get_interrupt_level(__ilevel_t);`
`void * get_isp(void)`	Refers to `ISP` value.	`__ilevel_t __get_interrupt_level(void);`
`void set_intb(void * data)`	Sets data to `INTB`.	`void __set_interrupt_table( unsigned long address)`
`void * get_intb(void)`	Refers to `INTB` value.	`unsigned long __get_interrupt_table(void)`
`void set_bpsw(unsigned long data)`	Sets data to `BPSW`.	–
`unsigned long get_bpsw (void)`	Refers to `BPSW` value.	–
`void set_bpc(void * data)`	Sets data to `BPC`.	–
`void * get_bpc(void)`	Refers to `BPC` value.	–
`void set_fintv(void * data)`	Sets data to `FINTV`.	`void __set_FINTV_register(unsigned long address)`
`void * get_fintv(void)`	Refers to `FINTV` value.	`unsigned long __get_FINTV_register(void)`
`signed long long emul (signed long data1, signed long data2)`	Signed multiplication of significant 64 bits.	–
`unsigned long long emulu (unsigned long data1, unsigned long data2)`	Unsigned multiplication of significant 64 bits.	–
`void chg_pmusr(void)`	Switches to user mode.	–
`void set_acc(signed long long data)`	Sets data to `ACC`.	–
`signed long long get_acc (void)`	Refers to `ACC` value.	–
`void setpsw_i(void)`	Sets the interrupt enable bit to 1.	`void __enable_interrupt(void)`
`void clrpsw_i(void)`	Clears the interrupt enable bit to 0.	`void __disable_interrupt(void)`
`long macl(short* data1, short* data2, unsigned long count)`	Multiply-and-accumulate operation of 2-byte data.	–
`short macw1(short* data1, short* data2, unsigned long count)` `short macw2(short* data1, short* data2, unsigned long count)`	Multiply-and-accumulate operation of fixed-point data.	–
`__sectop("<section name>")`	Refers to the start address of the specified <section name>.	`__section_begin("<section name>")`
`__secend("<section name>")`	Refers to the end address of the	`__section_end("<section name>")`

	specified <section name>.	
__secsize("<section name>")	Refers to the size of the specified <section name>.	__section_size("<section name>")
Preprocessor symbols		
__RX600 / __RX200	Processor type	__RX100__ / __RX200__ / __RX600__ / __RX610__
__BIG / __LIT	Little/big endian	__BIG_ENDIAN__ / __LITTLE_ENDIAN__
__DBL4 / __DBL8	Double size	__DOUBLE__
__INT_SHORT	INT size	__INTSIZE__
__SCHAR / __UCHAR	Plain char is signed/unsigned	-
__SBIT / __UBIT	Bitfield is signed/unsigned	-
__ROZ / __RON	Round to zero/round to nearest	-
__DON / __DOFF	Denormalize = on/off	-
__BITLEFT / __BITRIGHT	Bit order = left/right	-
__AUTO_ENUM	Automatic size for enum	-
__FUNCTION_LIB __INTRINSIC_LIB	Library = function/intrinsic	-
__FPU	FPU available	__FPU__
__RENESAS__	Renesas compiler	__IAR_SYSTEMS_ICC__
__RENESAS_VERION__ 0xAABBCC00	Compiler version	__VER__
_RX	Compiler used	__ICCRX__
_PIC	Position-independent code	__ROPI__
_PID	Position-independent data	__ROPI__ (constant data only)
Compiler options		
lang = c/cpp/ecpp/c99	Defines C variant: C89 / C ++ / embedded C++ / C99	--c89 / --ec++ / --eec++
include=<path name>[,]	Include file directory	-I <path>
preinclude=<file name>[,]	Files to be included at compilation start	--preinclude <file name>
define = <sub>[,] <sub>:<macro name>[=<string>]	Macro definitions	-
undefine = <sub>[,] <sub>:< macro name >	Macro remove	-
Message	Enables information message output.	--remark
nomessage[=<error number> [-<error number>][,]]	Disables message output	--diag_suppress=tag[,tag,...]
change_message =<sub>[,] <sub>:<level> [=<n>[-m][,]] <level>:{Information | warning | error}	Changes the severity level of the compiler output messages.	--diag_error=tag[,tag,...] --diag_remark=tag[,tag,...] --diag_suppress=tag[,tag,...] --diag_warning=tag[,tag,...]
file_inline_path=< path name>[,]	Path to files for inter-file inline expansion.	-
comment = { nest | nonest }	Nesting of C comments enable/disable	-
check={ nc | ch38 | shc }	Check compatibility with other Renesas compiler.	-
output = {prep | src | obj | abs | hex | sty} [= file name]	Define output file format	Via the IAR ELF Tool (ielftool.exe)
noline	Disables #line output at preprocessor expansion.	-
debug / nodebug	Enables/disables output of debug information	--debug
section = <sub>[,] <sub>: {P = <section name> | C = <section name> | D = <section name> | B = <section name> | L = <section name> | W = <section name>}	Change section name Program section Const section Data section BSS section Literal section	-

	Switch table section	
`stuff`	Allocates variables to sections matching the alignment value.	–

`nostuff[= {`
`B`
`| D`
`| C`
`| W } [,]]`
	Align section to 4-byte boundary.
	BSS section
	Data section
	Const section
	Switch table section
		–

`-instalign4[={ loop | inmostloop }]`	Aligns instructions at branch destinations to 4-byte boundaries.	`--align_func={1|2|4|8}`
`-instalign8[={ loop | inmostloop }]`	Aligns instructions at branch destinations to 8-byte boundaries.	`--align_func={1|2|4|8}`
`-noinstalign`	Does not align instructions at branch destinations.	–

Assembler-specific details

Renesas HEW/e2studio	IAR Embedded Workbench
Limitations in source code structure	
Interrupt functions in assembler	

	Interrupt functions should be declared as `ROOT` so that they cannot be discarded by the linker even if no symbols in the segment are referred to. To insert an entry in the interrupt vector table, define the destination with the `DW` directive, for example like this:
	`  COMMON INTVEC:CODE:ROOT(1)`
	`  ORG     0x08    ;INTP0`
	`branchToInter0:`
	`  DW    inter0`

Segments	
All segments are defined using `.section` command.	Code segments are defined using the assembler directives `SECTION` or `RSEG`, which means segments. A `CSTACK` segment can also be defined.

	`RSEG name:CODE`
	`RSEG name:DATA`
	`RSEG name:CONST`
	
	or
	
	`SECTION name:CODE`
	`SECTION name:DATA`
	`SECTION name:CONST`

	Bit segments cannot be defined explicitly, but can easily be defined using bit operators in code or data segments. Because a byte is the smallest allocatable memory segment, no memory is lost or gained using either tool.
Number representation	
Numbers can be used in -Binary fomat (append B or b) -Octal format (append O or o) -Decimal fomat -Hexadecimal fomat (append H or h, must not start with a character)	Binary, octal, decimal and hexadecimal numbers are supported.

Renesas HEW/e2studio		IAR Embedded Workbench
Integer constants		
`1010B, 1010b`	Binary	`1010b, b'1010`
`1234O, 1234o`	Octal	`1234q, q'1234, 01234`
`1234`	Decimal	`1234, -1, d'1234, 1234d`

`0FFFFH, 0FFFFh, 0xFFFFh`	Hexadecimal	`0FFFFh, 0xFFFF, h'FFFF`
Assembler directives		
`.ORG <numeric value>`	Declares the start address. The section including this directive becomes an absolute-addressing section.	-
`.OFFSET <numeric value>`	Specifies an offset from the beginning of the section. This directive can be used only in a relative-addressing section.	-

`.ENDIAN BIG`
`.ENDIAN LITTLE`
	Specifies the byte order for the section.	-
		-

`<label name:> .BLKB <operand>`	Allocates a RAM area in 1-byte units.	`DS8`
`<label name:> .BLKW <operand>`	Allocates a RAM area in 2-byte units.	`DS16`
`<label name:> .BLKL <operand>`	Allocates a RAM area in 4-byte units.	`DS32`
`<label name:> .BLKD <operand>`	Allocates a RAM area in 8-byte units.	`DS64`
`<label name:> .BYTE <operand>`	Stores 1-byte data in a ROM area.	`DC8`
`<label name:> .WORD <operand>`	Stores 2-byte data in a ROM area.	`DC16`
`<label name:> .LWORD <operand>`	Stores 4-byte data in a ROM area.	`DC32`
`<label name:> .FLOAT <operand>`	Stores 4-byte floating data in a ROM area.	`DF32`
`<label name:> .DOUBLE <operand>`	Stores 8-byte floating data in a ROM area.	`DF64`
`.ALIGN <alignment value>`	Corrects a location counter to a multiple of the boundary alignment value.	`ALIGNRAM`
`<name> .EQU <numeric value>`	Defines a symbol	`EQU`
`.END`	Specifies the end of an assembly-language file.	`END`
`.INCLUDE <include file name>`	Inserts the contents of the specified file to the location where this directive is written.	`#include`
`.SECTION <section name>, ALIGN=[2|4|8] <section attribute>: [CODE|ROMDATA|DATA]`	Defines a section, which is the minimum unit used for address relocation.	`SECTION segment :type [flag] [(align)]`
`.GLB <name>[,<name> …]`	Declares an external symbol.	`EXTERN`
`.RVECTOR <number>,<name>`	Registers a symbol as a variable vector.	-
`.LIST [ON|OFF]`	Switch output to list file on/off	`LSTOUT{+|-}`

`.IF conditional expression`
`body`
`.ELIF conditional expression`
`body`
`.ELSE`
`body`
`.ENDIF`
	Conditional assembly	`IF`
		
		`ELSEIF`
		
		`ELSE`
		
		`ENDIF`

`.ASSERT "<string>">> <file name>`	Outputs a string specified in an operand to the standard error output or a file.	-

`<mnemonic > ?+`
`<mnemonic > ?-`
	Defines and references a temporary label.	

`<string>@<string>[@<string> ...]`	Concatenates strings specified before and after `@` so that they are handled as one string.	-
`..FILE`	Indicates the name of the assembly-language file being processed by	-

	the assembler.	
.STACK <name>=<numeric value>	Defines a stack value for a specified symbol.	-
.LINE <file name>,<line number>	Changes line number.	-
<symbol name> .DEFINE <string>	Defines a replacement symbol.	-
<macro name> .MACRO[<parameter>[,...]]	Defines a macro name and the beginning of a macro body.	<macro name> MACRO [argument] [,argument] ...
.EXITM	Terminates macro body expansion.	EXITM
.LOCAL <label name>[,...]	Declares a local label in a macro.	LOCAL symbol [,symbol] ...
.ENDM	Specifies the end of a macro body.	ENDM
[<label>:] .MREPEAT <numeric value>	Specifies the beginning of a repeat macro body.	REPT
.ENDR	Specifies the end of a repeat macro body.	ENDR
..MACPARA	Indicates the number of arguments in a macro call.	-
..MACREP	Indicates the count of repeat macro body expansions.	-
.LEN {"<string>"}	Indicates the number of characters in a specified string.	-
.INSTR { "<string>","<search string>",<search start position> }	Indicates the start position of a specified string in another specified string.	-
.SUBSTR { "<string>",<extraction start position>,<extraction character length> }	Extracts a specified number of characters from a specified position in a specified string.	-
._LINE_TOP ._LINE_END	These directives are output when the functions specified by #pragma inline_asm have been expanded.	-
.SWSECTION .SWMOV .SWITCH	These directives are output when the branch table is used in the switch statement.	-
.INSTALIGN	This directive is output when #pragma instalign4, or #pragma instalign8 is used.	-
Assembler options		
include=<path name>[,...]	Specifies the name of the path to the folder that stores the include file.	-I <path name>
define=<sub>[,...] <sub>: <replacing symbol name> =<string>	Defines <string> as <replacing symbol name>.	-
chkpm	Checks for a privileged instruction.	-
chkfpu	Checks for a floating-point operation instruction.	-
chkdsp	Checks for a DSP instruction.	-
output= <output file name>	Specifies the relocatable file name.	-o <output file name> --output <output file>
debug nodebug	Does /does not output debug information.	--debug
goptimize	Outputs additional information for inter-module optimization.	-
listfile[=<file name>] nolistfile	Does / does not output a source list file.	-l[a][d][e][m] [o][x][N][H] {filename|directory}
show = { conditionals | definitions | expansions } [,...]	Specifies the contents of the output source list file.	-l[a][d][e][m] [o][x][N][H] {filename|directory}
cpu = { rx600 | rx200 }	Generates a relocatable file for the RX600 / RX200 Series.	--core={RX100|RX200|RX600|RX610}
endian = { big | little }	Big / little endian	--endian={b|big|l|little}

`fint_register = { 0 | 1 | 2 | 3 | 4 }`	Specifies general registers to be used only for fast interrupts.	-
`base = { rom = <register> | ram = <register> | <address> = <register>}[,...]`	Specifies the base register for ROM / RAM / SFR	-
`patch = { rx610 }`	Avoids a problem specific to the CPU type.	-
`pic`	Generates an object with the PIC function enabled.	-
`pid = { 16 | 32 }`	Generates an object with the PID function enabled and selects the offset width.	-
`nouse_pid_register`	Does not use the PID register for code generation.	-
`logo` `nologo`	Enables / disables copyright message	-
`subcommand = <file name>`	Inputs command line specifications from a file.	-

`euc`
`sjis`
`latin1`
	Selects character input code: EUC / SJIS / ISO-Latin1	-

Linker and library details

Renesas HEW/e2studio	IAR Embedded Workbench
Device-specific header files	
All standard projects use a file called `iodefine.h` for all processor specific SFRs.	All SFRs are defined in `ioxxx.h` files located in the `rx\inc` directory.

Renesas HEW/e2studio		IAR Embedded Workbench
Linker options		
`Input = <file name> [(<module name>[,...])] [{,|Δ}...]`	Input files	No specific option. Just list the files.
`LIBrary = <file name>[,...]`	Input library files	No specific option. Just list the files.
`Binary = <file name>(<section name> [:<boundary alignment>] [/<section attribute>] [,<symbol name>]) [,...]`	Input binary files	-
`DEFine = <symbol name> = {<symbol name> | <numerical value>} [,...]`	Symbol definition	-
`ENTry = {<symbol name>| <address>}`	Execution start address	`--entry <symbol>`
`NOPRElink`	Disables prelinker start	-
`FOrm ={ Absolute | Relocate | Object | Library [= {S|U}] | Hexadecimal | Stype | Binary }`	Output format	Can only output Elf/Dwarf format. To convert, use `ielftool.exe.`

`DEBug`
`SDebug`
`NODEBug`
	Debug information in output file
	Debug information in debug file
	No debug information
		Specified at compile time.

`REcord={ H16 | H20 | H32 | S1 | S2 | S3 }`	Format definition for hex-file output	-
`ROm = <ROM section name> = <RAM section name> [...]`	Reserves an area in RAM for the relocation of a symbol with an address in RAM.	-

`OUtput = <file name>[={<start address> -<end address> | <section name>[:...]} [,...] ]`	Specifies output file (range specification and divided output are enabled)	`-o <file name> /`
		`--output <file name>`

`MAp [= <file name>]`	Specifies output of the external symbol-allocation information file (for SuperH Family and RX Family)	`--map {filename|directory}`
`SPace [= {<numerical value> | Random}]`	Specifies a value to output to unused area	-

`Message`
`NOMessage [=<error code> [-<error code>] [,...]]`
	Output information messages
	Disable information messages (all or selected)
		`--remarks`

`MSg_unused`	Notification of unreferenced symbol	-
`BYte_count=<numerical value>`	Specification of data record byte count	-

`CRc = <write address> = <start address>-<end address>[,...]`
`[/{ CCITT | 16 }]`
`[:{BIGendian | LITTLEendian}]`
	CRC calculation	Is done by `ielftool.exe`, but space can be reserved with `--place_holder symbol [,size[,section[,alignment]]]`

`PADDING`	Filling padding data at section end	-
`VECTN=<vector number>={<symbol> | <address>} [,...]`	Initialize interrupt vector	By default, the vector table is populated with a *default interrupt handler* which calls the abort function. For each interrupt source that has no explicit interrupt service routine, the default interrupt handler will be called. If you write your own service routine for a specific vector, that routine will override the default interrupt handler.
`VECT={<symbol>|<address>}`	Address setting for unused variable vector area	See above.
`JUMP_ENTRIES_FOR_PIC =<section name>[...]`	Outputs a jump table (for the PIC function of RX Family)	-
`LISt [ = <file name>]`	Output list file	`--map file|directory`
`SHow [ = {SYmbol | Reference | SEction | Xreference | Total_size| VECTOR| ALL } [,...] ]`	List file contents	-

`OPtimize = {STring_unify | SYmbol_delete | Variable_access | Register | SAMe_code | SHort_format | Function_call | Branch | Speed | SAFe }[...]`
 	
`NOOPtimize`
	Enable optimization	`--inline`
		`--vfe=[forced]`

`SAMESize = <size>`
`(default: sames=1e)`
	Specifies the minimum size to unify same codes.	-

`PROfile = <file name>`	Specifies a profile information file. (Dynamic optimization is provided.)	-

`SYmbol_forbid= <symbol name>[,...]`
 	
`SAMECode_forbid= <function name>[,...]`
 	
`Variable_forbid= <symbol name>[,...]`
 	
`FUnction_forbid= <function name>[,...]`
 	
`SEction_forbid = [<file name>| <module name>] (<section name>[,...]) [,...]`
 	
`Absolute_forbid= <address>[+<size>][,...]`
	Optimization partially disabled	-

`STARt = [(]<section name> [{ : | , }<section name>[,...]] [)][,...] [/<address>] [,...]`	Define section arrangement in memory.	Done in linker configuration file with the `place in` directive. Read more in EWRX Development guide *Linking using ILINK*.

`FSymbol = <section name>[,...]`	Outputs externally defined symbol addresses to a definition file.	-
`ALIGNED_SECTION = <section name>[,...]`	Changes the section alignment value to 16 bytes.	-
`CPu = { <cpu information file name> | <memory type> = <address range>[,...] | STRIDE}`	Specifies a specifiable allocation range for section addresses.	-
`PS_check= <start address> - <end address> , <start address> -<end address> [,...]`	Specifies address ranges that might overlap each other in the physical space.	-
`CONTIGUOUS_SECTION = <section name>[,...]`	The specified section will not be divided.	-
`S9`	Always outputs the S9 record.	-
`STACk`	Outputs a stack use information file.	-
`Compress NOCOmpress`	Compresses debugging Information or not	-
`MEMory = [ High | Low ]`	Specifies the memory size occupied for linkage.	-
`REName = {<file name> (<name>=<name>[,...] ) | <module name> (<name>=<name>[,...] ) } [,...]`	Symbol or section name modification	`--redirect <from_symbol>=<to_symbol>`
`DELete = {<module name> | [ <file name>] (<name>[,...] ) } [,...]`	Deletes a symbol name or module name.	-
`REPlace = <file>[(<module>[,...]) ] [,...]`	Replaces modules of the same name in a library file.	-
`EXTract = <module>[,...]`	Extracts the specified module in a library file.	-
`STRip`	Deletes debug information in an absolute file or a library file.	`--strip`
`CHange_message={Information | Warning | Error } [=<error number> [-<error number>] [,...] ] [,...]`	Modifies message levels.	`--diag_error=tag [,tag,...]` `--diag_remark=tag [,tag,...]` `--diag_suppress=tag [,tag,...]` `--diag_warning=tag [,tag,...]`
`Hide`	Deletes local symbol name information	-
`Total_size`	Sends total sizes of sections after linkage to standard output.	-
Segments/Sections		
`B / B_2 / B_1`	BSS section: uninitialized data, alignment 4/2/1 byte	-
`D / D_2 / D_1`	Data section: initialized data, alignment 4/2/1 byte	`.data32.data / .data16.data`
`P`	Program section	`.text`
`R / R_2 / R_1`	ROM section: initialization data for “D” , alignment 4/2/1 byte	`.data32.data_init / .data16.data_init`
`C / C_2 / C_1`	Constant section, alignment 4/2/1 byte	`.data32.rodata / .data16.rodata`
`W / W_2 / W_1`	switch statement branch table area, alignment 4/2/1 byte	`.switch.rodata`
`L`	Literal section	
`C$INIT`	C++ initial processing/ postprocessing data area	`DIFUNCT`
`C$VTBL`	C++ virtual function table area	-

C$VECT	Variable vector area	-
SU	User stack area	USTACK
SI	Interrupt stack area	ISTACK
$ADDR_<section>_<address>	Absolute address variable area	-

Runtime environment

Renesas HEW/e2studio		IAR Embedded Workbench
Calling convention		
Parameters passed on the stack		
Functions with a variable number of registers. Parameter 5 and more for functions with more than 4 parameters.		Functions with a variable number of registers. Parameter 5 and more for functions with more than 4 parameters.
Parameters passed in registers		
R1-R4	8-bit values in:	R1-R4
R1-R4	16-bit values in:	R1-R4
R1-R4	24-bit values in:	R1-R4
R1-R4	32-bit values in:	R1-R4
R1-R4	Floating-point values in:	R1-R4
Return values		
R1	8-bit values in:	R1
R1	16-bit values in:	R1
R1	24-bit values in:	R1
R1	32-bit values in:	R1
R1-R2	64-bit values in:	R1-R2
R1	32-bit Floating-point values in:	R1
R1-R2	64-bit Floating-point values in:	R1-R2
R1-R4	Structs up to 16 byte in:	R1-R4
Preserved registers		
R0, R6-R13		R0, R6-R13
Scratch registers		
R1-R5, R14, R15		R1-R5, R14, R15
System startup and exit code		
The system startup code is located in resetprg.c and uses dbsct.c. Customized hardware initialization can be placed in the function HardwareSetup() in the file hwsetup.c. Interrupt vectors and interrupt functions are predefined for all possible interrupt sources. These can be found in intprg.c and vecttbl.c.		The system startup code is located in the ready-made cstartup.s file. In addition, you specify additional settings, for example for the stack and heap size. It is likely that you need to customize the code for system initialization. This might be the case if, for example, your application needs to initialize memory-mapped special function registers, or omit the default initialization of data segments performed by cstartup. You can do this by providing a customized version of the routine __low_level_init, which is called from cstartup before the data segments are initialized. Modifying cstartup directly should be avoided.
Global variable initialization		
Static and global variables are initialized: zero-initialized variables are cleared and the values of other initialized variables are copied from ROM to RAM memory.		Static and global variables are initialized: zero-initialized variables are cleared and the values of other initialized variables are copied from ROM to RAM memory. This initialization can be overridden by returning 0 from the __low_level_init function. Variables declared __no_init which are not initialized at all: __no_init int i;
Reentrancy and recursive functions		
The library generator has an option to generate reentrant code or not.		The compiler is always reentrant when using the DLIB library.