Customer Notification

EWRL78 V2.xx/V3.xx/V4.xx/V5.xx

Embedded Workbench® for RL78 V2.xx/V3.xx/V4.xx/V5.xx

Operating Precautions

Y-IAR-EWRL78-FULL-MOBILE
Y-IAR-EWRL78-FULL

Renesas Electronics
www.renesas.com

Document No. R20UT3407ED0141
Date Published: February 2024

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.

2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.

4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.

5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.

6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.

7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.
   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti- crime systems; safety equipment; and medical equipment not specifically designed for life support.
   “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.

8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.

9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.

10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.

12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority- owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

Table of Contents

A) Table of Operating Precautions for the IDE EWRL78

No.	Outline	Version	EWRL78								
			8.1.4.5777 (4.10.1)	8.4.0.6247 (4.20.1)	8.4.2.6370 (4.20.2)	8.5.2.7561 (4.21.1)	8.5.2.7561 (4.21.2)	8.5.2.7561 (4.21.3)	9.1.7.10450 (5.10.1)	9.1.9.10638 (5.10.3)	9.1.9.10638 (5.10.4)
A1	Wrong flash mirror configuration for the device family RL78/G10		×	✓	✓	✓	✓	✓	✓	✓	×
A2	Wrong Section-Name in Linker-Control-File-Templates		✓	✓	✓	✓	✓	✓	✓	✓	×
A3	Wrong mirror end address		✓	✓	✓	✓	✓	✓	✓	✓	×
A4	Mirror Area Size not checked		×	✓	✓	✓	✓	✓	×	×	×
A5	Loading of *.ipcf file generates warnings		✓	✓	✓	✓	✓	✓	✓	✓	×
A6	The symbol _NEAR_CONST_LOCATION_SIZE will be wrong calculated if Mirror ROM 1 is selected.		✓	✓	✓	✓	✓	✓	✓	✓	×
A7	MISRA C violation on Assembler module		×	×	×	×	×	×	×	×	×
A8	The ROM mirror area size is one byte smaller when using the IAR Embedded Workbench		✓	✓	✓	✓	✓	✓	✓	✓	×
A9	Library Configuration tab under General Options displays the old style of library names		×	✓	✓	✓	✓	✓	✓	✓	×
A10	Two windows might become invisible		×	×	×	×	×	×	×	×	×
A11	The Renesas E2 self utility does not work		×	×	×	✓	✓	✓	✓	✓	×
A12	FAA code and data areas in RAM are not reserved		×	×	×	✓	✓	✓	✓	✓	✓

×: Applicable ✓: Not applicable -: Not checked

B) Table of Operating Precautions for the Assembler ARL78

No.	Outline	Version	ARL78									
			3.10.1	4.10.1	4.20.1	4.20.2	4.21.1	4.21.2	4.21.3	4.21.4	5.10.1	5.10.3
B1	**RSEG Directives cannot be used in Macro Definitions**		×	×	×	×	×	×	×	×	×	×
B2	**Assembler File must contain at least one Directive**		×	×	×	×	×	×	×	×	×	×

×: Applicable ✓: Not applicable **-** : Not checked

C) Table of Operating Precautions for C/C++ Compiler ICCRL78

No.	Outline		ICCRL78									
		Version	3.10.1	4.10.1	4.20.1	4.20.2	4.21.1	4.21.2	4.21.3	4.21.4	5.10.1	5.10.3
C1	Internal Compiler Error: Stack Overflow		×	×	×	×	×	×	×	×	×	×
C2	Internal Compiler Error: Size mismatch		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C3	Internal Compiler Error: Bad Operator		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C4	Scratch Registers are not saved in Interrupt Service Routine		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C5	Internal Compiler Error: Illegal State		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C6	Wrong Code may generated for Instructions using Operand imm[BC]		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C7	Inconsistency of extended Keyword __monitor		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C8	Floating point comparison fails if the difference between the operands is one bit only.		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C9	An internal error will be generated in case of accessing a section address by using sfe		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C10	An internal error will be generated in case of sequential pointer casting		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C11	Wrong Optimization of static local Variable		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C12	Inserted NOP after DIVWU/DIVHU Instruction moved (cross call optimization)		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C13	The C library function isblank(c) will in some cases erroneously return true		×	✓	✓	✓	✓	✓	✓	✓	✓	✓
C14	Switch statement inside recursive function does not work correctly		×	✓	✓	✓	✓	✓	✓	✓	✓	✓
C15	Error in case a simple character literal is followed by a wide character literal		×	✓	✓	✓	✓	✓	✓	✓	✓	✓
C16	Range error on nextXXX() functions		×	✓	✓	✓	✓	✓	✓	✓	✓	✓
C17	No output to stdout when putchar(-1) is used		×	✓	✓	✓	✓	✓	✓	✓	✓	✓
C18	Different return value between iswctype and iswblank		×	✓	✓	✓	✓	✓	✓	✓	✓	✓
C19	%Z format output for strftime is wrong		×	×	×	×	×	×	×	×	×	×
C20	Square root function in the floating point library returns +0.0 for sqrt(-0.0)		×	✓	✓	✓	✓	✓	✓	✓	✓	✓
C21	errno() might cause a range error		×	✓	✓	✓	✓	✓	✓	✓	✓	✓

No.	Outline		ICCRL78									
		Version	3.10.1	4.10.1	4.20.1	4.20.2	4.21.1	4.21.2	4.21.3	4.21.4	5.10.1	5.10.3
C22	Wrong result in case of Complex_I multiplication with -0.0		×	×	×	×	×	×	×	×	×	×
C23	Function cosh() does not set errno()		×	×	×	×	×	×	×	×	×	×
C24	A const long long int array element value is not referenced correctly		×	✓	✓	✓	✓	✓	✓	✓	✓	✓
C25	If there are multiple if-statements that refer to function argument values, value judgment is incorrect.		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C26	A long long int array element value with auto storage duration is not referenced correctly.		×	×	✓	✓	✓	✓	✓	✓	✓	✓
C27	A long long int array element value is not referenced using the const pointer correctly within the for-statement.		×	✓	✓	✓	✓	✓	✓	✓	✓	✓
C28	printf outputs nothing after long long int two-dimension arrays operation		×	✓	✓	✓	✓	✓	✓	✓	✓	✓
C29	Internal Compiler Error: Double Defined Interrupt Vector		×	✓	✓	✓	✓	✓	✓	✓	✓	✓
C30	Files based on the UTF-8 (BOM) format cannot be compiled		×	✓	✓	✓	✓	✓	✓	✓	✓	✓
C31	Compiler can generate faulty code for 8-bit logical and arithmetic operations		×	✓	✓	✓	✓	✓	✓	✓	✓	✓
C32	Data model will be ignored in case of using #pragma constseg		×	✓	✓	✓	✓	✓	✓	✓	✓	✓
C33	Inline Assembler instruction generates an illegal syntax error		×	×	✓	✓	✓	✓	✓	✓	✓	✓
C34	Error in floating point division		×	×	✓	✓	✓	✓	✓	✓	✓	✓
C35	Memory dependency problem		×	×	✓	✓	✓	✓	✓	✓	✓	✓
C36	Casting two far pointers to long integer and saving the difference will result in a wrong subtraction		✓	✓	×	✓	✓	✓	✓	✓	✓	✓
C37	Internal error will be thrown in case optimization "Function inlining" is activated		×	×	×	✓	✓	✓	✓	✓	✓	✓
C38	Incorrect code will be generated if Compiler optimization “Common subexpression elimination” is active		✓	×	×	✓	✓	✓	✓	✓	✓	✓
C39	C++ Compiler can generate incorrect code for comparisons of floating-point numbers		×	×	×	✓	✓	✓	✓	✓	✓	✓
C40	Byte order of the offset in the opcode for MOVW offset[BC/B/C],AX is swapped.		×	×	×	×	✓	✓	✓	✓	✓	✓
C41	long long operations which are using the __Mul64 function are not reentrant		×	✓	✓	✓	✓	✓	✓	✓	✓	✓

No.	Outline		ICCRL78									
		Version	3.10.1	4.10.1	4.20.1	4.20.2	4.21.1	4.21.2	4.21.3	4.21.4	5.10.1	5.10.3
C42	**Faulty code for switches if the code for the switch and its associated cases span across a 64k border**		✗	✗	✗	✗	✗	✗	✗	✗	✓	✓
C43	**Constants located outside of the near area (flash mirror) cannot be used as parameter for printf function**		✗	✗	✗	✗	✗	✗	✗	✗	✓	✓
C44	**Copying several bits (1-bit bitfields) in sequence to the same destination byte can generate faulty code on optimization level medium or higher.**		✓	✓	✗	✗	✗	✗	✓	✓	✓	✓
C45	**The tools crash when try to enter the debugger with E1 or E20.**		✓	✓	✓	✓	✓	✓	✗	✓	✓	✓
C46	**Accessing the SFR area (0xFFF00) via near_pointer+index throws an error in case of Compiler optimization “Medium” and higher.**		✗	✗	✗	✗	✗	✗	✗	✗	✓	✓

✗: Applicable ✓: Not applicable **-** : Not checked

D) Table of Operating Precautions for the Linker ILINKRL78 and ELF-Tools

No.	Outline		ILINKRL78 and ELF-Tools									
		Version	3.10.1	4.10.1	4.20.1	4.20.2	4.21.1	4.21.2	4.21.3	4.21.4	5.10.1	5.10.3
D1	**Runtime Model Conflict using far Runtime-Library-Calls**		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
D2	**Area in ROM marked as read-write-data in MAP-File**		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
D3	**Routines for HW-Multiplier/Division Unit don't support far runtime library calls**		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
D4	**Internal error will be thrown if the section to be copied by "initialize manually" or "initialize by copy" feature is not placed**		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
D5	**The symbol _NEAR_CONST_LOCATION_SIZE will be wrong calculated if Mirror ROM 1 is selected.**		Now listed as IDE bug. See No. A6									
D6	**Constant Data with Memory Attribute 'near' are treated as 'readwrite' Data in the Map File**		×	✓	✓	✓	✓	✓	✓	✓	✓	✓
D7	**The linker does not issue a warning if more than one interrupt function uses the same interrupt vector**		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
D8	**Switch/case statement over 64KB page**		×	✓	✓	✓	✓	✓	✓	✓	✓	✓
D9	**__sfb() returns wrong address for section .text**		✓	×	×	×	×	×	×	×	×	×
D10	**End address of SADDR region is wrong**		×	×	✓	✓	✓	✓	✓	✓	✓	✓
D11	**Linker can terminate with an internal error if other (relevant) linking errors are present**		×	×	×	×	✓	✓	✓	✓	✓	✓
D12	**Internal error will be thrown in during linking object files generated by the Renesas compiler**		×	×	×	×	✓	✓	✓	✓	✓	✓
D13	**Error will be thrown by using the ielftool and filler bytes**		×	×	×	×	✓	✓	✓	✓	✓	✓
D14	**EWRL78 ielftool --fill option doesn't fill the first byte of unused area if the END address defined for --fill points**		×	×	×	×	×	×	×	×	✓	✓
D15	**Hardware multiplication replacement routines for 64-bit cannot be placed above the address 0xFFFF**		-	-	-	-	-	-	-	-	×	✓
D16	**Initializing data can fail if the block to copy is too big to fit inside the first available block in ROM**		×	×	×	×	×	×	×	×	×	✓

×: Applicable ✓: Not applicable **-** : Not checked

E) Table of Operating Precautions for Debugger C-SPY

No.	Outline		C-SPY										
		Version	3.10.1	4.10.1	4.20.1	4.20.2	4.21.1	4.21.2	4.21.3	4.21.4	5.10.1	5.10.3	5.10.4
E1	E1 C-SPY Driver: Debug Session closed after Error 'Flash macro service ROM accessed or stepped in'		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
E2	The C-SPY system macro __setLogBreak() does not work for emulators		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
E3	IECUBE C-SPY Driver: Wrong average timer results		✓	✓	✓	✓	✓	✓	✓	✓	-	-	-
E4	E1 C-SPY Driver: Step-in Step over doesn't work for switch case construct with more than 99 cases		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
E5	E1 C-SPY Driver: Specifying the serial number for the E1/E20 emulator sometime causes it not to be found.		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
E6	Wrong sampled values might be shown in the Data Sample/Sampled Graphs window in case of sampling a variable with a size of 2 bytes		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
E7	E1 C-SPY Driver: Download of an additional image might destroy a part of the original application.		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
E8	Data sampling time not constant		✗	✗	✗	✗	✗	✗	✗	✗	-	-	-
E9	Min. update interval value for Live Watch and Memory Window is wrong		✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
E10	Binary image not showing symbol info in “Disassembly” window		✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
E11	Simulator interrupts can go wrong if code is above 64KB		✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
E12	Simulator interrupts have wrong priority levels in case of shared vector		✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
E13	E1/E2 C-SPY Driver: OCD Trace automatically disabled in case Step-in/Step-over is used.		✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓
E14	EWRL78 hanged-up while power debugging		✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
E15	In some situations, the debugger crashes when using an OCD emulator.		✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
E16	Debugging via hot-plugin doesn’t work		✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓
E17	Simulator result of MACH instruction is wrong		✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓

E18	**Loading of extra images fails if there is data in the EEPROM memory in the extra image.**	✘	✘	✘	✘	✘	✘	✘	✘	✘	✓	✓
E19	**Several registers are missing in the Register view. E.g. the register “ADCSR”**	✓	✓	✓	✓	✓	✓	✘	✘	✘	✘	✘
E20	**FAA breakpoints cannot be removed after setting**	-	-	-	-	-	-	-	-	-	✘	✓
E21	**The execution speed decreases by approximately half when activate FAA debugging**	-	-	-	-	-	-	-	-	-	✘	✓
E22	**The C-SPY system macro __setCodeBreak does not work with the emulator.**	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✓

✘: Applicable ✓: Not applicable - : Not checked

F) Table of Operating Precautions for Runtime Library, Linker Files and Include Files

No.	Outline	Version	Runtime Library, Linker Files and Include Files										
			2.21.2	3.10.1	4.10.1	4.20.1	4.20.2	4.21.1	4.21.2	4.21.3	4.21.4	5.10.1	5.10.3
F1	Range error will be thrown by the linker in case of using the “far runtime library calls” feature in combination with integer arithmetic libraries		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
F2	If the option “Use far runtime library calls” is used, assembler support routines from the run time library are placed in an incorrect section called “.ftext”		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
F3	The section name .textf_unit64kp is misspelled in the linker files		×	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
F5	Overlapping of two registers		×	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
F6	In rare cases, the linker might fail with the following internal error		×	×	✓	✓	✓	✓	✓	✓	✓	✓	✓
F7	Libraries generated with IAR V2.xx version cannot be linked on newer IAR versions if the library includes a vector table.		✓	×	×	×	×	×	×	×	×	×	×

×: Applicable ✓: Not applicable - : Not checked

G) Description of Operating Precautions for the IDE EWRL78

No. *A1*	**Wrong flash mirror configuration for the device family RL78/G10**

IAR Reference EW26031

Details
The flash mirror area of the device family RL78/G10 is specified as follows:

0000H to 003FFH/007FFH/00FFFH mirrored to the area F8000H to F83FFH/F87FFH/F8FFFH

However, by using the IDE the flash mirror configuration is wrong. The flash mirror address starts from the address 8000H instead of 0000H

8000H to 083FFH/087FFH/08FFFH mirrored to the area F8000H to F83FFH/F87FFH/F8FFFH

Workaround

Workaround via command line:

Define the linker symbol _NEAR_CONST_LOCATION_START manually within the linker file and build the application from the command line.

define symbol _NEAR_CONST_LOCATION_START=0x00D8;
define symbol _NEAR_CONST_LOCATION_SIZE=0x100;

Workaround via IDE:

1) Define new linker symbols like e.g. _MY_NEAR_CONST_LOCATION_START and _MY_NEAR_CONST_LOCATION_SIZE within your linker file
2) Rename all the symbols _NEAR_CONST_LOCATION_START and _NEAR_CONST_LOCATION_SIZE within the linker file according to the new names

No. *A2*	**Wrong Section-Name in Linker-Control-File-Templates**

IAR Reference EWRL78-511

Details
The section name .textf_unit64kp is misspelled in the linker-control-file-templates (*.icf):

Example:

```
 "ROMFAR":place in ROM_far { block INIT_ARRAY,
                             R_TEXTF_UNIT64KP,
                             ro section .text_unit64kp,
                             ro section .constf,
                             ro section .switchf,
                             ro };
```

Workaround

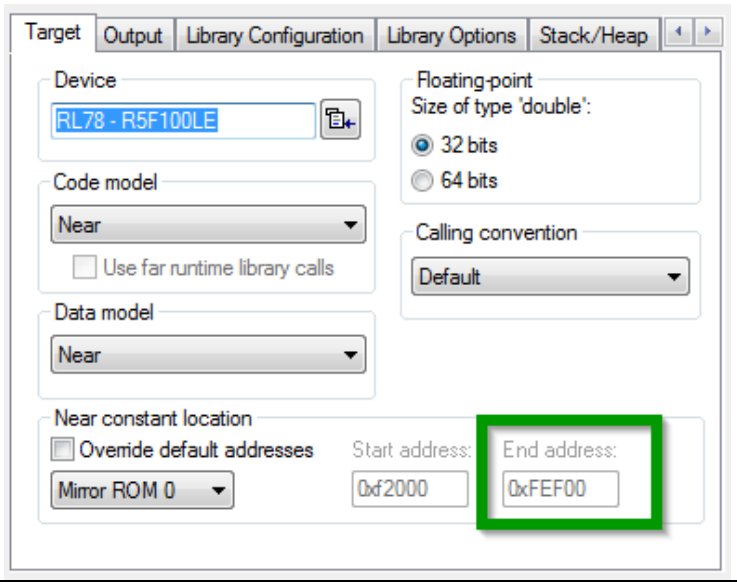
Correct the section name manually:

```
"ROMFAR":place in ROM_far { block INIT_ARRAY,
                            R_TEXTF_UNIT64KP,
                            ro section .textf_unit64kp,
                            ro section .constf,
                            ro section .switchf,
                            ro };
```

No. *A3*	**Wrong mirror end address**

IAR Reference EWRL78-541

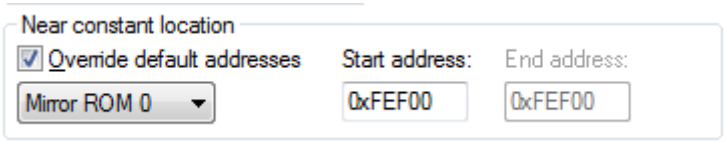
Details
The mirror area end address displayed in the project settings dialog is set to the real end-address plus one instead of the just the end-address. This does not affect the linked code which uses correct values.



No. *A4*	**Mirror Area Size not checked**

IAR Reference EWRL78-558

Details
The size check of the mirror area is ignored if the size is set to zero. The linker allows to place near-variables without checking the size size of the mirror area.

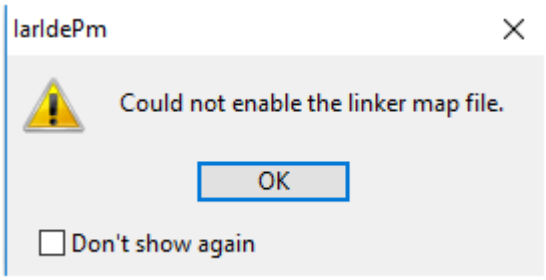


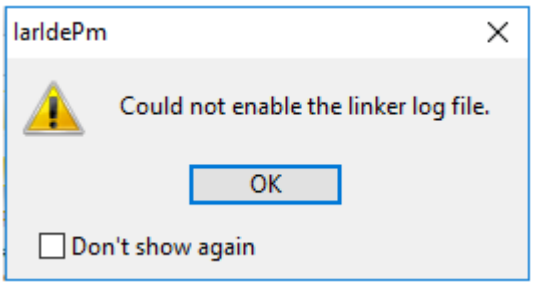
Workaround
If enough Flash memory is available, don't use a mirror area of size zero.

No. *A5*	**Loading of .ipcf file generates warnings**

IAR Reference IDE-2878

Details
During the load procedure "Add Project Connection..." of an *.ipcf file the following warnings might occur:



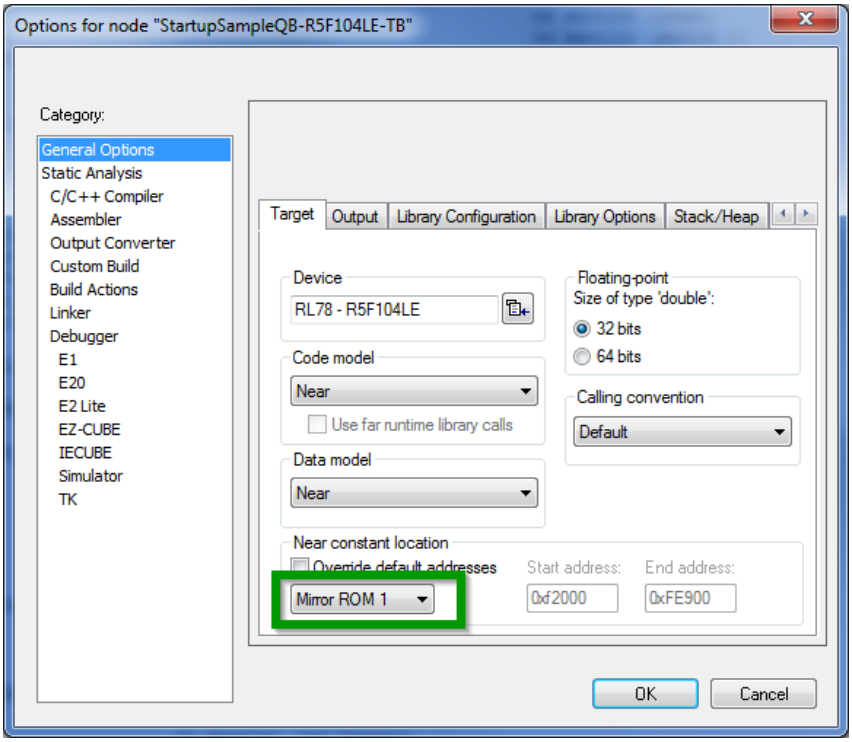


Workaround
Press the "OK" button and ignore the messages.

No. A6	**The symbol _NEAR_CONST_LOCATION_SIZE will be wrong calculated if Mirror ROM 1 is selected**

IAR Reference EWRL78-540

Details
The symbol _NEAR_CONST_LOCATION_SIZE will be wrong calculated if Mirror ROM 1 is selected as shown below:



Workaround
Replace _NEAR_CONST_LOCATION_SIZE in the linker file at the following two places with the correct size.

```
define block MIRROR_ROM with maximum size = _NEAR_CONST_LOCATION_SIZE
{ ro R_CONST_init, ro section .const_init, ro section .switch_init };

define block MIRROR_RAM with maximum size = _NEAR_CONST_LOCATION_SIZE
{ rw R_CONST, rw section .const, rw section .switch };
```

Example:

The size for the above screenshot values shall be calculated as follows
_NEAR_CONST_LOCATION_SIZE = 0xFE900 – 0xF2000 = 0xC900

The define _NEAR_CONST_LOCATION_SIZE shall be replaced within the linker file as follows:

```
define block MIRROR_ROM with maximum size = 0xC900 { ro R_CONST_init,
ro section .const_init, ro section .switch_init };

define block MIRROR_RAM with maximum size = 0xC900 { rw R_CONST, rw
section .const, rw section .switch };
```

No. A7	**MISRA C violation on Assembler module**

IAR Reference EWRL78-636

Details
The cstartup.s file provided by IAR as a template includes two symbol definitions @cstart and @cend. These symbols are not referenced by the application, but they will be used by the Renesas CS+ debugger in order to identify the start and the end of the application.

In case the source file cstartup.s is added to the application and the MISRA C checker is activated an error for the MISRA-C 2004 rule 8.10 will be thrown, because a symbol is defined in an assembler module but is not referenced.

Workaround
If the Renesas CS+ debugger is not used for debugging you can remove the definition of the symbols @cstart and @cend from the cstartup.s file. Otherwise, treat that error as a warning and document why the MISRA C error appears.

No. A8	**The ROM mirror area size is one byte smaller when using the IAR Embedded Workbench**

IAR Reference EWRL78-673

Details
The ROM mirror area size is one byte smaller when using the IAR Embedded Workbench.

Workaround
Replace _NEAR_CONST_LOCATION_SIZE in the linker file at the following two places with the correct size.

define block MIRROR_ROM with maximum size = _NEAR_CONST_LOCATION_SIZE { ro R_CONST_init, ro section .const_init, ro section .switch_init };

define block MIRROR_RAM with maximum size = _NEAR_CONST_LOCATION_SIZE { rw R_CONST, rw section .const, rw section .switch };

Example:

The mirror size for the R5F104LE shall be 0xC900. However, in case of using the IDE for the build the size is one byte smaller:
_NEAR_CONST_LOCATION_SIZE = 0xC8FF

The symbol _NEAR_CONST_LOCATION_SIZE shall be replaced within the linker file as follows:

define block MIRROR_ROM with maximum size = 0xC900 { ro R_CONST_init, ro section .const_init, ro section .switch_init };

define block MIRROR_RAM with maximum size = 0xC900 { rw R_CONST, rw section .const, rw section .switch };

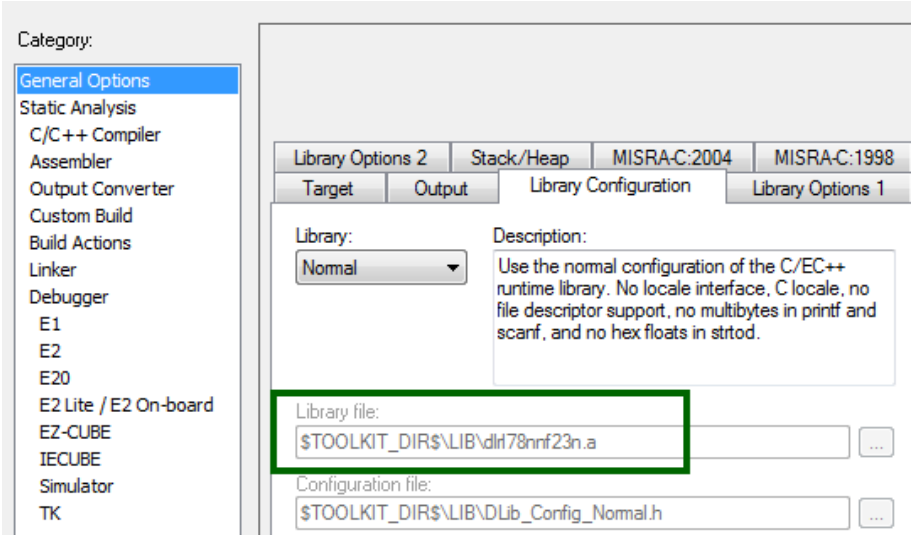
No. A9	**Library Configuration tab under General Options displays the old style of library names**

IAR Reference EWRL78-763

Details
The Library Configuration tab under General Options displays the old style of library names instead of the new one used by V4.10

Example:

The library file in the following example should be dlrlnnf23n.a instead of dlrl**78**nnf23n.a:



Workaround
Please ignore the displayed library file. Internally the correct file will be used.

No. A10	**Two windows might become invisible**

IAR Reference EWRL78-775/IDE-4531

Details
If you undock two windows in the debugger and, put them outside the IDE and, put them together and, leave the debugger, these windows become invisible and cannot be used for sub-sequent debug sessions.

Workaround
None

No. A11	**The Renesas E2 self utility does not work**

IAR Reference EWRL78-799

Details
The Renesas E2 self utility does not work with the E2.

Workaround
Download the latest E2 self utility (E2SCP_Vxxxx.exe) via the following link:
http://www.renesas.eu/update?oc=RTE0T00020KCE00000R#packageInfo

No. A12	**FAA code and data areas in RAM are not reserved**

IAR Reference: EWRL78-1097

Details

The FAA code and data areas in RAM are not reserved in the linker configuration files in a way that it guarantees that user's variables cannot overwrite.

Workaround

None

H) Description of Operating Precautions for the Assembler ARL78

No. B1	**RSEG Directives cannot be used in Macro Definitions**

Details

The assembler calculates a wrong relative jump-distance if the RSEG directive is used within a macro definition:

Example

```
myDummyMacro MACRO
        RSEG    CODE:CODE
        NOP
        ENDM
```

Workaround

Don’t use the RSEG directive in macro definitions. The used code-segment must be defined in the code where the macro is expanded to.

No. B2	**Assembler File must contain at least one Directive**

Details

An assembler module without any assembler directive causes the following error message:

```
Error[As074]: Each file must contain at least one directive
```

Example

```
#if PLATFORM == RL78
   ; section without directive
#else
   ; section without directive
#endif
```

Workaround

Please use the END directive:

```
#if PLATFORM == RL78
   ; section code
   END
#else
   ; section code
   END
#endif
```

I) Description of Operating Precautions for the C/C++ Compiler ICCRL78

No. C1	**Internal Compiler Error: Stack Overflow**

IAR Reference: EW24353

Details

Very deep nestlings of structure declarations, parenthesis or if-else statements, may generate a stack overflow error in the compiler.

```
Internal Error:
[CoreUtil/General]: Stack overflow (0xXXXXXXXX) at xxxxxxxx
```

Examples

1)

```
#define LBR1 ( ( ( ( ( ( ( ( (
#define LBR2 LBR1 LBR1 LBR1 LBR1 LBR1 LBR1 LBR1 LBR1 LBR1 LBR1
#define LBR3 LBR2 LBR2 LBR2 LBR2 LBR2 LBR2 LBR2 LBR2 LBR2 LBR2
#define LBR4 LBR3 LBR3 LBR3 LBR3 LBR3 LBR3 LBR3 LBR3 LBR3 LBR3
#define RBR1 ) ) ) ) ) ) ) ) )
#define RBR2 RBR1 RBR1 RBR1 RBR1 RBR1 RBR1 RBR1 RBR1 RBR1 RBR1
#define RBR3 RBR2 RBR2 RBR2 RBR2 RBR2 RBR2 RBR2 RBR2 RBR2 RBR2
#define RBR4 RBR3 RBR3 RBR3 RBR3 RBR3 RBR3 RBR3 RBR3 RBR3 RBR3

int q5_var = LBR4 0 RBR4;
```

2)

```
#define ONE   else if (0) { }
#define TEN   ONE ONE ONE ONE ONE ONE ONE ONE ONE ONE
#define HUN   TEN TEN TEN TEN TEN TEN TEN TEN TEN TEN
#define THOU  HUN HUN HUN HUN HUN HUN HUN HUN HUN HUN

void foo()
{
  if (0) { }
  THOU THOU THOU THOU THOU THOU THOU THOU THOU THOU THOU
}
```

Workaround

Avoid such code, this will be listed as a known problem.

No. C2	**Internal Compiler Error: Size mismatch**

IAR Reference: EW25533

Details

Reading a 16-bit SFR that is located between 0xFFF00-0xFFF1F might generate an internal error :

```
Internal error:
[CoreUtil/General]: Size mismatch for “   MOVW  HL, S:0xFFFxx  ;; 1
cycle, inserted as 3 bytes, assembled as 2 bytes.
```

Examples

```
#include <ior5f10ppj.h>

unsigned short v1[10];
unsigned char  v2;

void test(void)
{
    v1[v2] = ADCR;
}
```

Workaround

Use a static temporary variable:

```
void test(void)
{
    static unsigned short dummy;
    dummy = ADCR;
    v1[v2] = dummy;
}
```

The issue will be fixed in future update.

No. C3	**Internal Compiler Error: Bad Operator**

IAR Reference: EW25541

Details

In case of using explicit double casting, an internal compiler error occurs:

```
Internal error:
[GoBinaryExprCvm::Evaluate]: bad operator
```

Example

```
void test ( void )
{
 (void)(unsigned short int)( (*(unsigned short *)0xF06E6) );
}
```

Workaround

Either remove the `(void)` cast or make the pointer cast volatile:

```
(void)(unsigned short int)( (*(unsigned short volatile *)0xF06E6) )
```

No. C4	**Scratch Registers are not saved in Interrupt Service Routine**

IAR Reference: EW25593

Details

Interrupt service routines using the new Renesas calling convention (v2) fail to save the scratch registers. This occurs independently of the used optimization.

Example

```
__far const unsigned char   data[] = { 0xfa, 0xfa, 0xfa};
             unsigned long v1;

#pragma vector = 0x7A
__interrupt void isr01(void)
{
    v1 = (unsigned long)&data[0];
}
```

Workaround

Change the calling convention of the for the interrupt service routine:

```
__v1_call __interrupt void isr01(void)
{
   v1 = (unsigned long)&data[0];
}
```

No. C5	**Internal Compiler Error: Illegal State**

IAR Reference: EW25713

Details

Far pointers that have a constant value (known at compile time) that points into the short address area can generate an internal error.

Example

```
typedef union {
    struct {
        unsigned char   p10     :1;
        unsigned char   p11     :1;
        unsigned char   reserve1:6;
    } ;
    unsigned char   all;
} SFRDEF;

typedef union {
    SFRDEF  byte;
} SFR;


__far SFR sfr @(0xFFF01) ;

void test( void )
{
        sfr.byte.p10  = 0;
        sfr.byte.all  = 0;
}
```

Workaround

Avoid absolute addressing by using a user defined data segment:

```
#pragma dataseg= __saddr MySeg
__far SFR sfr;
#pragma dataseg= default
```

No. C6	**Wrong Code may generated for Instructions using Operand imm[BC]**

IAR Reference: EW25763

Details

Instructions that have one operand of type imm[BC] can in some cases generate wrong offsets to BC if the offset is a constant (not a label).

Example

```
#define D (*((volatile T __near *)(0x1234)))

typedef struct {
    unsigned char c[10];
} T;

int i = 0;
int j;

void test( void )
{
  j      = D.c[i];
  //  wrong generated code:
  //  000004 49 3412      MOV       A, (0x1234)[BC]    ;; 1 cycle

 D.c[i] = j;
  //  correct code:
  //  000013 48 1234      MOV       (0x1234)[BC], A    ;; 1 cycle

}
```

Workaround

None.

No. C7	**Inconsistency of extended Keyword __monitor**

IAR Reference: EW25971

Details

Using IAR function object attributes (like __monitor) with member functions of template classes defined outside the class definition does not work properly. Specifying the attribute both on the declaration and the definition of the function results in a nonsensical error message ("declaration is incompatible with ...").

Example:

```
template <typename T, unsigned long Size>
class buffer
{
  __monitor void clear();
};

template <typename T, unsigned long Size>
__monitor void buffer<T, Size>::clear() {
  // ...
}
```

Workaround

None; it will be fixed in next update.

No. C8	**Floating point comparison fails if the difference between the operands is one bit only.**

IAR Reference: EW26007

Details

A floating point comparison fails if the difference between the operands is one bit only.

Example:

The following code should return 0, because the value of the expression (-16777215.0F <= -16777216.0F) is false. But it returns 1.

```
volatile float a;
const float t = -16777216.0F;

int main()
{
    int ret = 0;

    a = (-16777215.0F);
    if(a <= -16777216.0F) ret |= 1;
    if(a <= t)            ret |= 2;
    return ret;
}
```

Workaround

Compare with a (const) volatile variable or an external const variable instead of a constant.

No. C9	**An internal error will be generated in case of accessing a section address by using sfe**

IAR Reference: EW25997

Details

An internal error will be generated in case of accessing section address by using the sfe and inline Assembler. Following internal error will be thrown.

```
Internal Error: [CoreUtil/General]: Access violation (0xc0000005) at 0040997E
(reading from address 0x18) Internal
Error: [CoreUtil/General]: Access violation (0xc0000005) at 0040997E (reading
from address 0x18)
```

Example:

```
int main()
{
    asm("MOVW SP, #LWRD(sfe(""CSTACK""))");
}
```

Workaround

Use #pragma section before accessing section addresses:

```
#pragma section="CSTACK"
asm("MOVW SP, #LWRD(sfe(""CSTACK""))");
```

No. C10	**An internal error will be generated in case of sequential pointer casting**

IAR Reference: EWRL78-506

Details

An internal error can be generated in case of casting a near pointer to a short, then casting it to far pointer and then casting to a long, if optimization level medium or higher is used.

```
Internal Error: [TaOpPrefix::GetWordIndex]:
Diagnostics: Not implemented yet)
```

Example:

```
unsigned long l;
char __near np;

void test()
{
  l = (unsigned long) (void __far *) (unsigned short) &np;
}
```

Workaround

Avoid pointer casting sequence or reduce optimization level for the function by using #pragma optimize.

No. C11	**Wrong Optimization of static local Variable**

IAR Reference: EWRL78-547

Details

At optimization level ‘high’, static local variables assigned only the constants 0 and 1, but initialized with another value, can be optimized incorrectly.

Example:

```
typedef enum {
  tt1 = 0,
  tt2,
  tInvalid
} tMyTpe;

int g1, g2;

void test()
{
   static tMyTupe v1;

   if (g1 < g2)  && (v1 != tt2) {

   }
}
```

Workaround

Set initial start value of the first struct member to 1:

```
typedef enum {
  tt1 = 1,
  tt2,
  tInvalid
} tMyTpe;
```

No. C12	**Inserted NOP after DIVWU/DIVHU Instruction moved (cross call optimization)**

IAR Reference: EWRL78-576

Details

The compiler adds a NOP instruction for the RL78 S3 MCU core after every DIVWU and DIVHU instruction as a workaround for an error in the MCU. However, the cross call optimizer will in some cases move an instruction in between the DIVHU/DIVWU instruction and the NOP.

This happens only if cross call optimization is activated.

Example:
None

Workaround

Disable the cross call optimization by using the compiler option `--no_crosscall`

No. C13	**The C library function isblank(c) will in some cases erroneously return true**

IAR Reference: EW26558/EWRL78-584

Details

The C library function isblank(c) will in some cases erroneously return true for a few characters (\f, \n, \r and \v).

Example

```
if( isblank( '\v' ) ) {
  printf( "This line will be printed in case of wrong return value!!!" );
}
```

Workaround
None

No. C14	**Switch statement inside recursive function does not work correctly.**

IAR Reference: EW26549/EWRL78-585

Details

On optimization level -Om or higher the compiler can generate erroneous code for functions with a recursive call followed directly by a switch statement where one of the switch cases has the only effect that the function exits.

Example

```
#include <stdio.h>

int val = 0;
void func( int p )
{
  if( p > 0 ) {
    func( -1 );
    switch( val ) {
    case 0 :
      val = 1;
      break ;
    case 1 :
      val = 2;
      break ;
    default :
      break ;
    }
  }
}

int main( void )
{
  func( 1 );

  if( val != 1 ) {
    printf( "FAILED" );
  } else {
    printf( "OK");
  }
```

Workaround
None

No. C15	**Error in case a simple character literal is followed by a wide character literal**

IAR Reference: EW26564/EWRL78-587

Details

If the code contains a simple character literal followed by a wide character literal, an error is issued. See Example.

Example

```
wchar_t buf[] = L"1""2"  ;

Error:[Pe1282]: string literals with different character kinds cannot be
concatenated
```

Workaround
None

No. C16	**Range error on nextXXX() functions**

IAR Reference: EWRL78-603

Details

The range error occurs when the first argument of the following function is 0.0
nextafter / nextafterf / nextafterl / nexttoward / nexttowardf / nexttowardl.

Example

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <errno.h>

int main( void )
{
errno = 0 ;
nextafter( 0.0, 1.0 ) ;

if ( errno == 0 ) {
  printf( "OK" ) ;
} else {
  printf( "NG" ) ;
}

return( 0 ) ;
}
```

Workaround
None

No. C17	**No output to stdout when putchar(-1) is used**

IAR Reference: EWRL78-606

Details

The library function putchar() does not handle the input value -1 according to the standard. Instead of printing '\0377' (-1 casted to unsigned char) to stdout and return this value it does not output anything and returns -1.

Example

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <errno.h>

int main( void )
{
errno = 0 ;
nextafter( 0.0, 1.0 ) ;

if ( errno == 0 ) {
  printf( "OK" ) ;
} else {
  printf( "NG" ) ;
}

return( 0 ) ;
}
```

Workaround

Cast the parameter to unsigned char when calling putchar.

```
putchar((unsigned char)-1);
```

No. C18	**Different return value between iswctype and iswblank**

IAR Reference: EWRL78-602 / EW26582

Details

The return value of iswctype(wc, wctype("blank")) and the return value of iswblank(wc) are NOT same.

```
res1 = iswblank( L' ' ) ;                    // res1=1
res2 = iswctype( L' ', wctype( "blank" ) ) ;  // res2=0
```

IAO/IEC9899:1999 describes that iswctype(wc, wctype("blank")) and iswblank(wc) have the same return value.

++++
IAO/IEC9899:1999 : 7.25.2.2.1 The iswctype function Each of the following expressions has a truth-value equivalent to the call to the wide character classification function (7.25.2.1) in the comment that follows the expression:
iswctype(wc, wctype("blank")) // iswblank(wc)
++++

Example

```
#include <stdio.h>
#include <wctype.h>

int main( void )
{
int res1, res2 ;

res1 = iswblank( L' ' ) ;
res2 = iswctype( L' ', wctype( "blank" ) ) ;

if( res1 != res2 ) {
  printf( "NG") ;
} else {
  printf( OK" ) ;
}

return( 0 ) ;
}
```

Workaround

None

No. C19	**%Z format output for strftime is wrong**

IAR Reference: EWRL78-605 / EW26595

Details

By default the character ":" is used as a replacement for %Z if the application has not implemented time zone handling. However, here the value 0x00 will be written instead of 0x3A ":".

Example

```
#include <stdio.h>
#include <time.h>
#include <string.h>

int main( void )
{
 char   expected[] = ":" ;
 char   result[100] ;
 struct tm input ;

 input.tm_sec = 0 ;
 input.tm_min = 0 ;
 input.tm_hour = 0 ;
 input.tm_mday = 1 ;
 input.tm_mon  = 0 ;
 input.tm_year = 0 ;
 input.tm_wday = 0 ;
 input.tm_yday = 0 ;
 input.tm_isdst = 0 ;

 strftime( result, 100, "%Z", &input ) ;
 if( strcmp( result, expected ) == 0 ) {
   printf( "OK" ) ;
 } else {
   printf( "NG" ) ;
 }

 return( 0 ) ;
}
```

Workaround

None

No. C20	**Square root function in the floating point library returns +0.0 for sqrt(-0.0)**

IAR Reference: EWRL78-607 / EW26605

Details

The square root function in the floating point library returns +0.0 for sqrt(-0.0) and not -0.0 as the standard specifies.

Example

```
#include <stdio.h>
#include <math.h>

volatile float sqrt_result;
float compare_value = -0.0f;

unsigned long int * value_1 = (unsigned long int *)&sqrt_result;
unsigned long int * value_2 = (unsigned long int *)&compare_value;


int main( void )
{

  sqrt_result = sqrt(-0.0f);

  if( *value_1 == *value_2 ){
    printf( "OK" );
  } else {
    printf( "NG" );
  }
return( 0 ) ;
}
```

Workaround

None

No. C21	**errno() might cause a range error**

IAR Reference: EWRL78-604 / EW26577

Details

errno() might cause a range error if the first argument to a function is ±DBL_MIN and the sign of the second argument is opposite to the first argument.

Example

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <errno.h>
#include <float.h>

int main( void )
{
errno = 0 ;
nextafter( DBL_MIN, -0.1 ) ;

if ( errno == 0 ) {
  printf( "OK" ) ;
} else {
  printf( "NG" ) ;
}

return( 0 ) ;
}
```

Workaround
None

No. C22	**Wrong result in case of Complex_I multiplication with -0.0**

IAR Reference: EWRL78-601 / EW26599

Details

A multiplication of a real floating point type (r1) with a complex type will promote r1 to a complex type before the multiplication. This will produce undesirable results when infinite number, NaNs, or -0.0:s are involved. The same thing happens when you divide a complex type with a real floating type.

Example

```
#include <stdio.h>
#include <math.h>
#include <complex.h>
#include <string.h>

int main( void )
{
complex double d = -0.0 * _Complex_I ;
char real[10], image[10] ;

sprintf(  real, "%g", creal( d ) ) ;
sprintf( image, "%g", cimag( d ) ) ;

if( (strcmp( real, "-0" ) != 0) || (strcmp( image, "-0" ) != 0 ) ) {
  printf( "%-12s %04d:NG [-0][-0]--->[%s][%s]\n", __FILE__, __LINE__, real, image ) ;
} else {
  printf( "%-12s %04d:OK\n", __FILE__, __LINE__ ) ;
}

return( 0 ) ;
}
```

Workaround
None

No. C23	**Function cosh() does not set errno()**

IAR Reference: EWRL78-612 / EW26609

Details

The standard library function cosh() called with an infinite does not set errno() to EDOM (domain error) as expected.

Example

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <errno.h>
int main( void )
{
        double result ;
        union u_data {
                double d ;
                signed long dt[2] ;
        } pt = { 0.0 } ;
        errno = 0 ;
        pt.dt[1] |= 0x7ff00000ul  ; //
        result = cosh( pt.d ) ;
        printf( "cosh--->[%E][%s]\n", result, strerror( errno ) ) ;
        return( 0 ) ;
}
```

Workaround
None

No. C24	**A const long long int array element value is not referenced correctly**

IAR Reference: EWRL78-646

Details

The compiler can sometimes fail to calculate correct live ranges for local long long arrays causing them to share the same stack space with other local variables.

Example

```
#include <stdio.h>

int flg = 0 ;

void sub( void );

void sub( void )
{
    int i ;
    const signed long long int ary[1] = { 0LL } ;

    for ( i = 0 ; i < 1 ; i++ ) {
        if ( ary[i] != 0LL ) {
            flg++ ;
        }
    }
}

int main( void )
{
    sub() ;

    if( !flg ) {
        printf( "%-12s %04d:OK\n", __FILE__, __LINE__ ) ;
    } else {
        printf( "%-12s %04d:NG\n", __FILE__, __LINE__ ) ;
    }

    return( 0 ) ;
}
```

Workaround
None

No. C25	**If there are multiple if-statements that refer to function argument values, value judgment is incorrect.**

IAR Reference: EWRL78-644

Details

The compiler can sometimes remove 16-bit compares in if statements if the variable value instead of being re-read is restored by adding a constant before the compare.

Example

```
#include <stdio.h>

void sub( signed int );

void sub( signed int a )
{
        if ( a > 10 ) {
                printf( "%-12s %04d:NG [1]\n", __FILE__, __LINE__ ) ;
        } else if ( a > 0 && a <= 10 ) {
                printf( "%-12s %04d:NG [2]\n", __FILE__, __LINE__ ) ;
        } else if ( a >= -10 && a < 0 ) {
                printf( "%-12s %04d:NG [3]\n", __FILE__, __LINE__ ) ;
        } else {
                printf( "%-12s %04d:OK\n", __FILE__, __LINE__ ) ;
        }
}

int main( void )
{
        sub( 0 ) ;
        return( 0 ) ;
}
```

Workaround
None

No. C26	**A long long int array element value with auto storage duration is not referenced correctly.**

IAR Reference: EWRL78-645

Details

The compiler can sometimes fail to calculate correct live ranges for local long long arrays causing them to share the same stack space with other variables.

Example

```
#include <stdio.h>

int flg = 0 ;

#define N 2

void func( void );

void func( void )
{
              int i ;
              long long int a[N] = { 0, 1 } ;

              for ( i = 0; i < N; i++ ) {
                            if ( a[i] != i ) flg++ ;
              }
}

int main( void )
{
              func() ;

              if( flg == 0 ) {
                            printf( "%-12s %04d:OK\n", __FILE__, __LINE__ ) ;
              } else {
                            printf( "%-12s %04d:NG\n", __FILE__, __LINE__ ) ;
              }

              return( 0 ) ;
}
```

Workaround
None

No. C27	**A long long int array element value is not referenced using the const pointer correctly within the for-statement.**

IAR Reference: EWRL78-640/EWRL78-641

Details

Taking the address of a local long long array/struct and using it to initialize a local long long pointer can cause the two variables to share the same stack address.

Example

```
#include <stdio.h>

int flg = 0 ;

void sub( void );

void sub( void )
{
              int i ;
              signed long long int ary[1] = { 0LL } ;
              const signed long long int *ptr = &ary[0] ;
              for ( i = 0 ; i < 1 ; i++, ptr++ ) {
                            if ( *ptr != 0LL ) {
                                          flg++ ;
                            }
              }
}

int main( void )
{
              sub() ;
              if( !flg ) {
                            printf( "%-12s %04d:OK\n", __FILE__, __LINE__ ) ;
              } else {
                            printf( "%-12s %04d:NG\n", __FILE__, __LINE__ ) ;
              }
              return( 0 ) ;
}
```

Workaround
None

No. C28	**printf outputs nothing after long long int two-dimension arrays operation**

IAR Reference: EWRL78-638

Details

The compiler can sometimes fail to calculate correct live ranges for local long long arrays causing them to share the same stack space.

Example

```
#include <stdio.h>
int flg = 0 ;
void sub( void );

void sub( void )
{
        int i, j ;
        signed long long int ary1[1][6] = { { 1, 1, 1, 1, 1, 1, } } ;
        signed long long int ary2[1][6] = { { 1, 1, 1, 1, 1, 1, } } ;
        for( i = 0 ; i < 1 ; i++ )
                for( j = 0 ; j < 6 ; j++ ) {
                        ary1[i][j] -= ary2[i][j] ;
                        if ( ary1[i][j] != 0 ) {
                                flg++ ;
                        }
        }
}

int main( void )
{
        sub() ;
        if( !flg ) {
                printf( "%-12s %04d:OK\n", __FILE__, __LINE__ ) ;
        } else {
                printf( "%-12s %04d:NG\n", __FILE__, __LINE__ ) ;
        }
        return( 0 ) ;
}
```

Workaround
None

No. C29	**Internal Compiler Error: Double Defined Interrupt Vector**

IAR Reference: EWRL78-705/EWRL78-675

Details

Internal Compiler error will be thrown if a interrupt vector is double defined.

Example

```
#define my1_vect              (0x3E)
#define my2_vect              (0x3E)

#pragma vector=my1_vect, my2_vect
static __interrupt void my_interrupt (void) {

}
```

Following internal error will be thrown.

Tool Internal Error:
Internal Error: [Front end]: assertion failed: construct_message: not all fill-ins used (..\..\Translator\compiler_core\src\parser\edg\error.c, line 3989)

Internal Error: [Front end]: assertion failed: construct_message: not all fill-ins used (..\..\Translator\compiler_core\src\parser\edg\error.c, line 3989)

Tool Internal Error:
Internal Error: [OgModuleLabels::Def::Define]: Label already defined: ___interrupt_0x3E
Internal Error: [OgModuleLabels::Def::Define]: Label already defined: ___interrupt_0x3E
Error while running C/C++ Compiler

Workaround
None

No. C30	**Files based on the UTF-8 (BOM) format cannot be compiled**

IAR Reference: EWRL78-719

Details

The compiler emits "Error[Pe007]: unrecognized token" for UTF-8 (BOM) encoded source files.

Example
None

Workaround
None

No. C31	**Compiler can generate faulty code for 8-bit logical and arithmetic operations**

IAR Reference: EWRL78-699

Details

Depending on the register allocation, the compiler can generate faulty code for 8-bit logical and arithmetic operations where one of the operands is an indirection of a pointer.

Example

```
static unsigned char   cs_check(COMM_INFO *pi)
{
        RCV_DATA               *pp;
        unsigned char  rtn;
        unsigned char  cs;
        unsigned int   i, dtlen;

        pp = &pi->Rx;
        cs = 0;
        dtlen = pp->Len;
        for(i = 0; i < dtlen; i++)
        {
                cs ^= pp->Buf[i];
        }
        if(cs == 0)
        {       /* CheckByte OK */
                rtn = TRUE;
        }
        else
        {       /* CheckByte Ng */
                rtn = FALSE;
        }
        return rtn;
}
```

Workaround
Declare one of the operands volatile.

No. C32	**Data model will be ignored in case of using #pragma constseg**

IAR Reference: EWRL78-698

Details

#pragma constseg defaults to near memory regardless of selected data model.

Example

If project data model is set to far the constant TSV in the example shall be treated as far constant. However, in the example below the TSV constant will be treated as near constant whereas the data model is far.

```
const unsigned char TCV = 0x33;
#pragma constseg=__far "TSFar"
const unsigned char TSVfar = 0x55;
#pragma constseg="TS"
const unsigned char TSV = 0x77;
#pragma constseg = default

unsigned char TC;
unsigned char TSfar;
unsigned char TS;

void main(void)
{
  TC = TCV;
  TSfar = TSVfar;
  TS = TSV;
}
```

Workaround
Specify a memory attribute when you use #pragma constseg:

```
#pragma constseg= __far "MY_SEG"
```

No. C33	**Inline Assembler instruction generates an illegal syntax error**

IAR Reference: EWRL78-747

Details

The instruction MOV ES, S:label generates an illegal syntax error.

Example

```
__saddr unsigned char _AA;

int main( void )
{
       asm("MOV ES, S:_AA");
       return _AA;
}
```

Workaround
None

No. C34	**Error in floating point division**

IAR Reference: EWRL78-769

Details

Casting (explicit or implicit) a subnormal float to a double can cause the program to loop endlessly if the exact value of the float is 0x00100000. For other large subnormal values the result will be incorrect (values between 0x00080000-0x000FFFFF).

Example

None

Workaround

A workaround is to keep all operations in the float domain if possible.

Example:
f * 1.0
change it to
f * 1.0f

No. C35	**Memory dependency problem**

IAR Reference: EWRL78-770

Details

Due to a memory dependency problem, the compiler might generate slightly different code depending on which software license locking criteria is used. The code correctness is not affected.

Example

None

Workaround

A workaround is to keep all operations in the float domain if possible.

Example:
f * 1.0
change it to
f * 1.0f

No. C36	**Casting two far pointers to long integer and saving the difference will result in a wrong subtraction**

IAR Reference: EWRL78-774

Details

Calculating a memory area size by casting two far pointers to long integer and saving the difference will result in a subtraction of the index part of the pointers, i.e. the lower 16 bits, instead of the expected 32-bit subtraction.

Example

```
u_32 GetMemAreaSize(u_32 const __far *pMemAreaStart, u_32 const __far *pMemAreaEnd)
{
    u_32 MemAreaSize = 0;
    MemAreaSize = (u_32) ((u_32)pMemAreaEnd - (u_32)pMemAreaStart);
    return MemAreaSize;
}
```

GetMemAreaSize returns incorrect result 0xFFFFFFFE when called with 0x10002 & 0x30000. Correct value is 0x1FFFE.

Workaround

```
u_32 GetMemAreaSize(u_32 const __far *pMemAreaStart, u_32 const __far *pMemAreaEnd){
    u_32 MemAreaSize = 0;
    MemAreaSize = (u_32)((u_32 const __huge *)pMemAreaEnd - (u_32 const __huge *)pMemAreaStart);
    return MemAreaSize;
}
```

No. C37	**Internal error will be thrown in case optimization "Function inlining" is activated**

IAR Reference: EWRL78-788

Details

When using the function inlining optimization with far data model, accessing the first struct member via a pointer to said struct might cause an internal error.

Workaround

None

No. C38	**Incorrect code will be generated if Compiler optimization “Common subexpression elimination” is active**

IAR Reference: EWRL78-779

Details

Subtraction of char values can fail if there are two or more subtractions following each other where the minuends has the same value.

Workaround

Disable Compiler optimization “Common subexpression elimination”.

No. C39	**C++ Compiler can generate incorrect code for comparisons of floating-point numbers**

IAR Reference: EWRL78-773

Details

The C++ compiler can generate incorrect code for comparisons of floating-point numbers, on all optimization levels. If at least one of the numbers is NaN, the result of the comparison is sometimes reversed.

Workaround

None.

No. C40	**Byte order of the offset in the opcode for MOVW offset[BC/B/C],AX is swapped.**

IAR Reference: EWRL78-827

Details

The byte order of the offset in the opcode for MOVW offset[BC/B/C],AX is swapped. This only affects assembler code and C inline assembler, as the C/C++ compiler does not generate this instruction. Other instructions that use this address mode work correctly.

Example:

```
__asm("movw 0xdf22[BC],AX");
```

The above listed MOVW inline instruction will generate a wrong
OP code: 78 22DF → correct OP code shall be 78 DF22

Workaround

Manually swap byte order for the offset to BC/B/C for the instruction MOVW offset[BC/B/C], AX:

```
__asm("movw 0xdf22[BC],AX"); → __asm("movw 0x22df[BC],AX");
```

No. C41	**long long operations which are using the __Mul64 function are not reentrant**

IAR Reference: EWRL78-650, EWRL78-647, EWRL78-648, EWRL78-646, EWRL78-641, EWRL78-638

Details

Operations on long long variables might access the IAR __Mul64 library function which is using the RL78 MACH instruction. By executing the MACH instruction, the result will be stored into the MACR register. Since the __Mul64 function doesn't backup/restore the contents of MACR register that function is not reentrant and shall not be used inside of ISRs.

Workaround

Disable interrupts during the operation of long long variables were __Mul64 is used or avoid using long long operations inside of ISRs.

No. C42	**Faulty code for switches if the code for the switch and its associated cases span across a 64k border**

IAR Reference: EWRL78-831

Details

A program built with far code model, or using __far_func functions, can generate faulty code for switches if the code for the switch and its associated cases span across a 64k border. This only happens for one specific switch pattern. To check if the code might have this bug, check the compiler list files and the linker map file for labels containing the string VSWITCH.

Workaround

Change the placement of section .textf in the linker file from ROM_huge to ROM_far.

No. C43	**Constants located outside of the near area (flash mirror) cannot be used as parameter for printf function**

IAR Reference: EWRL78-800

Details

The printf function cannot handle a pointer which points to a memory area outside of the near area (flash mirror).

Example:

```
#pragma location=0x10000
__root __far const my_const[]=”hello”;

int main(void){
  printf(my_const);
}
```

Workaround
Copy the data from the constant into a RAM buffer and pass that buffer to the printf function.

No. C44	**Copying several bits (1-bit bitfields) in sequence to the same destination byte can generate faulty code on optimization level medium or higher.**

IAR Reference: EWRL78-883

Details

Copying several bits (1-bit bitfields) in sequence to the same destination byte can generate faulty code on optimization level medium or higher.

Example:

```
void copy_info(tester_nvm_t * p_xxx, const yyy_t * p_yyy) {
  p_xxx->bbb = p_yyy->bbb;       /* should write 0x55, this works */
  p_xxx->ccc = p_yyy->ccc;       /* should write 0xAA, this works */
  p_xxx->bit1 = p_yyy->bit1;     /* should write 0x01, this works */
  p_xxx->bit2 = p_yyy->bit2;     /* should write 0x00, !!!!BUG!!!!! */
}
```

Workaround
Use optimization level lower than medium for the affected code.

No. C45	**The tools crash when try to enter the debugger with E1 or E20.**

IAR Reference: EWRL78-903

Details

E1 and E20 emulators cannot update firmware due to missing files in the installation. This leads to a crash in the debug driver.

Workaround

Install a new instance of EWRL78 V4.21.1 and copy the following files to EWRL78 V4.21.3:

Files from EWRL78 V4.21.1
<IAR_INSTALL_FOLDER>\rl78\config\renesas\execs\BfwE20rl78_V152.s
<IAR_INSTALL_FOLDER>\rl78\config\renesas\execs\BfwE20mini2_V131.s

Copy to the following folder in EWRL78 V4.21.3
<IAR_INSTALL_FOLDER>\rl78\config\renesas\execs

No. C46	**Accessing the SFR area (0xFFF00) via near_pointer+index throws an error in case of Compiler optimization “Medium” and higher.**

IAR Reference: EWRL78-991

Details

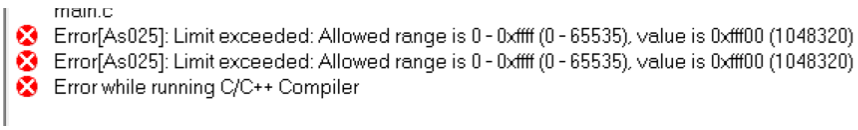
Initializing a near pointer with a constant value can generate illegal instructions on medium or high optimization levels. It mainly occurs with array and pointer indexing with a variable as index, i.e. ptr[index] or *(ptr+index).

Sample:

```
volatile char i = 3;
volatile char *firstIOAddress = (char *)0xFF00u;

/* error by reading access + index */
retVal = *(firstIOAddress+i);
retVal = firstIOAddress[i] ;
```

Following Compiler error will be thrown:



Workaround

Lower optimization level or make the pointer variable volatile or initialize the pointer in another file.

J) Description of Operating Precautions for Linker ILINKRL78 and ELF-Tools

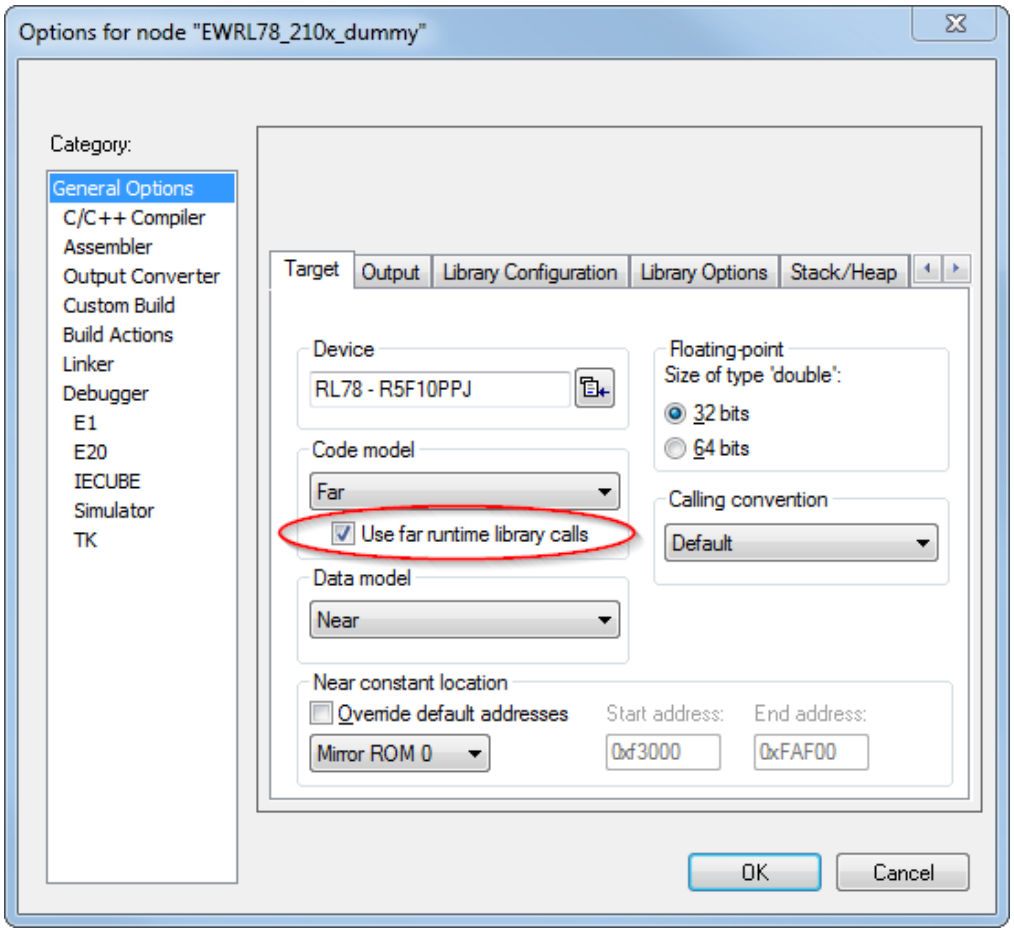
No. D1	**Runtime Model Conflict using far Runtime-Library-Calls**

IAR Reference EW25570

Details
In case of using far runtime-library-calls, the following linker error occurs as a matching runtime library variant is missing:

```
Error[Li009]: runtime model conflict: Module
__dbg_xxexit.o(dbgrl78fnf23d.a) specifies that '__far_rt_calls' must be
'false', but module <xxxxx.o> has the value 'true'
```

Using far runtime library calls is only necessary, if the runtime library itself shall be executed in RAM. The feature can be enabled in the GUI or by compiler command line option "--generate_far_runtime_library_calls" :



Workaround
Use a customer specific runtime library build with option ‘Using far runtime library calls ‘enabled.

No. D2	**Area in ROM marked as read-write-data in MAP File**

IAR Reference EW25758

Details
Although located in ROM memory-areas reserved for debugging are included as "read-write" memory in the linker map file module summary.
In the following sample the block OCD_ROM_AREA is listed as ‘rw data’ in the line ‘Linker created’:

```
*******************************************************************************
*** MODULE SUMMARY
***

    Module                      ro code  ro data  rw data  ro data  rw data
                                                           (abs)    (abs)
    ------                      -------  -------  -------  -------  -------
C:\...\QB-R5F10BMG-TB\startupsampleqb-r5f10bmg-tb\Debug\Obj: [1]
    globals.o                                          8
    interrupt.o                    24                          6
    low_level_initialization.o    137                         14       41
    main.o                        119                                   1
    ----------------------------------------------------------------------
    Total:                        280                  8       20       42

command line: [2]
    ----------------------------------------------------------------------
    Total:

dbgrl78nnf23nd.a: [3]
    __dbg_break.o                  3
    __dbg_xxexit.o                15
    ----------------------------------------------------------------------
    Total:                        18

dlrl78nnf23n.a: [4]
    cexit.o                        5
    cstartup.o                    56
    data_init.o                   66
    exit.o                         3
    huge_zero_init.o              107
    ----------------------------------------------------------------------
    Total:                        237

    Linker created                             24      640
----------------------------------------------------------------------
    Grand Total:                  535       24      648       20       42
```

Workaround
None. Will be fixed in next update

No. D3	**Routines for HW-Multiplier/Division Unit don’t support far runtime library calls**

IAR Reference EW25784

Details
The assembler routines for the Hardware Multiplier/Division Unit don’t support far runtime library calls and therefore cause a linker error:

```
Error[Lp002]: relocation failed: value out of range or illegal: ...
with
>place at address mem:0x20000       { ro section .text object LibReplacement.o };
```

Workaround
None. Will be fixed in next update

No. D4	**Internal error will be thrown if the section to be copied by “initialize manually” or “initialize by copy” feature is not placed**

IAR Reference EW25983

Details
By using the linker copy feature initialize manually or initialize by copy, a linker internal error will be generated if the section to be copied is not located in memory.

```
Error[Lc036]: no block or place matches the pattern "rw code section RAMSECTION in main.o
symbols: [_ram_func]"
Tool Internal Error:
Internal Error: [CoreUtil/General]: Access violation (0xc0000005) at 00441C1C (reading
from address 0x4) Internal Error:
[CoreUtil/General]: Access violation (0xc0000005) at 00441C1C (reading from address 0x4)
```

Workaround
Define all the sections which shall be copied via “initialize manually” or “initialize by copy”.

No. D5	**The symbol _NEAR_CONST_LOCATION_SIZE will be wrong calculated if Mirror ROM 1 is selected**

Now listed as IDE bug. See **No. A6**

No. D6	**Constant Data with Memory Attribute ‘near’ are treated as ‘readwrite’ Data in the Map File**

IAR Reference: EWRL78-527

Details

Constants defined with the __near memory attribute are included as "read-write" memory in the linker map file module summary.

Example:

```
__root const char my_const[4] = {0x11, 0x22, 0x33, 0x44};
```

For data model near the linker map file shows the following results:

```
  157 bytes of readonly  code memory
    4 bytes of readonly  data memory
  132 bytes of readwrite data memory
```

However, the above 4 byte array is a constant and therefore shall be added to the readonly memory instead of readwrite memory. Below you can see the correct results:

```
  161 bytes of readonly  code memory
    4 bytes of readonly  data memory
  128 bytes of readwrite data memory
```

Workaround
None. Will be fixed in next update.

No. D7	**The linker does not issue a warning if more than one interrupt function uses the same interrupt vector.**

IAR Reference: EWRL78-566

Details

Using of the same interrupt vector for two or more interrupt functions will not lead to a linker warning or error.

Example:

```
########
File1.c
########

static __interrupt void clock_monitor_interrupt(void);


#pragma vector = INTCLM_vect
static __interrupt void clock_monitor_interrupt(void)
{
    __no_operation();
}

########
File2.c
########

static __interrupt void other_interrupt(void);


#pragma vector = INTCLM_vect
static __interrupt void other_interrupt(void)
{
    __no_operation();
}
```

Workaround
None. Will be fixed in next update.

No. D8	**Switch/case statement over 64KB page**

IAR Reference: EWRL78-687

Details

A function using switches can fail if
- the switch is big enough to use a table and
- the switch cases are placed by the linker so that some of them end up in different 64k pages

Workaround
Use pragma location to place the function in its own section and add that section to the ROMFAR block in the linker file.

```
#pragma location=".my64kp"
void test()
{

}

"ROMFAR":place in ROM_far          { block INIT_ARRAY,
                                     block INIT_ARRAY_TLS,
                                     R_TEXTF_UNIT64KP,
                                     ro section .textf_unit64kp,
                                     ro section .constf,
                                     ro section .switchf,
                                     ro section .my64kp,
                                     ro };
```

No. D9	**__sfb() returns wrong address for section .text**

IAR Reference: EWRL78-753

Details

Using __sfb() to read the start of section .text in C/C++ can result in a wrong address being returned.

Example:
Start address of .text section is 0x300A.

Wrong result according to this bug:
__sfb(“.text”) → 0xF300A

Correct:
__sfb(“.text”) → 0x300A

Workaround
Mask with 0xFFFF to get a correct result for .text.

No. D10	**End address of SADDR region is wrong**

IAR Reference: -

Details

In all linker configuration file templates (*.icf) of the RL78/G10 series (R5F10Y14, R5F10Y16, R5F10Y17, R5F10Y44, R5F10Y46, R5F10Y47) the end address of the SDDR area is wrong. It must be 0xFFEDF instead of 0xFFEF7:

Workaround
Change the end address manually in the linker file.

Example for device R5F10Y14:

```
define region SADDR    = mem:[from 0xFFE60 to 0xFFEF7];
```

change to

```
define region SADDR    = mem:[from 0xFFE60 to 0xFFEDF];
```

No. D11	**Linker can terminate with an internal error if other (relevant) linking errors are present**

IAR Reference: EWRL78-784

Details

When linking object files generated with a compiler using the --mfc and --discard_unused_publics options, the linker can terminate with an internal error if other (relevant) linking errors are present at the same time

Workaround
None

No. D12	**Internal error will be thrown in during linking object files generated by the Renesas compiler**

IAR Reference: EWRL78-784, EWRL78-801

Details

When linking object files generated by the Renesas compiler (e.g. FSL, FDL and EEL), that contains symbols in meta sections (like the section symbol for a .rela relocation section), the linker might display non deterministic behavior for repeated builds. It can either successfully link the project, generate warnings (or errors), or terminate with an internal error.

Workaround
None

No. D13	**Error will be thrown by using the ielftool and filler bytes**

IAR Reference: EWRL78-811

Details

When generating filler bytes in applications where the segment ends on address X and the fill range ends on address X+1, ielftool fails to generate fill for address X+1.

If the byte on address X+1 is accessed by:
a) the checksum generator, this will trigger an error (uninitialized content).
b) the application itself, this will result in a read of uninitialized memory, the byte value is undefined.

Workaround
A workaround for this problem is to instead fill to address X (the last byte of the segment) or X+2 (or possibly more than 2, if X+1 has to be filled).

No. D14	**EWRL78 ielftool --fill option doesn't fill the first byte of unused area if the END address defined for --fill points**

IAR Reference: EWRL78-1013

Details

When generating filler bytes in applications where the segment ends on address X and the fill range ends on address X+1, ielftool fails to generate fill for address X+1. If the byte on address X+1 is accessed by:

a) the checksum generator, which will trigger an error (uninitialized content).
b) the application itself, which will result in a read of uninitialized memory, the byte value is undefined.

Workaround

A workaround for this problem is to instead fill to address X (the last byte of the segment) or X+2 (or possibly more than 2, if X+1 actually has to be filled).

Another workaround is to use ielftool v10.11.1 or newer.

No. D15	**Hardware multiplication replacement routines for 64-bit cannot be placed above the address 0xFFFF**

IAR Reference: EWRL78-1055

Details

The hardware multiplication replacement routines for 64-bit integers must be placed in the address range 0x0'0000-0x0'FFFF to avoid relocation errors while linking.

Workaround

Avoid using the hardware multiplier 64-bit replacement routines or make sure to place them in the first 64k code area.

No. D16	**Initializing data can fail if the block to copy is too big to fit inside the first available block in ROM**

IAR Reference: EWRL78-1056

Details

Using the initialize method copy to initialize data can fail if the block to copy is too big to fit inside the first available block in ROM, and the linker is forced to split it into two or more source parts while still maintaining a single destination part.

Example for “block in ROM” marked in yellow:

```
define region ROM_far  = mem:[from 0x000D8 to 0x03D8] | mem:[from 0x10000 to 0x103F0] ;
```

Workaround

Either make sure there is enough space in the first block used for the init data, or make sure to shrink the usable space in the first block by either filling it with stuff or shrinking the size of the block.

K) Description of Operating Precautions for Debugger C-SPY

No. E1	**E1 C-SPY Driver: Debug Session closed after Error 'Flash macro service ROM accessed or stepped in'**

IAR Reference EW25668

Details

The debug session is closed after error 'Flash macro service ROM accessed or stepped in' occurs. The error occurs, if a breakpoint is defined at a jump instruction while the Flash sequencer is active due to usage of a Renesas Flash Libraries. As the sequencer works asynchrony to program execution, the sequencer status is unknown to the user.
Due to the breakpoint C-SPY will use a step command to proceed even if the user command is "Go".

Workaround
Don't place a breakpoint on jump instructions while the Flash sequencer is active.

No. E2	**The C-SPY system macro __setLogBreak() does not work for emulators**

IAR Reference EW25840

Details

The C-SPY system macro __setLogBreak() does not work for emulators like E1, IECUBE etc. It can only be used via the Simulator.

Workaround
None.

No. E3	**IECUBE C-SPY Driver: Wrong average timer results**

IAR Reference EW25913

Details

In some cases it might happen that the timer average result of a conditional measurement is wrong.

Example:

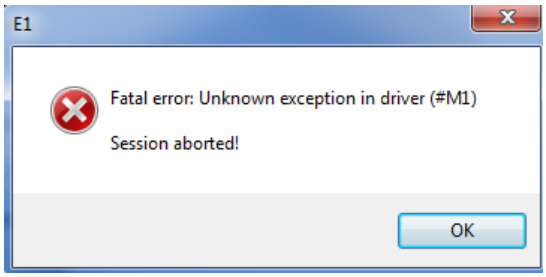
```
Timer 1: Pass count: 369. Average pass time: 5 msec. (total cycles: 239540413, average
cycles: 649161, min cycles: 12288621, max cycles: 12288686, rate: 8.33333 nsec/cycle).
```

Workaround
None. Please ignore the average result and use the min and max values for the investigation.

No. E4	**E1 C-SPY Driver: Step-in Step over doesn't work for switch case construct with more than 99 cases**

IAR Reference EW25950

Details
Temporary breakpoints are used for example in case of a single-step at C level. If you exceed the number of available temporary breakpoints that the OCD driver has allocated space for, an error is generated.



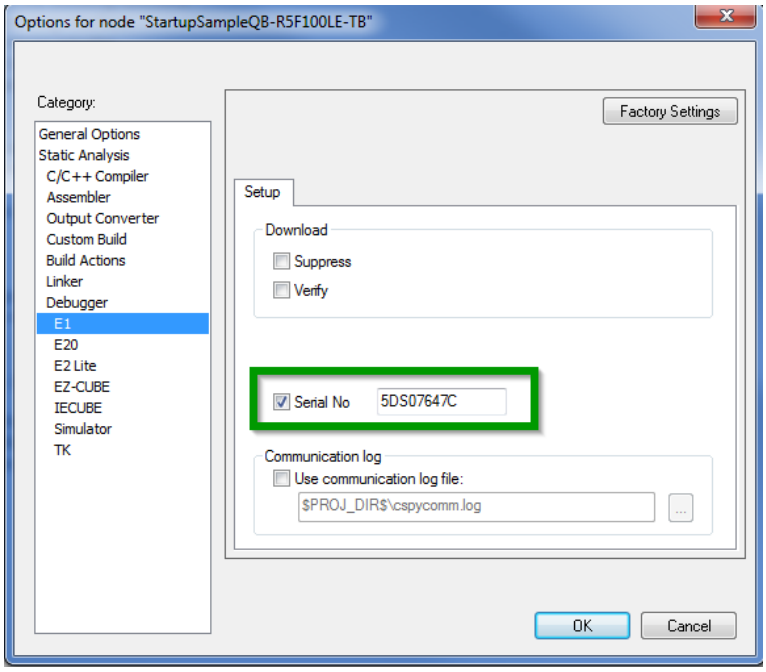
Workaround
Step at assembler level instead of at C level.

No. E5	**E1 C-SPY Driver: Specifying the serial number for the E1/E20 emulator sometime causes it not to be found.**

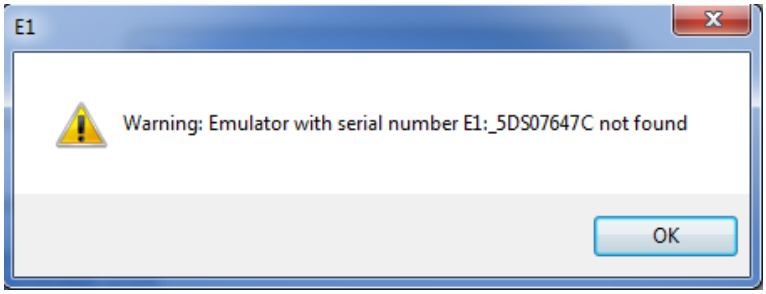
IAR Reference EWRL78-520

Details
The E1 serial number defined by the user will sometimes not be found whereas it is the correct one.

Example:



The debugger will throw the following warning and the debug session will be terminated.



Workaround
In case there is only one E1 required you can use the automated detection of the E1 instead of entering the serial number. Please uncheck the “Serial No” checkbox in that case.

No. E6	**Wrong sampled values might be shown in the Data Sample/Sampled Graphs window in case of sampling a variable with a size of 2 bytes**

IAR Reference EWRL78-533

Details
The sampling of two byte variables might lead to wrong values in the Data Sample or Sampled Graphs window. The probability to get a wrong value increases if the write frequency to the two byte variable is very high (e.g. toggle of the variable in a loop) and the sample period of the debugger very low (e.g. 10ms).

Workaround
None

No. E7	**E1 C-SPY Driver: Download of an additional image might destroy a part of the original application.**

IAR Reference EWRL78-513

Details

During the download procedure of an image the debugger performs the following steps:

1) Depending on the image size and location the flash will be erased by 2KB units
2) Image will be written to the flash memory

If the additional image to be downloaded is located directly below of the application it might happens that a part of the application will be destroyed.

Example:

Bootloader: 0x00000 - 0x0DBFF
Application: 0x0DC00 - 0x0FBFF

The above application is the main software which will be downloaded first and the bootloader will be downloaded afterwards as an image.

Because of the fact that the flash erase unit of the debugger is 2KB the image download will also erase the address 0xD800 to 0xDFFF. That means the first programmed application part (0x0DC00 to 0xDFFF) will be erased during the bootloader image download.

Workaround
Change the order of the download process:
1) Download the image with lower address range first (e.g. 0x00000 - 0x0DBFF)
2) Download the image with higher address range (e.g. 0x0DC00 - 0x0FBFF)

No. E8	**Data sampling time not constant**

IAR Reference EWRL78-671

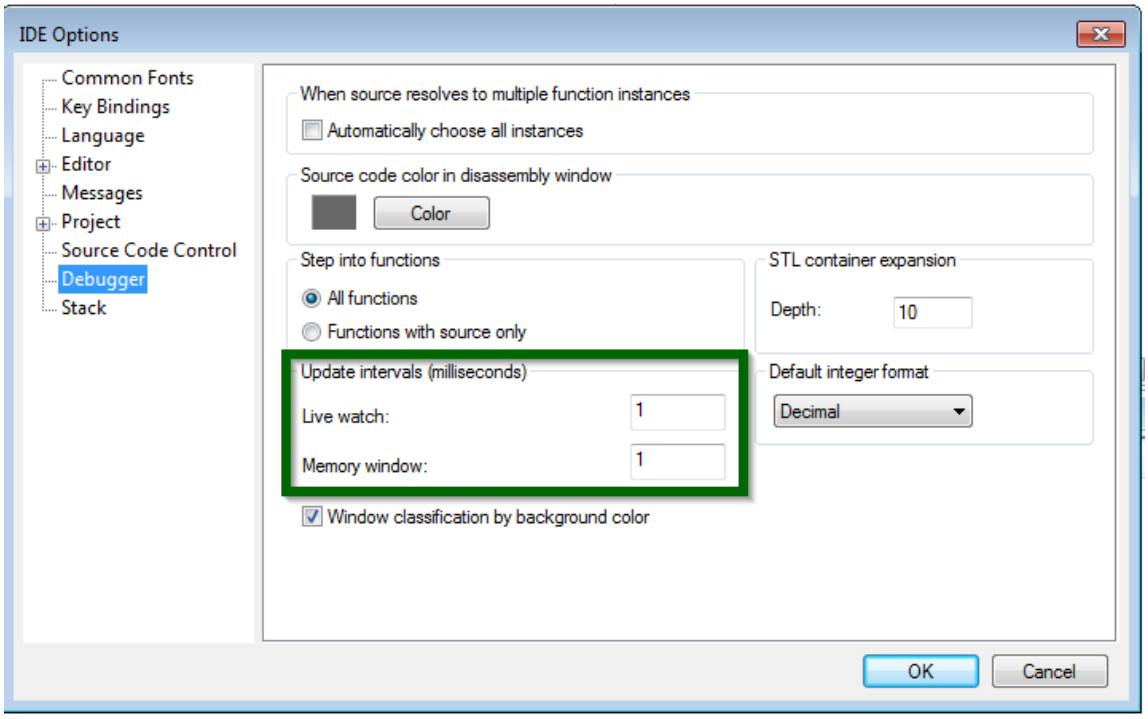
Details
The data sampling time can be configured for each variable separately with a minimum sampling of 10ms. However, because of the fact that the MS Windows is not a real-time OS the real sampling time depends on the current load of the PC and MS Windows. Therefore the data sampling time might not be constant.

Workaround
None

No. E9	**Min. update interval value for Live Watch and Memory Window is wrong**

IAR Reference EWRL78-671

Details
Currently the min. update interval value for the Live Watch and Memory Window can be set to 1ms. However, the min. supported update interval is 10ms.

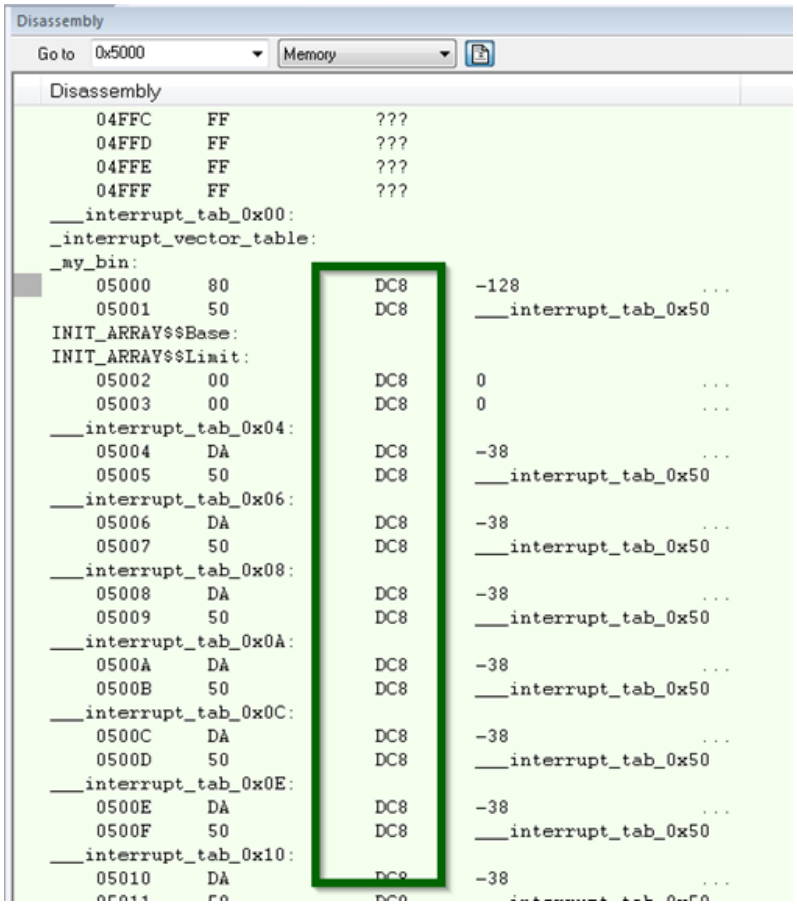


Workaround
Define an update interval which is not smaller than 10ms.

No. E10	**Binary image not showing symbol info in “Disassembly” window**

IAR Reference EWRL78-672

Details
During the build process of an application it is possible to link a pure binary file via the linker option –image_input. The content of the binary file could be code or constant data. However, in the debug session the “Disassembly” window always represents that content as a constant even if an additional debug file with code is loaded to the same area.



Workaround
None

No. E11	**Simulator interrupts can go wrong if code is above 64KB**

IAR Reference EWRL78-756

Details
Interrupts in the simulator can go wrong if the interrupt occurs when the current PC is on an address >= 64k.

Workaround
None

No. E12	**Simulator interrupts have wrong priority levels in case of shared vector**

IAR Reference EWRL78-764

Details
If there are several interrupts sharing a vector, the interrupt priority levels for the simulator will be wrong for all interrupts following.

Workaround
None

No. E13	**E1/E2 C-SPY Driver: OCD Trace automatically disabled in case Step-in/Step-over is used.**

IAR Reference EWRL78-771

Details
OCD Trace will be automatically disabled in case the user performs a Step-in or Step-over.

Workaround
None

No. E14	**EWRL78 hanged-up while power debugging**

IAR Reference EWRL78-786

Details
Using power debugging in C-SPY, the IDE sometimes hangs during single-stepping.

Workaround
None

No. E15	**In some situations, the debugger crashes when using an OCD emulator.**

IAR Reference EWRL78-778

Details
In some situations, the debugger crashes when using an OCD emulator.

Workaround
None

No. E16	**Debugging via hot-plugin doesn’t work**

IAR Reference EWRL78-862

Details
User is able to connect to the device via hot-plugin but features like Run/Break/Stop are not available.

Workaround
None

No. E17	**Simulator result of MACH instruction is wrong**
	IAR Reference EWRL78-941 *Details* Operation of the signed sum-of-products instruction (MACH) might be wrong in case of using the simulator for debugging. *Workaround* None

No. E18	**Loading of extra images fails if there is data in the EEPROM memory in the extra image.**
	IAR Reference EWRL78-1078 *Details* Loading of extra images fails if there is data in the EEPROM memory in the extra image. The problem lies in the memory cache and that the EEPROM memory is set to read only. After the main image is loaded, the cache is turned on and after that no writing of read only areas takes place. *Workaround* Change the 'Type' from 0x0000A to 0x00002 in the DDF file 'Memory map information' for the EEPROM area. Example of DDF file content: Map2 = 0xF1000, 0xF2FFF, 0x0000A, 0x1 Change to: Map2 = 0xF1000, 0xF2FFF, 0x00002, 0x1

No. E19	**Several registers are missing in the Register view. E.g. the register “ADCSR”**
	IAR Reference EWRL78-1112 *Details* If several sfr’s share the same address, only one of them is used in the group descriptions used by the register window. *Workaround* None

No. E20	**FAA breakpoints cannot be removed after setting**
	IAR Reference EWRL78-1103 *Details* The maximum number of breakpoints that can be set in FAA code is four. If more than four breakpoints are set, they cannot be removed again. *Workaround* None

No. E21	**The execution speed decreases by approximately half when activate FAA debugging**
	IAR Reference EWRL78-1099 *Details* The execution speed decreases by approximately half when you activate FAA debugging, because the SFR memory is read during execution. *Workaround* None

No. E22	**The C-SPY system macro __setCodeBreak does not work with the emulator.**
	IAR Reference EWRL78-1090 *Details* The C-SPY system macro __setCodeBreak does not work with the emulator. *Workaround* None

L) Description of Operating Precautions for Runtime Library

No. F1	**Runtime library are placed in the wrong section if generate far runtime library calls is chosen.**

IAR Reference EWRL78-497

Details

When building a project with the “far runtime library calls” feature and usage of integer arithmetic libraries the linker might throw a range error as shown within the following sample:

```
main.c
======
long long l = 2;
int main( void )
{
  if ( l < 4 )
    return 0;
  else
    return 1;
}

Linker output
=============

   IAR ELF Linker V2.21.1.1833 for RL78
   Copyright 2011-2016 IAR Systems AB.
Error[Lp002]: relocation failed: value out of range or illegal: 0x10044
     Kind    : R_RL78_DIR16U[0x4]
     Location: 0x10077
               "__CmpGes64" + 0x1
               Module:  longlong.o(dlrl78fff23nf.a)
               Section: 6 (.ftext)
               Offset:  0x608
     Target  : 0x10044
               "__Cmp64s"
               Module:  longlong.o(dlrl78fff23nf.a)
               Section: 6 (.ftext)
               Offset:  0x5be
```

Workaround
None

No. F2	**If the option “Use far runtime library calls” is used, assembler support routines from the run time library are placed in an incorrect section called “.ftext”**

IAR Reference EWRL78-496

Details

If the option “Use far runtime library calls” is used, assembler support routines from the run time library are placed in an incorrect section called “.ftext”

Workaround
None

No. F3	**The section name .textf_unit64kp is misspelled in the linker files.**

IAR Reference EWRL78-511

Details

The section name .textf_unit64kp is misspelled in the standard icf linker file. It is called ".text_unit64kp" instead of ".textf_unit64kp". Therefore the following warning might be thrown by the linker:

```
ro section .text_unit64kp,
                          ^
"C:\Program Files (x86)\IAR Systems\Embedded Workbench 7.4\rl78\config\lnkr5f104fj.icf",115
Warning[Lc059]: the section name in this pattern caused it to not match any sections.
```

Workaround
Please rename the section ".text_unit64kp" to ".textf_unit64kp".

No. F5	**Overlapping of two registers**

IAR Reference none

Details

Depending on the used peripherals of the device there are sometimes two register which are located at the same address. If both register are referenced by the application the linker will throw an overlapping error.

Example:

Referenced register are LMD0 and LMD1. The linker will throw the following error:

Error[e24]: Segment NEAR_A (seg part no 7, symbol "_A_LMD1" in module "sci_rl78", address [f06c8-f06c8]) overlaps segment NEAR_A (seg part no 23, symbol "_A_LMD0" in module "LIN_Drv_RL78", address [f06c8-f06c8]) .

The root cause for this problem is that both register are defined in the io header file (e.g. ior5f10pgg_ext.h) for the same address but in different unions:

```
…..
__near __no_bit_access __no_init volatile union { unsigned char LMD0; __BITS8 LMD0_bit; } @ 0xF06C8;
__near __no_bit_access __no_init volatile union { unsigned char LMD1; __BITS8 LMD1_bit; } @ 0xF06C8;
…..
```

Workaround
Add both register into one unit like shown below:

```
__near __no_bit_access __no_init volatile union {
                                    unsigned char LMD0; __BITS8 LMD0_bit;
                                    unsigned char LMD1; __BITS8 LMD1_bit;
                                    } @ 0xF06C8;
```

No. F6	**Linker might fail with the internal error Distributor::TraverseRanges**
	IAR Reference: EWRL78-665 *Details* In rare cases, the linker might fail with the following internal error: Internal Error: [CoreUtil/General]: Distributor::TraverseRanges - range overshoot: 0x155a3 > 0x10000 *Workaround* None

No. F7	**Libraries generated with IAR V2.xx version cannot be linked on newer IAR versions if the library includes a vector table.**
	IAR Reference: EWRL78-746 *Details* In version V3.10 the handling of the interrupt vector table was changed so it could handle a movable interrupt vector table. That broke backwards compatibility with version V2.xx since the compiler now utilizes a vector table instead of placing the vectors at fixed addresses. The result is that the vector table area is filled with the vector table and trying to link an old file with a vector entry at a fixed address will generate a placement error as the vector table area is already filled. *Workaround* Re-build the library on the version V3.xx or newer.

M) Valid Specification

Item	Date published	Document No.	Document Title
1	November 2022	UIDERL78_I-7	IAR Embedded Workbench IDE Project Management and Building Guide
2	February 2023	DRL78-I-6a	IAR Embedded Workbench C/C++ Development Guide Compiling and Linking for RL78
3	December 2018	ARL78_I-2	IAR Embedded Workbench Assembler User Guide for RL78
4	November 2022	UCSRL78_I-5	IAR Embedded Workbench C-SPY Debugging Guide for RL78
5	April 2016	MUBROFELFRL78_I-2	IAR Embedded Workbench Migrating from UBROF to ELF/DWARF
6	January 2011	EWMISRAC1998-4	IAR Embedded Workbench MISRA C 1998 Reference Guide
7	January 2011	EWMISRAC:2004-3	IAR Embedded Workbench MISRA C 2004 Reference Guide

N) Revision

Edition	Date published	Document No.	Comment
1	18-06-2015	R20UT3407ED0100	Initial release.
2	29-06-2015	R20UT3407ED0101	Item C2 added.
3	15-07-2015	R20UT3407ED0102	Compiler and Assembler update to V2.10.2, items C3 and D1 added.
4	21-08-2015	R20UT3407ED0103	Update V2.10.3 Item C4 added.
5	15-09-2015	R20UT3407ED0104	Item E1 added.
6	20-10-2015	R20UT3407ED0105	Items C5 and D2 added.
7	22-10-2015	R20UT3407ED0106	Item C6 added
8	16-11-2015	R20UT3407ED0107	Update EWRL78 V2.10.4 Item D3 added
9	21-12-2015	R20UT3407ED0108	Update EWRL78 V2.20.1 Item C6 sample updated Item E2 added.
10	01-02-2016	R20UT3407ED0109	Item E3 added.
11	16-03-2016	R20UT3407ED0110	Item C7 added
12	27.04.2016	R20UT3407ED0111	Item A1 added Item C8 added Item C9 added Item D4 added
13	08.07.2016	R20UT3407ED0112	Update EWRL78 V2.21.1/ V2.21.2 Item E4 added Item F1 added Item F2 added
14	14.07.2016	R20UT3407ED0113	Item C10 added
15	22.08.2016	R20UT3407ED0114	Item A2 added, update of valid specification
16	11.10.2016	R20UT3407ED0115	Item E5 added Item F3 added Item F5 added
17	23.11.2016	R20UT3407ED0116	Item E6 added
18	21.12.2016	R20UT3407ED0117	Item D5 added Item A3 added
19	06.02.2017	R20UT3407ED0118	Items A4, C11, and D6 added
20	13.03.2017	R20UT3407ED0119	Item A5 added Item D7 added
21	21.04.2017	R20UT3407ED0120	Update EWRL78 V2.21.5 Item C12 added Item E7 added

Edition	Date published	Document No.	Comment
22	09.06.2017	R20UT3407ED0121	Item C13 added Item C14 added Item C15 added Item C16 added Item C17 added Item C18 added Item C19 added Item C20 added Item C21 added Item C22 added Item C23 added
23	08.09.2017	R20UT3407ED0122	Item C24 added Item C25 added Item C26 added Item C27 added Item C28 added
24	02.10.2017	R20UT3407ED0123	Update EWRL78 V3.10.1 Item E8 added Item E9 added Item F6 added Specification Update (Chapter M))
25	08.11.2017	R20UT3407ED0124	Item A7 added Item A8 added
26	21.12.2017	R20UT3407ED0125	Item E10 added Item D8 added
27	21.03.2018	R20UT3407ED0126	Item C29 added
28	27.07.2018	R20UT3407ED0127	Item C30 added
29	13.11.2018	R20UT3407ED0128	Item C31 added Item C32 added Update C29: Bug with the ID EWRL78-675 is same as EWRL78-705
30	30.01.2019	R20UT3407ED0129	Update EWRL78 V4.10.1
31	06.05.2019	R20UT3407ED0130	Item C33 added Item F7 added
32	13.08.2019	R20UT3407ED0131	Item A9 added Item D9 added Item D10 added Item E11 added
33	09.12.2019	R20UT3407ED0132	Update EWRL78 V4.20.1 Item C34 added Item C35 added Item E12 added Item E13 added Specification Update (Chapter M)

Edition	Date published	Document No.	Comment
34	20.02.2020	R20UT3407ED0133	Update EWRL78 V4.20.2 Item C36 added Item C37 added Item C38 added Item C39 added Item E14 added Item E15 added Item D11 added
35	04.06.2020	R20UT3407ED0134	Item A10 added Item A11 added Item D8 updated: It showed that this error was partially fixed in v4.10. One kind of switch that requires tables was not fixed. Item D12 added
36	19.03.2021	R20UT3407ED0135	Item C40 added Item C41 added Item D13 added
37	14.10.2021	R20UT3407ED0136	Update EWRL78 V4.21.1 Item C26 updated (bug fixed in version V4.20.x) Item C42 added Item C43 added Item E16 added
38	28.04.2022	R20UT3407ED0137	Update EWRL78 V4.21.2 / V4.21.3 Item C44 added Item C45 added
39	16.11.2022	R20UT3407ED0138	Update EWRL78 V4.21.4 Item C46 added Item E17 added
40	28.03.2023	R20UT3407ED0139	Update EWRL78 V5.10.1 Item D14 added Item D15 added Item D16 added
41	16.10.2023	R20UT3407ED0140	Update EWRL78 V5.10.3 Item A4 corrected for V5.10.1 Item E18 added
42	19.02.2024	R20UT3407ED0141	Update EWRL78 V5.10.4 Item E19 added Item E20 added Item E21 added Item E22 added Item A12 added

Before using this material, please visit our website to confirm using the most current document available:
Current version of this document

In case of any technical question related to the Embedded Workbench for RL78, please feel free to contact the Renesas Software-Tool-Support Team.

Renesas Electronics Europe

R20UT3407ED0141
February 2024