

# Migration guide

## Migrating from Renesas HEW tool chain for RX to IAR Embedded Workbench® for RX

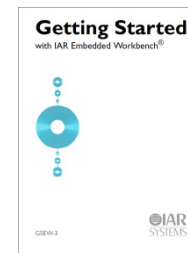
Use this guide as a guideline when converting project files and source code written for the Renesas HEW tool chain for RX to IAR Embedded Workbench® for RX.

	Product	Version number
Migrating from	Renesas HEW for RX (CCRX / HEW)	V.1.02.01 / 4.09
Migrating to	IAR Embedded Workbench for RX (EWRX)	V.2.41 and newer

### Migration overview

Migrating an existing project from HEW RX requires that you collect information about your current HEW 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:** Basic introduction to IAR Embedded Workbench and how to work in the IDE can be found in the document [Getting Started with IAR Embedded Workbench](#) available within the Information Center.



### Migration methods

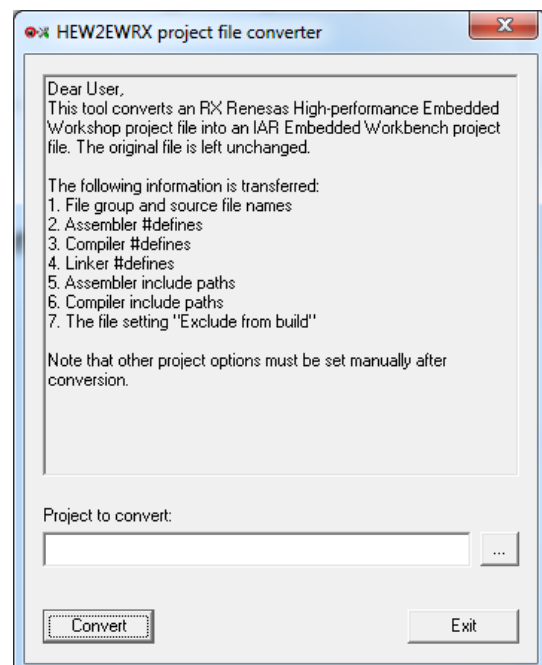
Migrating a HEW project can be done either via a graphical project conversion tool or with the help of Ruby scripts supplied with EWRX. Regardless of which method you choose you will most likely need to modify the source code slightly to succeed in building the project with EWRX.

#### Method 1 – Using the HEW2EWRX project conversion tool

The HEW2EWRX project conversion tool converts HEW 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.

##### Procedure

1. Start EWRX and open any EWRX workspace (to get access to the project conversion tool).
2. Start the HEW2EWRX project conversion tool available in the **Tools** menu.
3. Navigate to the HEW project to convert by clicking the Browse button.
4. Click the **Convert** 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, walk through the options and set the corresponding options in EWRX as suggested in the section **Important tool settings** below.



## Method 2 – Automatic conversion using Ruby scripts

With this method, source code migration as well as creating EWRX project files is done automatically. For each of the new EWRX projects, build options are copied from a EWRX template project and all project member files are copied from the old HEW project. The main advantages with using the Ruby scripts are:

- You can setup an EWRX template project to be reused for more than one project migration
- You can tune and extend the scripts to fit specific needs.
- The source code is migrated automatically to compile with EWRX.

Note: A 100% error-free conversion cannot be granted but the major part of required changes will be covered.

### Procedure

1. Start EWRX and create a new EWRX project by choosing **Project>Create New Project....**
2. Set the project options by choosing **Project>Options....**  
Hint: Open the original project in HEW, walk through the options and set the corresponding options in EWRX as suggested in the section **Important tool settings** below.
7. Locate the folder <install\_dir>\rx\src\scripts\HEW2EWRX\_migration and follow the instructions in the readme.txt file on how to use the script.

### Final touch

After successfully converting the HEW project you will most likely need to fine-tune parts of the source code so that it follows the EWRX syntax.

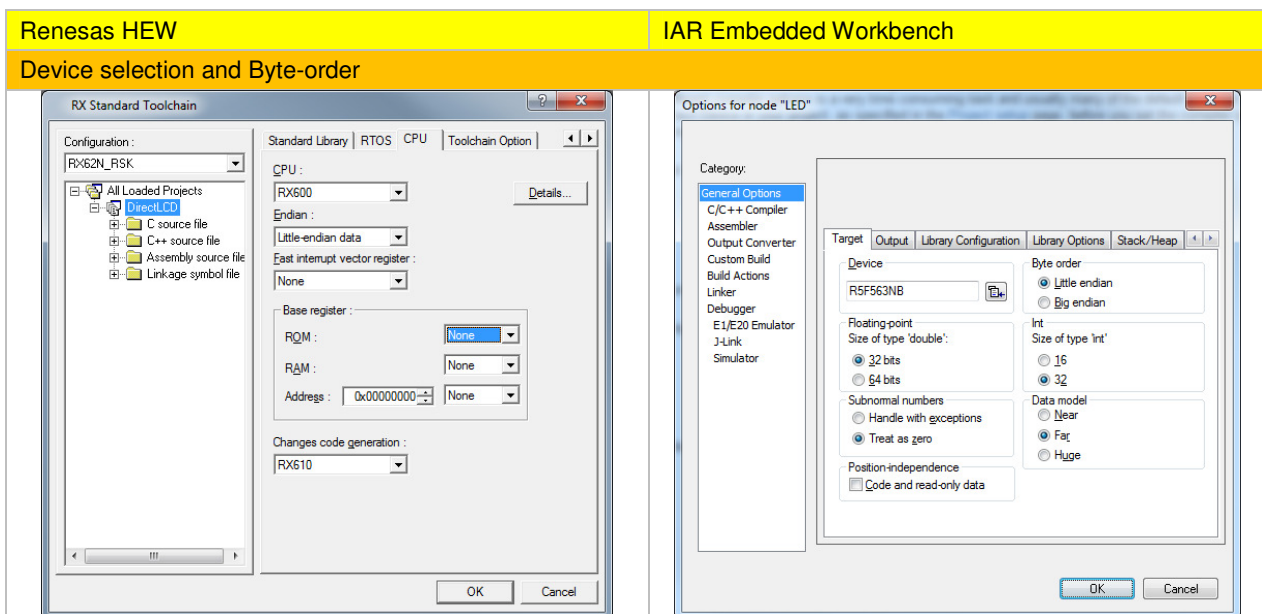
1. Choose **Project>Make.**
2. To find the different errors/warnings, press **F4** (Next Error/Tag).  
This will bring you to the location in the source code that generated this error/warning.
3. For each error/warning, modify the source code to match the EWRX syntax.  
Note: See the **Reference information** section below for this step.
4. 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.

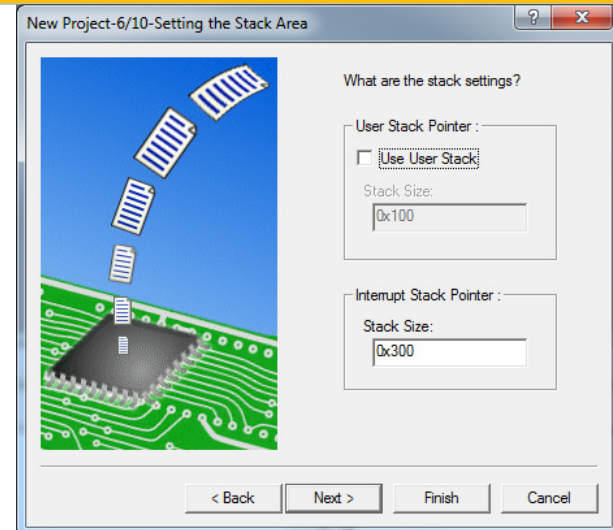
This is because errors and warnings later in the source code might just be effects of faulty syntax at the beginning of the source.

## Important tool settings

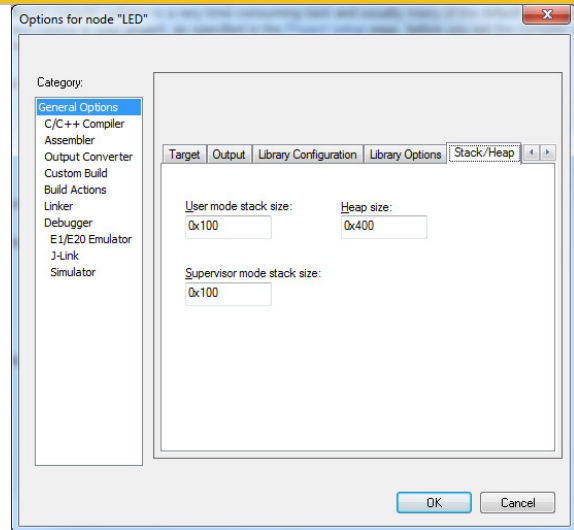
This is an overview of the most important tool settings. Make sure that they match your original HEW project. To change project settings, choose **Project>Options....**



## Stack/Heap size

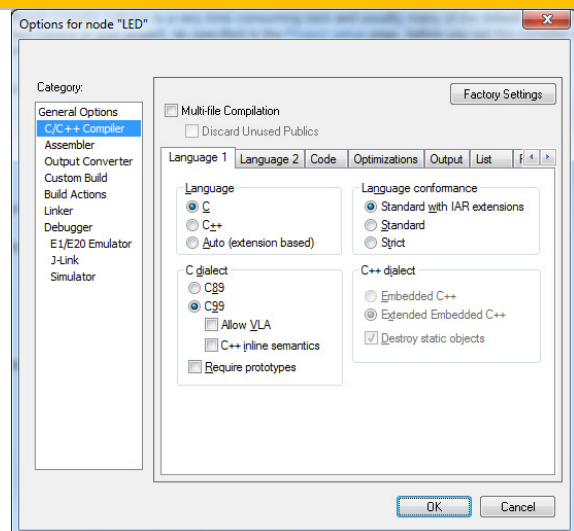
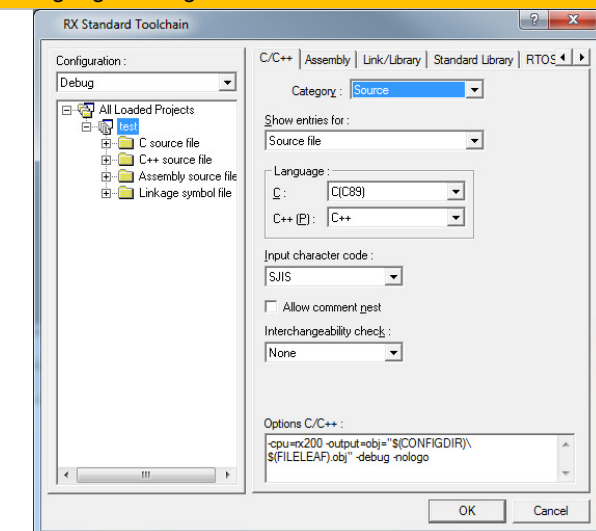


(Setup in project wizard which generates `stackset.h`)

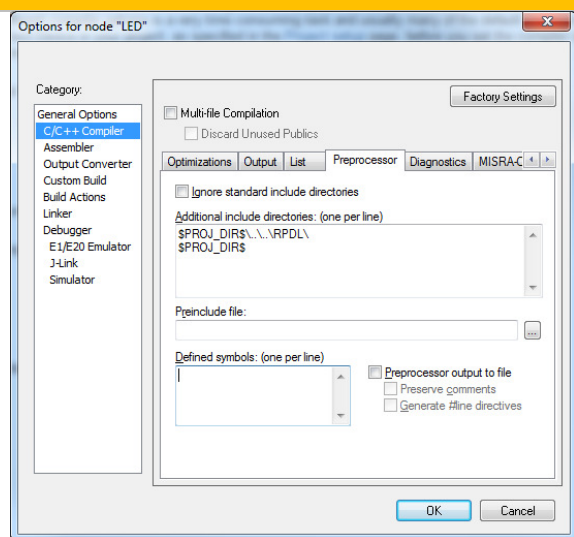
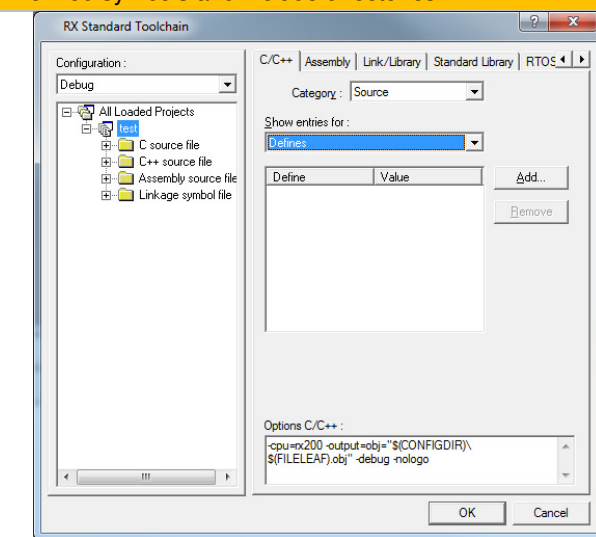


(Sets symbols used in the linker configuration file (`.icf`))

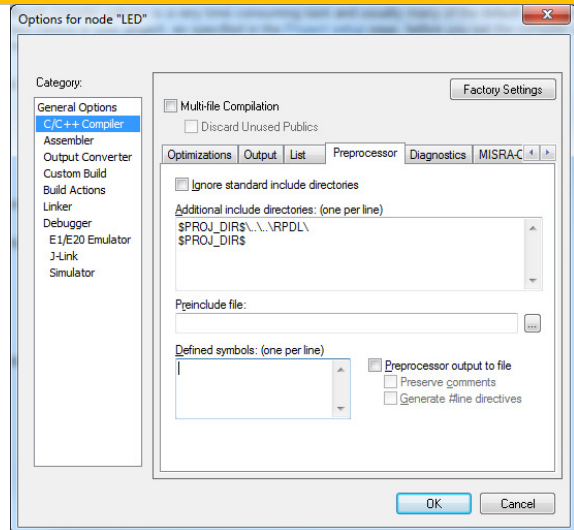
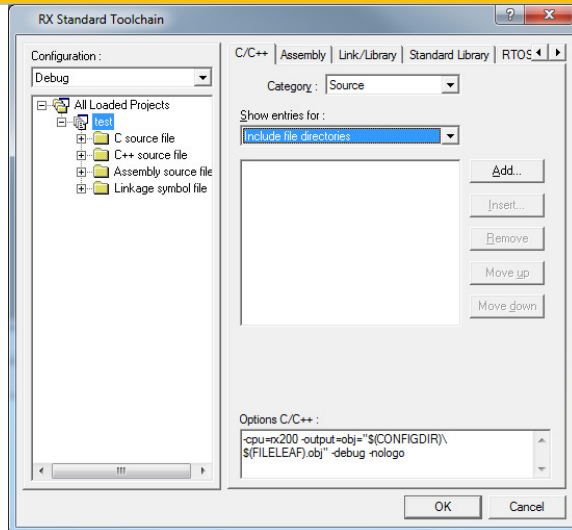
## Language settings



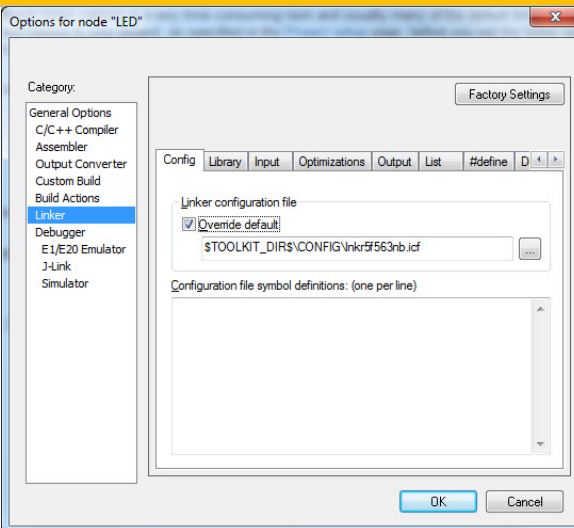
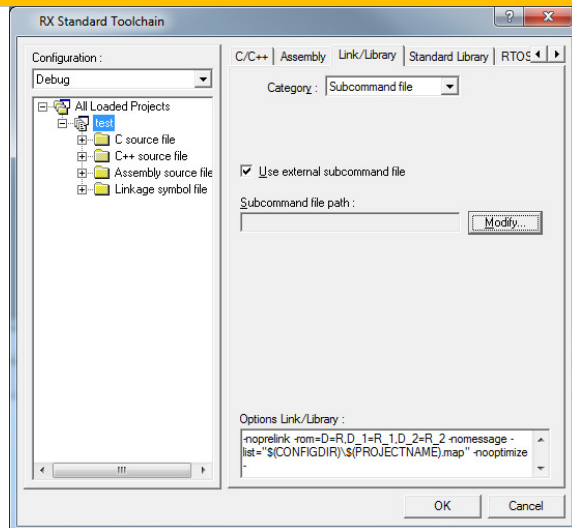
## Defined symbols and include directories



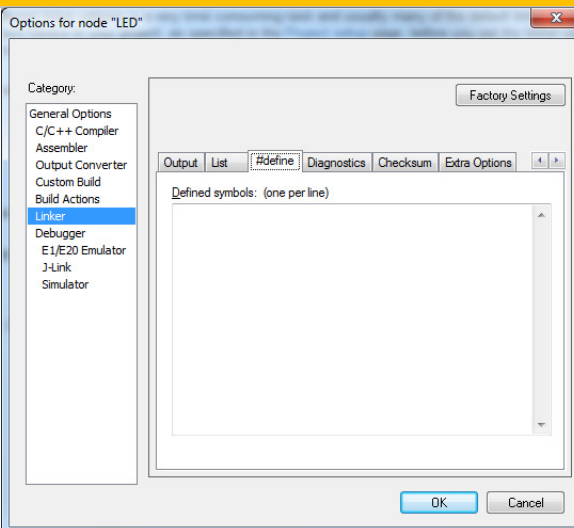
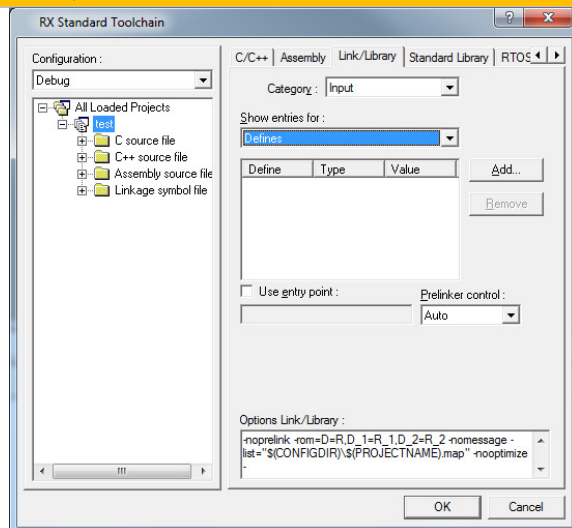
## Include directories



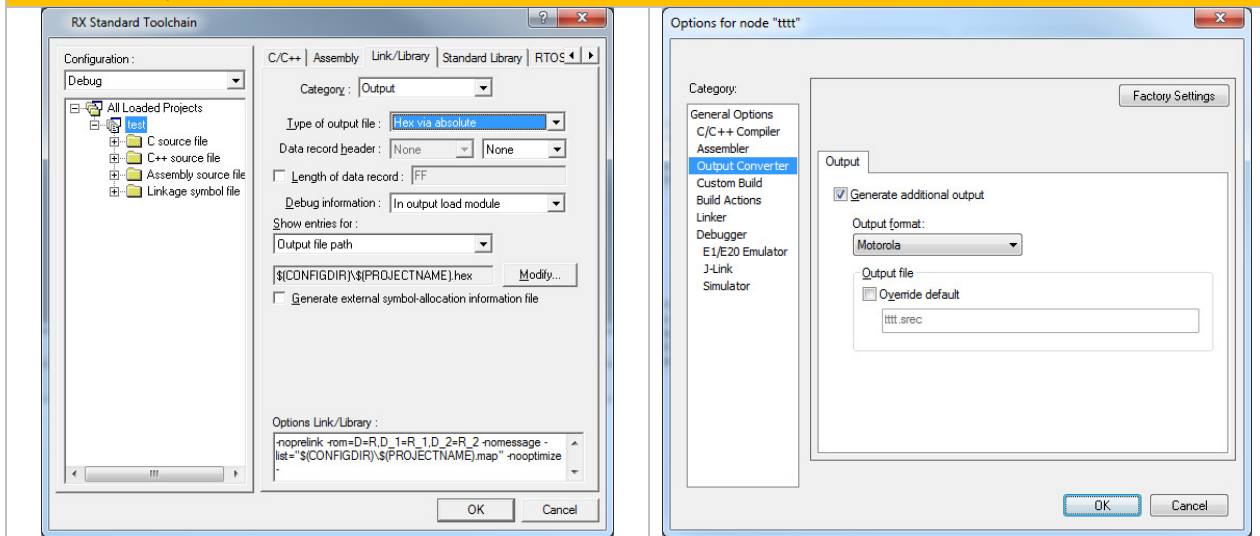
## Linker configuration file



## Linker symbols



## Additional output format



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	IAR Embedded Workbench
<b>Programming languages</b>	
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.
<b>Processor configuration</b>	
- 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
<b>Memory models/Data models/Code models</b>	
None	Supported data models (option <code>--data_model</code> ): Near: Low 32 Kbytes or high 32 Kbytes Far (default): Low 8 Mbytes or high 8 Mbytes Huge: The entire 4 Gbytes of memory
<b>Overriding default placement of given code/data model</b>	
Segment names for both code and data segments can be modified using the <code>#pragma section</code> command.	To place a variable or function in a named section, use: <code>#pragma location="FLASH"</code>
	To override default placement of the selected data model, use any of these memory attributes: <code>__data16</code> <code>__data24</code> <code>__data32</code>
<b>Absolute placement of variables</b>	
<code>#pragma ADDRESS variable_name = absolute_address</code>	<code>__no_init char a @0x80;</code>  or  <code>#pragma location=0x80</code> <code>__no_init const int a;</code>

**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

IAR Embedded Workbench

Sizes on integers and floating-point



## Migrating from Renesas HEW for RX to IAR Embedded Workbench for RX

8 bits	char	8 bits
32 bits	int	16 bits
16 bits	short	16 bits
32 bits	float	32 bits
32 bits	long	32 bits
64 bits	long long	32 bits
32 bits (when option dbl_size=8 is specified double is 64 bits)	double	32 bits (treated as float)
32 bits	size_t	-
32 bits	ptr_diff_t	-
32 bits	enum	-
32 bits	Pointer	-
8 bits	bool	8 bits
<b>Extended keywords</b>		
<code>_far, far</code>		<code>__data24</code>
<code>_near, near</code>		<code>__data32</code>
<code>__evenaccess &lt;type specifier&gt; &lt;variable name&gt;</code>  <code>&lt;type specifier&gt; __evenaccess &lt;variable name&gt;</code>	This extension guarantees access in the size of the target variable.	-
<b>Pragma directives</b>		
<code>#pragma section [&lt;section type&gt;] [ &lt;new section name&gt;]</code>	Switches sections	<code>#pragma section</code>
<code>#pragma stacksize {si=&lt;constant&gt;   su=&lt;constant&gt;}</code>	Creates a stack section	-
<code>#pragma interrupt [( &lt;function name&gt; [( &lt;interrupt specification&gt; [,...])],... )]</code>	Creates an interrupt function	<code>#pragma vector=[interrupt] __interrupt [__nested] void &lt;function&gt;</code>
<code>#pragma inline [( &lt;function name&gt;[,...])]</code> <code>#pragma noline [( &lt;function name&gt;[,...])]</code>	Performs inline expansion of a function or disables inlining of a function	<code>#pragma inline</code>
<code>#pragma inline_asm [( &lt;function name&gt; [,...])]</code>	Performs inline expansion of an assembly-language function	-
<code>#pragma entry [( &lt;function name&gt; )]</code>	Creates an entry function	-
<code>#pragma option [&lt;option string&gt;]</code>	Specifies options for a function	-
<code>#pragma bit_order [{left   right}]</code>	Switches the order of bit assignment	<code>#pragma bitfield=reversed</code>
<code>#pragma pack</code> <code>#pragma unpack</code> <code>#pragma packoption</code>	Specifies the boundary alignment value for structure members and class members	<code>#pragma pack</code>
<code>#pragma address [( &lt;variable name&gt;=&lt;absolute address&gt; [,...])]</code>	Specifies an absolute address for a variable	<code>#pragma location={address NAME}</code>
<code>#pragma endian [{big   little}]</code>	Specifies the byte order for initial values	-
<code>#pragma instalign4 [( &lt;function name&gt;[( &lt;branch destination type&gt;)],... )]</code> <code>#pragma instalign8 [( &lt;function name&gt;[( &lt;branch destination type&gt;)],... )]</code> <code>#pragma noinstalign [( &lt;function name&gt;[,...])]</code>	Specifies the function in which instructions at branch destinations are aligned for execution	-
<b>Intrinsic functions</b>		
<code>signed long max(signed long data1, signed long data2)</code>	Selects the maximum value.	-
<code>signed long min(signed long data1, signed long data2)</code>	Selects the minimum value.	-
<code>unsigned long revl(unsigned long data)</code>	Reverses the byte order in longword data.	-

## Migrating from Renesas HEW for RX to IAR Embedded Workbench for RX

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 long data)	Rotates data to the left.	-
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	Refers to ACC value.	-



## Migrating from Renesas HEW for RX to IAR Embedded Workbench for RX

(void)		
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 specified <section name>.	__section_end("<section name>")
__seclsize("<section name>")	Refers to the size of the specified <section name>.	__section_size("<section name>")
<b>Preprocessor symbols</b>		
__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)
<b>Compiler options</b>		
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	Define output file format	Via the IAR ELF Tool (ielftool.exe)

## Migrating from Renesas HEW for RX to IAR Embedded Workbench for RX

abs   hex   sty} [= file name]		
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	-
-installign4[={ loop   inmostloop }]	Aligns instructions at branch destinations to 4-byte boundaries.	--align_func={1 2 4 8}
-installign8[={ loop   inmostloop }]	Aligns instructions at branch destinations to 8-byte boundaries.	--align_func={1 2 4 8}
-noinstallign	Does not align instructions at branch destinations.	-

### Assembler-specific details

Renesas HEW	IAR Embedded Workbench
Limitations in source code structure	
Interrupt functions in assembler	
	<p>Interrupt functions should be declared as <code>ROOT</code> 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 <code>DW</code> directive, for example like this:</p> <pre>COMMON INTVEC:CODE:ROOT(1) ORG      0x08      ;INTP0 branchToInter0: DW      inter0</pre>
Segments	
All segments are defined using <code>.section</code> command.	Code segments are defined using the assembler directives <code>SECTION</code> or <code>RSEG</code> , which means segments. A <code>CSTACK</code> segment can also be defined.
	<pre>RSEG name:CODE RSEG name:DATA RSEG name:CONST</pre> <p>or</p> <pre>SECTION name:CODE SECTION name:DATA SECTION name:CONST</pre>
	<p>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.</p>
Number representation	

## Migrating from Renesas HEW for RX to IAR Embedded Workbench for RX

Numbers can be used in -Binary format (append B or b) -Octal format (append O or o) -Decimal format -Hexadecimal format (append H or h, must not start with a character)	Binary, octal, decimal and hexadecimal numbers are supported.	
<b>Renesas HEW</b>		<b>IAR Embedded Workbench</b>
<b>Integer constants</b>		
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
<b>Assembler directives</b>		
.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	Conditional assembly	IF  ELSEIF

## Migrating from Renesas HEW for RX to IAR Embedded Workbench for RX

.ELSE body .ENDIF		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.	–
<b>Assembler options</b>		
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.	–

## Migrating from Renesas HEW for RX to IAR Embedded Workbench for RX

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		IAR Embedded Workbench
<b>Device-specific header files</b>		
All standard projects use a file called <code>iodefine.h</code> for all processor specific SFRs.		All SFRs are defined in <code>ioxxx.h</code> files located in the <code>rx\inc</code> directory.
Renesas HEW		IAR Embedded Workbench
<b>Linker options</b>		
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 <code>ielftool.exe</code> .
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	-

## Migrating from Renesas HEW for RX to IAR Embedded Workbench for RX

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 <i>default interrupt handler</i> 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: same=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=	Optimization partially disabled	-

## Migrating from Renesas HEW for RX to IAR Embedded Workbench for RX

<pre>&lt;function name&gt;[,...]</pre> <pre>SSection_forbid = [&lt;file name&gt; </pre> <pre>  &lt;module name&gt;] (&lt;section</pre> <pre>name&gt;[,...]) [,...]</pre> <pre>Absolute_forbid=</pre> <pre>  &lt;address&gt;[+&lt;size&gt;][,...]</pre> <pre>START = [( &lt;section name&gt;</pre> <pre>  [{ :   , }&lt;section name&gt;[,...]]</pre> <pre>  )][,...] [ /&lt;address&gt;] [,...]</pre>		
<pre>FSymbol = &lt;section name&gt;[,...]</pre>	Define section arrangement in memory.	Done in linker configuration file with the place in directive. Read more in EWRX Development guide <i>Linking using ILINK</i> .
<pre>ALIGNED_SECTION = &lt;section</pre> <pre>name&gt;[,...]</pre>	Outputs externally defined symbol addresses to a definition file.	-
<pre>ALIGNED_SECTION = &lt;section</pre> <pre>name&gt;[,...]</pre>	Changes the section alignment value to 16 bytes.	-
<pre>CPu = { &lt;cpu information file</pre> <pre>name&gt;</pre> <pre>  &lt;memory type&gt; =</pre> <pre>  &lt;address range&gt;[,...]</pre> <pre>  STRIDE}</pre>	Specifies a specifiable allocation range for section addresses.	-
<pre>PS_check= &lt;start address&gt; - &lt;end</pre> <pre>address&gt; , &lt;start address&gt; -&lt;end</pre> <pre>address&gt; [,...]</pre>	Specifies address ranges that might overlap each other in the physical space.	-
<pre>CONTIGUOUS_SECTION</pre> <pre>  = &lt;section name&gt;[,...]</pre>	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.	-
<pre>REName = {&lt;file name&gt;</pre> <pre>  (&lt;name&gt;=&lt;name&gt;[,...])</pre> <pre>    &lt;module name&gt;</pre> <pre>  (&lt;name&gt;=&lt;name&gt;[,...]) }</pre> <pre>[,...]</pre>	Symbol or section name modification	--redirect <from_symbol>=<to_symbol>
<pre>DElete = {&lt;module name&gt;</pre> <pre>    [ &lt;file name&gt;]</pre> <pre>  (&lt;name&gt;[,...]) } [,...]</pre>	Deletes a symbol name or module name.	-
<pre>REplace = &lt;file&gt;[(&lt;module&gt;[,...])</pre> <pre>] [,...]</pre>	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
<pre>CHange_message={Information  </pre> <pre>Warning   Error } [=&lt;error number&gt;</pre> <pre>[-&lt;error number&gt;] [,... ] [,...]</pre>	Modifies message levels.	<pre>--diag_error=tag [,tag,...]</pre> <pre>--diag_remark=tag [,tag,...]</pre> <pre>--diag_suppress=tag</pre> <pre>[,tag,...]</pre> <pre>--diag_warning=tag [,tag,...]</pre>
Hide	Deletes local symbol name information	-
Total_size	Sends total sizes of sections after linkage to standard output.	-
<b>Segments/Sections</b>		
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	.data32.data_init /



## Migrating from Renesas HEW for RX to IAR Embedded Workbench for RX

	“D”, alignment 4/2/1 byte	.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		IAR Embedded Workbench
<b>Calling convention</b>		
<b>Parameters passed on the stack</b>		
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.
<b>Parameters passed in registers</b>		
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
<b>Return values</b>		
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
<b>Preserved registers</b>		
R0, R6-R13		R0, R6-R13
<b>Scratch registers</b>		
R1-R5, R14, R15		R1-R5, R14, R15
<b>System startup and exit code</b>		
<p>The system startup code is located in <code>resetprg.c</code> and uses <code>dbsect.c</code>.</p> <p>Customized hardware initialization can be placed in the function <code>HardwareSetup()</code> in the file <code>hwsetup.c</code>.</p> <p>Interrupt vectors and interrupt functions are predefined for all possible interrupt sources. These can be found in <code>intprg.c</code> and <code>vecttbl.c</code>.</p>		<p>The system startup code is located in the ready-made <code>cstartup.s</code> 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 <code>cstartup</code>.</p> <p>You can do this by providing a customized version of the routine <code>__low_level_init</code>, which is called from <code>cstartup</code> before the data segments are initialized. Modifying <code>cstartup</code> directly should be avoided.</p>
<b>Global variable initialization</b>		
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

	<p><code>__low_level_init</code> function.</p> <p>Variables declared <code>__no_init</code> which are not initialized at all:</p> <p><code>__no_init int i;</code></p>
<b>Reentrancy and recursive functions</b>	
The library generator has an option to generate reentrant code or not.	The compiler is always reentrant when using the DLIB library.

IAR, IAR Systems, IAR Embedded Workbench, C-SPY, visualState, The Code to Success, IAR KickStart Kit, IAR, and the logotype of IAR Systems are trademarks or registered trademarks owned by IAR Systems.

All information is subject to change without notice. IAR Systems assumes no responsibility for errors and shall not be liable for any damage or expenses.

© 2013 IAR Systems AB. Part number: EWRX\_MigratingFromRenesasHEWRX. 1st edition: April 2013