Customer Notification

EWRL78 V1.xx

Embedded Workbench® for RL78 V1.xx

Operating Precautions

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

Renesas Electronics
www.renesas.com

Document No. R20UT0521ED0168
Date Published: March 2021

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	EWRL78					
	Version	6.1.5	6.3.18	6.5.3	6.5.11	7.0.2	7.0.5
A1	An empty Workspace can not be saved	×	✓	✓	✓	✓	✓
A2	Empty Go to Function Window	×	×	×	✓	✓	✓
A3	Number of total Errors and Warnings doubled by Mistake	×	✓	✓	✓	✓	✓
A4	IarBuild.exe uses incorrect Hardware Multiplier / Divider Support File	×	×	✓	✓	✓	✓
A5	Bad initial Stack Size Value in C Project Template	✓	✓	×	✓	✓	✓
A6	Wrong NEAR_CONST Definition in XCL File Template	✓	✓	×	✓	✓	✓
A7	Wrong 16bit signed Operation using Hardware Multiplier/Divider	×	×	×	×	✓	✓
A8	Actual Linker-MAP-File not automatically updated in Editor	×	×	×	×	✓	✓
A9	Stack Size of 64Byte cannot be permanently defined in IDE	-	-	×	×	×	✓
A10	Hardware Multiplier/Divider Unit configuration changed unexpected in case of using two projects within one workspace	-	✓	×	×	×	×
A11	Double Entries in Create New Project Dialogue	✓	✓	✓	✓	×	×
A12	RL78 Mirror Area Configuration changed to default Values	-	-	×	×	×	×
A13	Incorrect Memory Area Definitions in XCL-File Templates	×	×	×	×	×	×
A14	Loading of *.ipcf file generates warnings	×	×	×	×	×	×

×: Applicable ✓: Not applicable - : Not checked

B) Table of Operating Precautions for the Assembler ARL78

No.	Outline	ARL78					
	Version	1.20.1	1.30.1	1.30.3	1.30.4	1.40.1	1.40.3
B1	RSEG Directives can not be used in Macro Definitions	×	×	×	×	×	×
B3	Assembler File must contain at least one Directive	×	×	×	×	×	×
B5	Assembler Error caused by Call Frame Information	×	✓	✓	✓	✓	✓
B6	Wrong Code Generated for relative Addressing	×	×	×	✓	✓	✓
B7	Incorrect Source Line Information	-	×	×	×	×	✓

×: Applicable ✓: Not applicable - : Not checked

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

No.	Outline	ICCRL78								
	Version	1.20.4	1.30.2	1.30.3	1.30.5	1.40.1	1.40.3	1.40.5	1.40.6	
C10	Inline Assembler: Double defined Label causes internal Compiler Error	×	✓	✓	✓	✓	✓	✓	✓	
C13	#pragma location Directive does not support Unions and Structures	×	✓	✓	✓	✓	✓	✓	✓	
C18	Keyword __no_bit_access does not work on auto Variables	×	×	×	✓	✓	✓	✓	✓	
C20	Signed Division HWMDU Functions for RL78 Core2 are not Interrupt safe	×	✓	✓	✓	✓	✓	✓	✓	
C21	Illegal 8bit Access to I/O Register allowing only 16bit Access	×	✓	✓	✓	✓	✓	✓	✓	
C22	Wrong Inline Assembler Translation	×	×	✓	✓	✓	✓	✓	✓	
C23	Bit Access generated although Keyword '__no_bit_access' was used	×	×	✓	✓	✓	✓	✓	✓	
C24	Wrong indirect post Increment of a Result of a post Increment	×	×	✓	✓	✓	✓	✓	✓	
C25	Wrong Optimization of indirect Variable increment in nested do Loops	×	×	✓	✓	✓	✓	✓	✓	
C26	Internal Compiler Error while copying a packed Structure	×	×	×	✓	✓	✓	✓	✓	
C27	Wrong Code generated for Pointer Comparison with Zero	×	×	×	✓	✓	✓	✓	✓	
C28	Wrong Code generated for local Variable Access	×	×	×	✓	✓	✓	✓	✓	
C29	Wrong Prototype Description in Compiler Manual of Function __segment_size	×	×	×	×	✓	✓	✓	✓	
C30	Wrong Code generated for local far Pointer loaded via far Pointer	✓	×	×	✓	✓	✓	✓	✓	
C31	Unclear Description of Parameter Passing for Structure Types in Compiler Manual	×	×	×	×	✓	✓	✓	✓	
C32	Internal Compiler Error by erroneous Bitfield Definition	-	×	×	×	✓	✓	✓	✓	
C33	Internal Compiler Error after several ordinary Errors	-	×	×	×	✓	✓	✓	✓	
C34	Internal Compiler Error if a Function of more than 255 Parameters is used	-	×	×	×	×	✓	✓	✓	
C35	Internal Compiler Error after illegal enum-Value Error	-	×	×	×	✓	✓	✓	✓	
C36	Internal Compiler Error after Error [Pe078]	-	×	×	×	✓	✓	✓	✓	
C37	Internal Compiler Error after Error [Pe066] (1)	-	×	×	×	✓	✓	✓	✓	
C38	Internal Compiler Error after Error [Pe066] (2)	-	×	×	×	✓	✓	✓	✓	
C39	Internal Compiler Error: Stack Overflow	-	×	×	×	×	×	×	×	

No.	Outline	ICCRL78								
	Version	1.20.4	1.30.2	1.30.3	1.30.5	1.40.1	1.40.3	1.40.5	1.40.6	
C40	Keyword '__no_bit_access' fails at explicit cast to 16bit data type	×	×	×	×	✓	✓	✓	✓	
C41	Internal Compiler Error at Function defined by Macros	×	×	×	×	×	×	×	×	
C42	Wrong Code generated for indirect Access to Structure Member	×	×	×	×	✓	✓	✓	✓	
C43	Internal Compiler Error in case of using nested Boolean Expressions	✓	×	×	×	✓	✓	✓	✓	
C44	Internal Compiler Error after Error Pe066	×	×	×	×	✓	✓	✓	✓	
C45	Near-Call in Floating Point Library causes a Linker Error	×	×	×	×	✓	✓	✓	✓	
C46	Wrong Code generated while Copying a 1-Bit Bitfield	-	×	×	×	×	✓	✓	✓	
C47	Keyword 'const' disables #pragma default_variable_attribute Directive	✓	×	×	×	×	✓	✓	✓	
C48	MISRA C 2004 Rule 10.6 not triggered	×	×	×	×	×	×	×	×	
C49	Wrong Result of signed Integer Division	✓	×	×	×	✓	✓	✓	✓	
C50	Manual Error in Description of Option '--disable_div_mod_instructions'	×	×	×	×	×	×	×	×	
C51	Huge constant Data placed in Segment 'NEAR_CONST'	✓	✓	✓	✓	×	✓	✓	✓	
C52	Stack Content can be corrupted by ISR	×	✓	✓	✓	✓	✓	✓	✓	
C53	MISRA C Rule 10.1 triggered by Mistake	×	×	×	×	×	×	×	×	
C54	Internal Error at Comparison of near Pointer	✓	✓	✓	✓	×	×	✓	✓	
C55	Internal Error at Bitfield Assignment	✓	×	×	×	×	×	×	×	
C56	Internal Error at Switch Statement	×	×	×	×	×	×	×	×	
C57	Wrong code generated for inline String Literal Copying	×	×	×	×	×	×	×	✓	
C58	Wrong Code generated for Multiple Bitfield Assignments of Constant	×	×	×	×	×	×	×	×	
C59	Wrong Code generated for multiple Bitfield Assignment in one Statement	-	×	×	×	×	×	×	×	
C60	Wrong Code generated for Calculation depending on Overflow of smaller Datatype	-	×	×	×	×	×	✓	✓	
C61	Wrong Code generated for Return-Value including Assignment	×	×	×	×	×	✓	✓	✓	
C62	Inserted NOP after DIVWU/DIVHU Instruction moved	✓	✓	✓	✓	✓	×	×	✓	
C63	User defined Stack Size overwritten by Default Size	✓	✓	✓	✓	✓	×	×	✓	
C64	Wrong Code generated for direct Access to of Hardware Multiplier / Divider Register	×	×	×	×	×	×	×	×	
C65	Internal Compiler Error using different I/O Register Definitions in different Modules	×	×	×	×	×	×	×	×	
C66	Compiler Error Pe147 triggered by Mistake	×	×	×	×	×	×	×	×	

No.	Outline	ICCRL78								
	Version	1.20.4	1.30.2	1.30.3	1.30.5	1.40.1	1.40.3	1.40.5	1.40.6	
C67	Internal Compiler Error using Datatype 'long long' as Switch-Expression	✓	✓	✓	✓	×	×	×	×	
C68	Internal Compiler Error using explicit double Casting	×	×	×	×	×	×	×	×	
C69	Inconsistency of extended Keyword __monitor	-	-	-	-	×	×	×	×	
C70	Floating point comparison fails if the difference between the operands is one bit only.	-	-	-	-	×	×	×	×	
C71	An internal error will be generated in case of sequential pointer casting	✓	✓	✓	✓	×	×	×	×	
C72	Wrong Optimization of static local Variable	-	×	×	×	×	×	×	×	
C73	Inserted NOP after DIVWU/DIVHU Instruction moved (cross call optimization)	×	×	×	×	×	×	×	×	
C74	The C library function isblank(c) will in some cases erroneously return true	×	×	×	×	×	×	×	×	
C75	Switch state in recursive function generates an internal error	×	×	×	×	×	×	×	×	
C76	Error in case a simple character literal is followed by a wide character literal	×	×	×	×	×	×	×	×	
C77	Sign-extending a signed int/short register variable to a long can destroy a variable	✓	✓	✓	×	×	×	×	×	
C78	Range error on nextXXX() functions	×	×	×	×	×	×	×	×	
C79	No output to stdout when putchar(-1) is used	×	×	×	×	×	×	×	×	
C80	Different return value between iswctype and iswblank	×	×	×	×	×	×	×	×	
C81	%Z format output for strftime is wrong	×	×	×	×	×	×	×	×	
C82	Square root function in the floating point library returns +0.0 for sqrt(-0.0)	×	×	×	×	×	×	×	×	
C83	errno() might cause a range error	×	×	×	×	×	×	×	×	
C84	Wrong result in case of Complex_I multiplication with -0.0	×	×	×	×	×	×	×	×	
C85	Function cosh() does not set errno()	×	×	×	×	×	×	×	×	
C86	A const long long int array element value is not referenced correctly	✓	✓	✓	✓	×	×	×	×	
C87	If there are multiple if-statements that refer to function argument values, value judgment is incorrect.	×	×	×	×	×	×	×	×	
C88	A long long int array element value with auto storage duration is not referenced correctly.	✓	✓	✓	✓	×	×	×	×	
C89	A long long int array element value is not referenced using the const pointer correctly within the for-statement.	✓	✓	✓	✓	×	×	×	×	
C90	printf outputs nothing after long long int two-dimension arrays operation	✓	✓	✓	✓	×	×	×	×	
C91	long long int switch-statement causes internal error	✓	✓	✓	✓	×	×	×	×	
C92	Operation with a long long int type member of structure causes internal error	✓	✓	✓	✓	×	×	×	×	

No.	Outline	ICCRL78								
	Version	1.20.4	1.30.2	1.30.3	1.30.5	1.40.1	1.40.3	1.40.5	1.40.6	
C93	An extraneous memory read can occur when you read a volatile bitfield	✓	×	×	×	×	×	×	×	
C94	Optimizer considers all long long constants as equal	-	-	-	-	✓	×	×	×	
C95	long long operations which are using the __Mul64 function are not reentrant	×	×	×	×	×	×	×	×	

×: Applicable ✓: Not applicable - : Not checked

D) Table of Operating Precautions for the Linker XLINK

No.	Outline	XLINK					
	Version	5.7.1.40	5.8.0.42	6.0.3.49	6.1.2.53	6.1.3.56	6.3.3.74
D5	ELF Output File Format: Error e113 'Illegal ELF Register'	×	×	✓	✓	✓	✓
D6	Erroneously Error e16 'Segment too long' is generated (I)	×	✓	✓	✓	✓	✓
D7	Erroneously Error e16 'Segment too long' is generated (II)	×	×	×	×	×	×
D8	Range Error using far Runtime-Library Calls	×	×	×	×	×	×
D9	Negative Value for N/A (alignment)	×	×	×	×	×	×
D10	Unused Addresses in Common Segments not filled correctly	×	×	×	×	×	×
D11	Comand Line Segment Alignment ignored	×	×	×	×	×	✓
D12	Symbol division results in a "division by zero" error	✓	✓	×	×	×	×
D13	End address of checksum is wrong when using the -M option	×	×	×	×	×	×
D14	Segment alignment fails by using the -Z option	×	×	×	×	×	×
D15	End address of SADDR region is wrong	×	×	×	×	×	×

×: Applicable ✓: Not applicable - : Not checked

E) Table of Operating Precautions for Debugger C-SPY

No.	Outline	C-SPY						
	Version	1.20.3	1.20.4	1.30.2	1.30.4	1.40.1	1.40.3	1.40.6
E5	All C-SPY Drivers: Structure not displayed Watch Windows	×	×	×	×	×	×	×
E9	E1 C-SPY Driver: No automatic Mapping for Variables added to Live Watch Window	×	×	✓	✓	✓	✓	✓
E10	All C-SPY Drivers: Symbols not listed in Symbolic Memory Window	×	×	×	×	×	×	×
E12	C-SPY IECUBE Driver: Pseudo Emulation of Temperature Sensor does not work	×	✓	✓	✓	✓	✓	✓
E13	C-SPY Simulator Driver: Display Problem in Timeline Window	×	×	✓	✓	✓	✓	✓
E14	Wrong Manual I/O Register Modification	✓	✓	×	✓	✓	✓	✓
E16	All C-SPY Drivers: Registers MDAL and MDAH not displayed in Register Window	×	×	×	✓	✓	✓	✓
E17	C-SPY E1 Driver: Unknown Break Error	✓	✓	×	✓	✓	✓	✓
E18	C-SPY E1 Driver: Application doesn't start after Debug Session	✓	✓	×	✓	✓	✓	✓
E19	C-SPY E1 Driver: Crash at Reaching a Software Breakpoint	✓	✓	×	✓	✓	✓	✓
E20	All C-SPY Drivers: Debug Session did not Start	×	×	×	×	✓	✓	✓
E21	IECUBE and E1 C-SPY Drivers: Data Flash Memory Window cannot be opened	✓	✓	✓	×	✓	✓	✓
E22	IECUBE and E1 C-SPY Drivers: Data Flash Memory Content cannot be changed in Memory Window	✓	✓	✓	×	✓	✓	✓
E23	E1 C-SPY Driver: IDE hangs due to Missing Frames in Trace Buffer	✓	✓	✓	×	✓	✓	✓
E24	IECUBE C-SPY Driver: Wrong Time Stamp Information	×	×	×	×	✓	✓	✓
E25	E1 C-SPY Driver: Data Sample Graph is not updated	✓	✓	×	×	✓	✓	✓
E26	E1 C-SPY Driver: Debug Session closed after Error 'Flash macro service ROM accessed or stepped in'	×	×	×	×	×	✓	✓

No.	Outline	C-SPY						
	Version	1.20.3	1.20.4	1.30.2	1.30.4	1.40.1	1.40.3	1.40.6
E27	E1 C-SPY Driver: RL78 device feature “RAM guard” doesn’t work in case of single step execution on assembler instruction level	×	×	×	×	×	×	✓
E28	E1 C-SPY Driver: Wrong Address area displayed in Error Message	✓	✓	✓	✓	✓	✓	✓
E29	IECUBE C-SPY Driver: Debug Session closed after Fail-Safe-Break	✓	✓	✓	✓	✓	✓	×
E30	E1 C-SPY Driver: Debug Session closed after Error 'Flash macro service ROM accessed or stepped in' (II)	×	×	×	×	×	×	×
E31	IECUBE C-SPY Driver: Wrong average timer results	×	×	×	×	×	×	×
E32	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	×	×	×	×	×	×	×
E33	E1 C-SPY Driver: Download of an additional image might destroy a part of the original application.	×	×	×	×	×	×	×

×: Applicable ✓: Not applicable - : Not checked

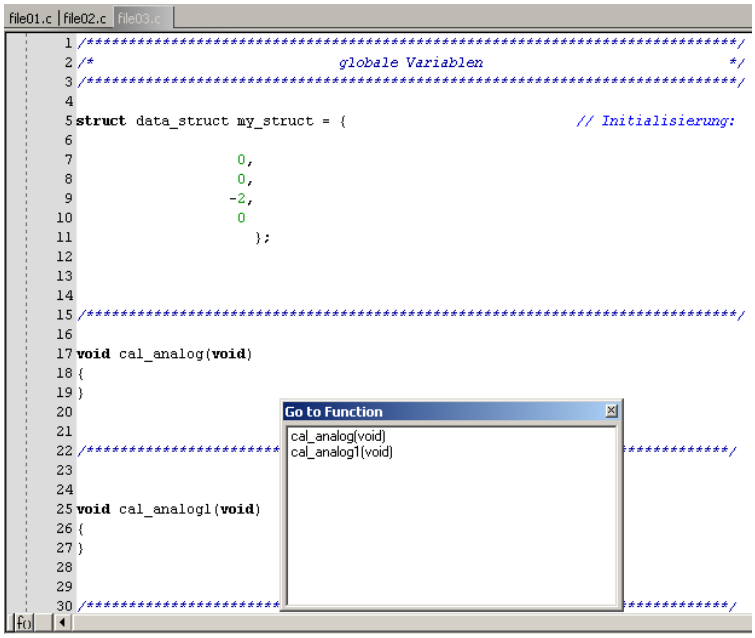
F) Description of Operating Precautions for the IDE EWRL78

No. *A*1	**An empty workspace can not be saved**
	Details Although it is described in the user's manual an empty workspace can not be saved. *Workaround* Add at least one project to the workspace before saving. The project may be an empty project.

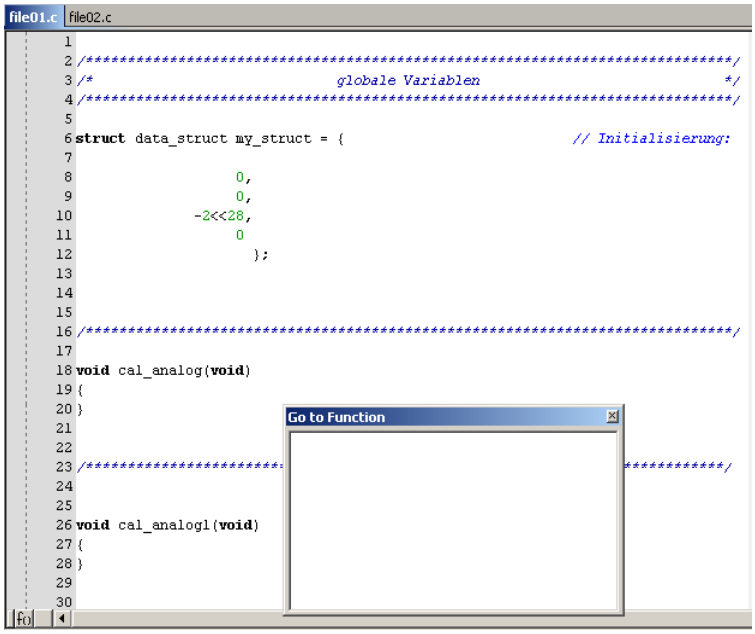
No. *A2*	**Empty Go to Function Window**

Details

Depending on some source code constructions (e.g. using shift operator to initialize a structure element) the Go to Function Window may be empty.
Correct Go to Function Window:



Empty Go to Function Window although there are several functions defined in the active source file:



Workaround
None. The problem will be fixed in the next EWRL78 platform update.

No. *A3*	**Number of total Errors and Warnings doubled by Mistake**

Details

The total number of errors and warnings presented by the compiler is doubled compared to the real amount.

Example:
Although there only one compiler warning (-> line 13 in module main.c), the listed total number of warnings is two.

```
Building configuration: N111209A - Debug
Updating build tree...
main.c
Warning[Pe177]: variable "i" was declared but never referenced

C:\Data\RL78\IAR Bugs\main.c 13

Linking

Total number of errors: 0
Total number of warnings: 2
```

Workaround
None. The problem will be fixed in the next EWRL78 platform update.

No. *A4*	**IarBuild.exe uses incorrect Hardware Multiplier / Divider Support File**

Details

A project build correctly by EWRL78 causes an error message about wrong CPU core if build by command line tool IarBuild.exe:

```
Fatal Error[Pe035]: #error directive: "Functions for RL78_2 core
devices only"
```

Reason is that IarBuild.exe always copies asm-file for RL78-Core2 to support the multiplier/divider to folder <target>\obj.

Workaround

Copy the correct file ‘`hwmdu_LibReplacement.s87`’ from subfolder ‘`\rl78\src\hw_multiply_division_units\RL78_1_core`‘ to subfolder obj of your project and mark it as read-only.

No. *A5*	**Bad initial Stack Size Value in C Project Template**

Details

New C projects created by the project wizard have a bad initial stack size value. Instead of the default size of 128 bytes the string '`"###Uninitialized###"` is used.

Workarounds

1) Enter a legal (= numeric) stack size

   or

2) Create an empty project and add the main module manually

No. *A6*	**Wrong NEAR_CONST Definition in XCL File Template**

IAR Reference: EW24075

Details

In the XCL file template the definition of segment 'NEAR_CONST' is wrong. In case of selecting mirror area 1 (= area 0x1xxxx - 0x1xxxx is mirrored to 0xFxxxx-0xFxxxx) an overflow occurs.

Wrong definition:

```
-Z(DATA)NEAR_CONST=(_NEAR_CONST_LOCATION_START+F0000)-
                   (_NEAR_CONST_LOCATION_END+F0000)
```

Correct definition:

```
-Z(DATA)NEAR_CONST=(_NEAR_CONST_LOCATION_START|F0000)-
                   (_NEAR_CONST_LOCATION_END|F0000)
```

Workarounds

Use a manually corrected XCL file instead of the default one.

No. *A7*	**Wrong 16bit signed Operation using Hardware Multiplier/Divider**

IAR Reference: EW24298

Details

In case of a sequence of multiple 16bit signed division or modulo operations, the result maybe wrong due to missing of CPU register preservation expected by compiler. The problem occurs only for RL78 Core2 devices (e.g. RL78/G14, RL78/F13, and RL78/F14).

Workarounds

Update the assembler modules supporting the RL78. The update patch 'EW24298_ewrl78_hw_multiply_division_units.zip' is available on the European Renesas Download Area.

No. *A8*	**Actual Linker-MAP-File not automatically updated in Editor**

IAR Reference: EW24451

Details

Although the option 'Scan for changed Files' is enabled in EW tool options, a linker map file in HTML format is not automatically updated.

Workarounds
Use text format or update the file manually.

No. *A9*	**Stack Size of 64Byte cannot be permanently defined in IDE**

IAR Reference: EW24879

Details

If CPU core S2 (respectively core 1 in EWL78 V1.30.x) or S3 (respectively core 2 in EWL78 V1.30.x) is selected, it is not possible to permanently define a stack size of 64 bytes. After reopening the options window, the stack size is again 128 byte. Definition of other values is possible.

Workarounds
If a stack size of 64 bytes is mandatory, please defined the stack size directly in the XCL file or the 'Extra Options' field of the linker configuration without using the predefined symbol '_CSTACK_SIZE'.

No. *A10*	**Hardware Multiplier/Divider Unit configuration changed unexpected in case of using two projects within one workspace**

IAR Reference: EW24987

Details

This failure occurs in case two projects are placed within one workspace. Both projects are configured for devices which support a Multiplier/Divider Unit. In such a case the configuration of Multiplier/Divider Unit within the first project may

- automatically change the Multiplier/Divider Unit configuration within the second project and/or
- generate a linker/assembler error due to wrong Multiplier/Divider Unit configuration and missing Multiplier/Divider Unit library functions

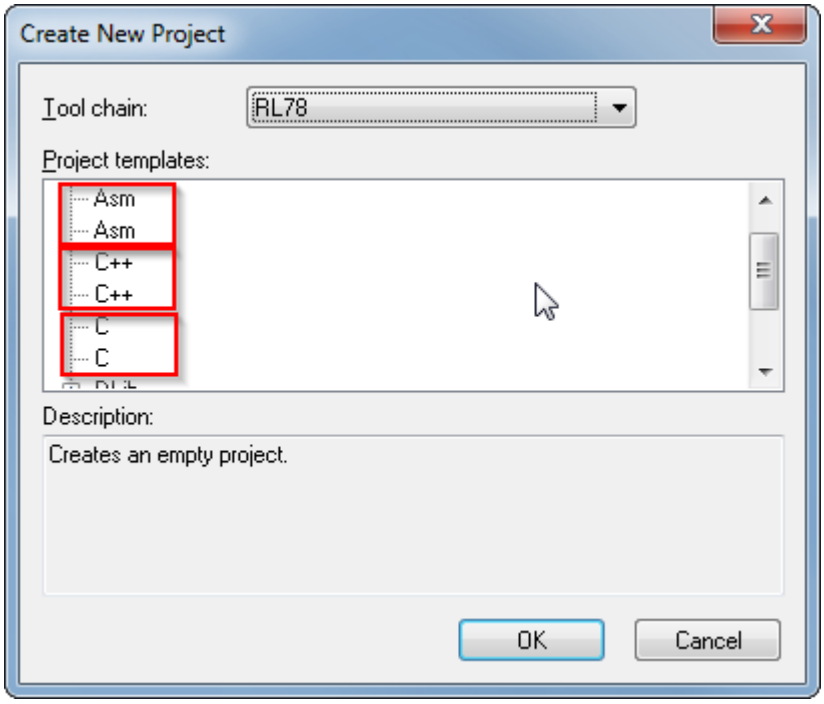
Workarounds
Use different workspaces for each project.

No. *A11*	**Double Entries in Create New Project Dialogue**

IAR Reference: EWxxxxx

Details

Due to a problem in localization of EWRL78 there are double entries for each project template in the 'Create New Project Dialogue':



The created project templates are correct, it is only a display issue.

Workarounds
Please replace some files by the files included in patch 'EWRL78_V1.40.6_UpdateProjectTemplate.zip'.

No. *A12*	**RL78 Mirror Area Configuration changed to default Values**

IAR Reference: EW25197

Details

In case of switching the project configuration in Project Manager, the mirror area start address setting is reset to default value, if different devices are used in different project configurations and a user-defined start address unequal to the default value is used.

Workarounds

1) Please use the default values for mirror start address and size.
   Or
2) Please use different projects rather than different project configurations, if a start address unequal the default value is used.

No. *A13*	**Incorrect Memory Area Definitions in XCL-File Templates**

IAR Reference: EW25271

Details

The allowed memory areas for segments SWITCH, NEAR_ID, SADDR_ID, and DIFUNCT are too small for devices with more than 64KB Flash memory. Although the reserved area for the OCD firmware is located in a higher segment, the area is additionally reserved in the first 64KB segment. This won't cause any runtime problems, but unnecessary limit the allowed memory area for these segments.

Example:

```
-Z(CONST)SWITCH=000D8-0FDFF
-Z(CONST)NEAR_ID=[000D8-0FDFF]/10000
-Z(CONST)SADDR_ID=[000D8-0FDFF]/10000
-Z(CONST)DIFUNCT=[000D8-0FDFF]/10000
```

Workarounds

Correct the XCL-file manually by changing the end-address to 0xFFFF.

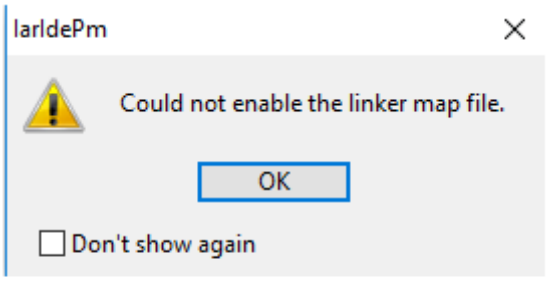
```
-Z(CONST)SWITCH=000D8-0FFFF
-Z(CONST)NEAR_ID=[000D8-0FFFF]/10000
-Z(CONST)SADDR_ID=[000D8-0FFFF]/10000
-Z(CONST)DIFUNCT=[000D8-0FFFF]/10000
```

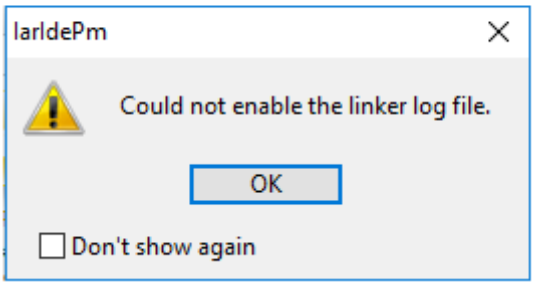
No. *A14*	**Loading of .ipcf file generates warnings**

IAR Reference: IDE-2878

Details

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





Workarounds
Press the “OK” button and ignore the messages.

G) Description of Operating Precautions for the Assembler ARL78

No. B1	**RSEG Directives can not 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
        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. B3	**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
```

No. B5	**Assembler Error caused by Call Frame Information**

Details

An assembler file generated by C compiler with the option "Include call frame information" enabled causes an internal error and/or an assembler error.

Example

```
CFI Resource MACRH:16, MACRL:16, W0:8, W1:8, W2:8, W3:8, W4:8, W5:8
```

Workaround

Disable the compiler option "Include call frame information".

No. B6	**Wrong Code Generated for relative Addressing**

IAR Reference EW24012

Details

In case of using absolute segments (ASEG or ASEGN) the assembler generates wrong hex code for the relative addressing at branch instructions like e.g. BZ, BNZ and BR. An incorrect branch address is calculated.

Example

```
    ASEGN C2:CODE,0x10
m1:
    MOV   a,#1
    CMP   a,#0
    BNZ   m1
    RET
```

List-File:

```
    000014 DF0E         BNZ   m1     <- wrong hex code should be DFFA
```

Workaround

Use relocatable instead absolute segments:

```
    RSEG RCODE:CODE

m1:
    MOV   a,#1
    CMP   a,#0
    BNZ   m1
    RET
```

List-File:

```
    000004 DFFA             BNZ   m1
```

No. B7	**Incorrect Source Line Information**

IAR Reference EW24609

Details

Assembler source code containing end-of-line comments (;) in the non-active part of assembler conditionals (IF/ENDIF) can cause incorrect source positions for subsequent lines. This can affect assembler diagnostics as well as source level debugging of assembler code.

Workaround

None.

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

No. C10	**Inline Assembler: Double defined Label causes internal Compiler Error**

Details

The double definition of an inline assembler label causes an internal compiler error instead of an error message:

```
Internal error [OgModuleLabels::Def::Define]: Label already defined: label1
Fatel error detected aborting
```

Example

```
void test (void)
{
   asm("BR label1 \n"
       "nop       \n"
       "label1:     ");

   asm("nop");

   asm("BR label1 \n"
       "nop       \n"
       "label1:     ");

   asm("nop");
}
```

Workaround

Please use only allowed destination registers according to the instruction set.

No. C13	**#pragma location Directive does not support Unions and Structs**

Details

The #pragma location directive does not support unions and structs. An warning is generated to inform the user:

```
Warning[Pe609]: this kind of pragma may not be used here
```

Example

```
typedef struct
{
    unsigned char no0:1;
    unsigned char no1:1;
    unsigned char no2:1;
    unsigned char no3:1;
    unsigned char no4:1;
    unsigned char no5:1;
    unsigned char no6:1;
    unsigned char no7:1;
} __BITS8;

#pragma location = 0xFFF22;
__sfr __no_init volatile union {
      unsigned char PM2;
      __BITS8       PM2_bit;};
```

Workaround

Use the @ operator instead of #pragma location to define an absolute address:

```
__sfr __no_init volatile union {
         unsigned char PM2;
         __BITS8       PM2_bit;
} @ 0xFFF22;
```

No. C14	**Internal Compiler Error while using __segment_size as memcpy Parameter**

Details

Using intrinsic function __segment_size as size parameter for memcpy function causes an internal compiler error:

`Internal Error: [CoreUtil/General]: Access Violation`

Example

```
#include <string.h>

#pragma segment="MY_SEGMENT_1" __near
#pragma segment="MY_SEGMENT_2" __near

void test(void)
{
    memcpy(__segment_begin("MY_SEGMENT_1"),
           __segment_begin("MY_SEGMENT_2"),
           __segment_size("MY_SEGMENT_2"));
}
```

Workaround

Use a temporary variable:

```
void workaround(void)
{
    size_t my_var;
    my_var= __segment_size("MY_SEGMENT_2");

    memcpy(__segment_begin("MY_SEGMENT_1"),
           __segment_begin("MY_SEGMENT_2"),
           my_var);
}
```

No. C15	**Wrong Code generated for Pointer Access to Extended I/O Area**

Details

Pointer expressions of the kind *(Base - imm*variable) may generate faulty code using the imm[BC] address mode (e.g. instruction `MOV (0x5EA & 0xFFFF)[BC], A`).

Example

```
typedef struct {
  unsigned char  const  e1;
  unsigned short const  e2;
} t_s1;

typedef struct {
  unsigned char      low;
  unsigned char      high;
} Bytes;

typedef union {
  unsigned char      u8_view;
} Byte;

typedef union {
  unsigned short     u16_view;
  Bytes              u08_2_view;
} Word;

__near __no_bit_access __no_init volatile Byte ab1 @ 0xF05EA;
__near __no_bit_access __no_init volatile Word ab2 @ 0xF05EC;

void test (unsigned char p1,const t_s1* p2)
{
 (( (volatile Byte __near *) &ab1)+((-0x250*p1)))->u8_view  = p2->e1;
 (( (volatile Word __near *) &ab2)+((-0x250*p1)))->u16_view = p2->e2;
}
```

Workaround

Use direct I/O access instead of indirect pointer access.

No. C16	**Wrong Code generated for Far Pointer Access**

Details

If a hip optimization level is used, wrong code is generated for the far pointer read access. Register ES is loaded by values 0x0F instead of 0x00.

Example

```
#pragma segment="MYROM1"
#pragma segment="MYRAM1"
#pragma segment="MYROM2"
#pragma segment="MYRAM2"

void test(void)
{
  unsigned char       *ptr_dst;
  unsigned char __far *ptr_src;

  ptr_src = (unsigned char __far *) __segment_begin("MYROM1") ;
  ptr_dst = (unsigned char       *) __segment_begin("MYRAM1") ;
  while( ptr_src < (unsigned char __far *)__segment_end("MYROM1")){
    *ptr_dst++ = *ptr_src++;
  }
  ptr_src = (unsigned char __far *) __segment_begin("MYROM2") ;
  ptr_dst = (unsigned char       *) __segment_begin("MYRAM2") ;
  while( ptr_src < (unsigned char __far *)__segment_end("MYROM2")){
    *ptr_dst++ = *ptr_src++;
  }
}
```

Workarounds

1) Reduce optimization for function to medium:

```
#pragma optimize=medium
void test(void)
{
  …
}
```

2) Use a static source pointer:

```
void test(void)
{
  unsigned char * ptr_dst;
  static unsigned char __far * ptr_src;
  …
}
```

No. C18	**Keyword __no_bit_access does not work on auto Variables**

Details

The keyword __no_bit_access does not work on auto variables. The compiler uses bit instructions although not allowed.

Example

```
char var1;

void test(void)
{
   __no_bit_access char local;

   local = var1;
   var1 = (local | 0x04) & 0xFE;
}
```

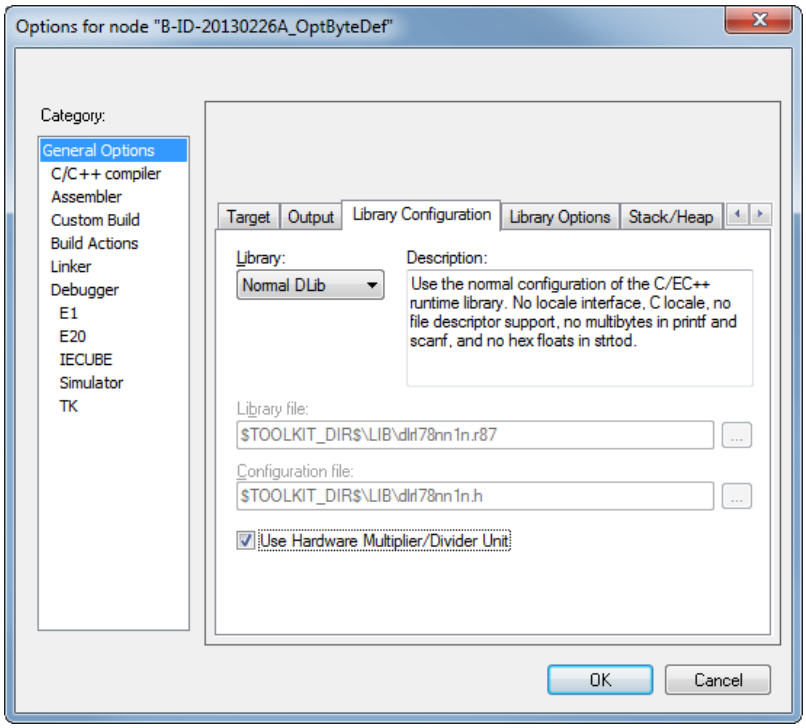
Workaround
None.

No. C20	**Signed Division HWMDU Functions for RL78 Core2 are not Interrupt safe**

Details

The signed division functions using the hardware multiplier/divider used as replacement for the IAR runtime library functions for signed divisions are not interrupt-safe. Affected are all RL78 Core2 devices, e.g. series RL78/G14, RL78/F13, RL78/F14
If a signed division is interrupted and a second division shall be executed in the ISR, the first division result may be wrong.

Example



Workaround

In case of using the Embedded Workbench to build the application:

1) Download the update V1.05 of the HW-MDU functions and add the module 'hwmdu_LibReplacement.s87' to your application.
2) Disable the feature 'Use Hardware Multiplier Divider Unit in IDE.
3) Add the following options to linker 'Extra Options Field:
   -eHWDIV_8_8_8=?UC_DIV_L01
   -eHWSDIV_8_8_8=?SC_DIV_L01
   -eHWSDIV_16_16_16=?SI_DIV_L02
   -eHWSDIV_32_32_32=?SL_DIV_L03

In case of using the command line to build the application:

1) Download the update V1.05 of the HW-MDU functions and replace the existing module 'hwmdu_LibReplacement.s87' in your application.

No. C21	**Illegal 8bit Access to I/O Register allowing only 16bit Access**

Details

Due to a wrong I/O register definition in the device header file the compiler accesses an I/O register allowing only 16bit access by an 8bit access

Example

```
#include <ior5f100le_ext.h>

void test()
{
  SO1 = 0x0101;
  SO1_bit.no8 = 0;
  SO1_bit.no0 = 0;
}
```

Workaround

The problem is fixed by new device header files included in EWRL78 V1.30.2.
As a workaround for previous versions please replace the old header by the new ones.

No. C22	**Wrong Inline Assembler Translation**

Details

An illegal inline assembler statement is in some cases translated to a completely different assembler statement instead of generating an error message.

Example

```
void test(void)
{
  asm("MOVW SP,0xDF82");  // incorrect translation to MOVW AX,N:0xDF82
}
```

Workaround

none

No. C23	**Bit Access generated although Keyword '__no_bit_access' was used**

Details

The compiler doesn't take care on the keyword __no_bit_access in pointer definitions. Although a pointer is correctly defined using the keyword '__no_bit_access', the compiler generates a bit access. For some I/O registers this causes an illegal I/O register access.

Example

```
volatile unsigned short __no_bit_access  v1;
volatile unsigned short __no_bit_access* ptr1 = &v1;

void test (void)
{
    *ptr1  = 0x0123U;
    *ptr1 |= 0x4000U;
}
```

Workaround

Use direct access instead of indirect pointer access

```
void workaround (void)
{
    v1  = 0x0123U;
    v1 |= 0x4000U;
}
```

No. C24	**Wrong indirect post Increment of a Result of a post Increment**

Details

Independent of the selected optimization level the compiler generates wrong code for the indirect post increment of a result of a post increment

Example

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

char   c[2] = {'a','b'};
char *pc[2] = {&c[0],&c[1]};
char **ppc  = &pc[0];

int test(void)
{
    char cc_ret;
    cc_ret = *(*ppc++)++;
    assert(pc[0]==pc[1]);
    return (int)cc_ret;
}
```

Workaround

Use separate statements for post increment:

```
int workaround (void)
{
   …
   cc_ret = *(*ppc);    /* problem */
   (*ppc)++;
   ppc++;
   …
}
```

No. C25	**Wrong Optimization of indirect Variable increment in nested do Loops**

Details

On high optimization, a variable (v) can be optimized incorrectly if *v is incremented with a constant value inside a do loop, *the do loop has a computable trip count, *the do loop is surrounded by another loop with a computable trip count, and *v is not used inside either of the two loops, except for the increment.

Example

```
#include <assert.h>

int i, i0, i1, i2, i3,  i4, i5, i6, i7;

void test(void)
{
  i = i3 = i4 = 0;
  i0 = 2;
  do {
      i1 = 2;
      while (i1--) {
      for (i2 = 0; i2 < 2; i2++) {
        i5 = 2;
        do {
          i6 = 2;
          while (i6--) {
            for (i7 = 0; i7 < 2; i7++)
              i++;
            }
        } while (--i5);
      }
    }
  } while (--i0);
  assert (i == 64);
}
```

Workaround

Reduce the optimization to medium either by using a compiler option or by #pragma optimize.

No. C26	**Internal Compiler Error while copying a packed Structure**

IAR Reference: EW24136

Details

An internal error is generated when trying to copy a packed structure.

Example

```
typedef struct
{
  char e1;
  int  e2;
  int  e3;
} T_S1;

typedef struct
{
  T_S1 s1;
} T_S2;


#pragma pack(1)
typedef struct
{
  T_S2 s1[3];
} T_S3;

#pragma pack()

T_S3 object;

T_S2 func1(void)
{
  T_S2 test;
  test = object.s1[0];
  return test;
}
```

Workaround

None. Issue will be fixed in service pack V1.30.4 (October 2013)

No. C27	**Wrong Code generated for Pointer Comparison with Zero**

IAR Reference: EW24151

Details

Casting a near pointer to a far pointer via an unsigned or signed short can result in a near to far cast instead of a zero-extend cast on medium and higher optimization levels.

Example

```
extern  unsigned short  var1;

void test void)
{
          unsigned short       addr;
    const unsigned short __far *compare;

    *((unsigned short __far *) 0xF800) = 0x0000;
    addr    = (unsigned short) var1;
    compare = (unsigned short __far*)addr;

    if ((unsigned short)0x00 == (*((unsigned short __far*)compare)))
    {
        __asm("BR N:0x2B05");
    }

}
```

Workaround

Use optimization level low:

```
#pragma optimize=low
void test void)
{
 …
}
```

No. C28	**Wrong Code generated for local Variable Access**

IAR Reference: EW24074

Details

The memory tracking of auto variables can optimize away storing of values under rare circumstances at high optimization.

Example

```
unsigned char test (unsigned char lub_1, unsigned char lub_2,
                    unsigned char lub_3, unsigned char lub_4)
{
    unsigned char loc1;
    unsigned char loc2;

    loc1 = 0x90u;
    asm("nop");
    if( lub_1 == 0xFFu ){
        loc1 = 0u;
    }
    if( (loc1&0x80u) == 0u ){
        loc2 = 0xFFu;
        if( lub_2 == lub_3 ){
            loc2 = lub_2;
            if( lub_4 != loc2 ){
                loc1 |= 0x10u;
            }
        }
        if( (loc2&0xF0u) == 0x00u ){
            loc1 |= loc2;
        }else{
            loc1 = 0x90;
        }
    }
    return loc1;
}
```

Workaround

Use optimization level low:

```
#pragma optimize=medium
void test void)
{
 …
}
```

No. C29	**Wrong Prototype Description in Compile Manual of Function __segment_size**

	IAR Reference: EW24186
	
	Details
	
	At page 121 of the RL78 C/C++ Compiler Reference Guide (2nd Edition) the prototype of function __segment_size is described as
	
	`size_t * __segment_size(char const * segment)`
	
	But the correct prototype is:
	
	`size_t __segment_size(char const * segment)`
	
	Workaround
	
	Please use the corrected prototype. An updated explanation is given in the release notes V1.30.5

No. C30	**Wrong Code generated for local far Pointer loaded via far Pointer**

IAR Reference: EW24198

Details

Loading a far pointer via a far pointer can generate faulty code.

Example

```
typedef struct
{
   unsigned char *    data;
   unsigned char      a;
} st;


st  s1;


void test (unsigned char buffer[], unsigned char n)
{
   unsigned char i;
   for (i = 0U; i < s1.a; i++)
   {
      *(s1.data + i) = buffer[i + 1U];
   }
   return;
}
```

Workaround

```
void copy (unsigned char buffer[], unsigned char n)
{
   unsigned char i;
   static unsigned char * ptr;
   for (i = 0U; i < s1.a; i++)
   {
      ptr  = (s1.data + i);
      *ptr = buffer[i + 1U];
   }
   return;
}
```

No. C31	**Unclear Description of Parameter Passing for Structure Types in Compiler Manual**

IAR Reference: EW24223

Details

At page 108 of the RL78 C/C++ Compiler Reference Guide (2nd Edition) parameter passing to function is described. It is described that structure types parameters are passed via stack except the size is 1,2,4 and 4 bytes:

```
Structure types: struct, union, and classes, except structs and unions
of sizes 1, 2, and 4
```

This is correct, but additionally the structure type element must be word aligned. The alignment of the element is defined by the data type of the largest member.

Example

```
typedef struct  {
  unsigned char e1;
  unsigned char e2;
} s1_TYPE;
```

The above structure is passed via stack as only byte aligned elements are included.

Workaround

Include the structure type element in a union to force word alignment:

```
typedef union  {
  struct {
  unsigned char e1;
  unsigned char e2;
  };
  unsigned short dummy;
} s1_TYPE;
```

An updated explanation is given in the release notes V1.30.5

No. C32	**Internal Compiler Error by erroneous Bitfield Definition**

IAR Reference: EW24368

Details

After the compiler generates the error, Error[Pe168]: a function type is not allowed here, for an erroneous bitfield type it can produce an internal error

```
Internal Error: [Front end]: assertion failed:
set_field_size_and_alignment: bad curr_container_avail_bits adjustment
```

Example

```
typedef void (func_type) ();

struct s {
 func_type f:32;
};

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

Workaround
Use only data type attributes for bitfield definitions.

No. C33	**Internal Compiler Error after several ordinary Errors**

IAR Reference: EW24366

Details

After a sequence of several ordinary errors an internal compiler error may occur.

```
Internal Error: [PaType - MemoryAttribute]: no memory attribute set
```

Workaround
Fix the ordinary errors.

No. C34	**Internal Compiler Error if a Function of more than 255 Parameters is used**

IAR Reference: EW24359

Details

A function receiving more than 250 parameters may generate an internal error in the compiler if the object format is UBROF.

```
Internal Error: [Front end]: assertion failed:
set_field_size_and_alignment: bad curr_container_avail_bits adjustment
```

Example

```
#define PAR1 int, int, int, int, int, int, int, int, int, int
#define PAR2 PAR1, PAR1, PAR1, PAR1, PAR1, PAR1, PAR1, PAR1, PAR1, PAR1
#define PAR3 PAR2, PAR2, PAR2, PAR2, PAR2, PAR2, PAR2, PAR2, PAR2, PAR2

extern void func (PAR3);

#define ARG1 0,1,2,3,4,5,6,7,8,9
#define ARG2 ARG1, ARG1, ARG1, ARG1, ARG1, ARG1, ARG1, ARG1, ARG1, ARG1
#define ARG3 ARG2, ARG2, ARG2, ARG2, ARG2, ARG2, ARG2, ARG2, ARG2, ARG2

void caller(void)
{
  func (ARG3);
}
```

Workaround
Reduce number of Parameters.

Since V1.40.1 a new error message is generated to inform the user that functions with more than 250 parameters are not supported

No. C35	**Internal Compiler Error after illegal enum-Value Error**

IAR Reference: EW24363

Details

An error for an enum constant with an illegal value may be followed by an internal error.

```
Internal Error: [Front end]: assertion failed at:
"..\..\Translator\compiler_core\src\parser\edg\const_ints.c", line 360
```

Example

```
struct B {};
struct NonPOD : B {};

struct A
{
  static int check(...);
  static NonPOD GetNonPOD(void);
  enum { value = sizeof(A::check(A::GetNonPOD())) };
};


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

Workaround
Use a valid enum value.

No. C36	**Internal Compiler Error after Error [Pe078]**

IAR Reference: EW24362

Details

After the error, Error [Pe078]: a parameter declaration may not have an initializer, the compiler may produce an internal error.

```
Internal Error:
[CoreUtil/General]: Access violation (0xXXXXXXXX) at XXXXXXXX (reading
from address 0x40)
```

Example

```
#ifndef __USER_LABEL_PREFIX__
#define PREFIX ""
#else
#define xstr(s) str(s)
#define str(s)  #s
#define PREFIX  xstr(__USER_LABEL_PREFIX__)
#endif

typedef unsigned short int __uint16_t;
enum
{
  _ISupper = (1 << (0)), _ISlower = (1 << (1)), _ISalpha =
    (1 << (2)), _ISdigit = (1 << (3)), _ISxdigit = (1 << (4)), _ISspace
    =
    (1 << (5)), _ISprint = (1 << (6)), _ISgraph = (1 << (7)), _ISblank
    =
    (1 << (8)), _IScntrl = (1 << (9)), _ISpunct = (1 << (10)), _ISalnum
    =
    (1 << (11))
};
typedef __uint16_t __ctype_mask_t;
extern const __ctype_mask_t *__C_ctype_b;
extern
__typeof (__C_ctype_b)
   __C_ctype_b __asm__ (PREFIX "__GI___C_ctype_b")
  __attribute__ ((visibility ("hidden")));
     static const __ctype_mask_t __C_ctype_b_data[] = {
     };

const __ctype_mask_t *__C_ctype_b = __C_ctype_b_data + 128;
extern
__typeof (__C_ctype_b)
     __EI___C_ctype_b __attribute__ ((alias ("" "__GI___C_ctype_b")));


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

Workaround

Correct the parameter declaration.

No. C37	**Internal Compiler Error after Error [Pe066] (1)**

IAR Reference: EW24358

Details

After the error Error [Pe066]: enumeration value is out int range, the compiler may produce an internal error.

```
Internal Error:
[Front end]: assertion failed at:”...\...\xxx.c", line xxx
```

Example

```
enum err {
  err_IO = 0x8a450000,
  err_NM,
  err_EOF,
  err_SE,
  err_PT
};
static enum err E_;
int error()
{
  switch (E_) {
    case err_IO : break;
    case err_NM : break;
    case err_EOF : break;
    case err_SE : break;
    case err_PT : break;
    default : return 0;
}

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

Workaround
Enable IAR ANSI C extensions or reduce the enumeration value to an integer.

No. C38	**Internal Compiler Error after Error [Pe066] (2)**

IAR Reference: EW24357

Details

After the error Error[Pe066]: enumeration value is out int range, the compiler may produce an internal error.

```
Internal Error:
[Front end]: assertion failed at: ”..\..\xxx.c", line xxx
```

Example

```
typedef enum OMX_ERRORTYPE
{
  OMX_ErrorNone = 0,
  OMX_ErrorInsufficientResources = 0x80001000
} OMX_ERRORTYPE;

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

Workaround
Enable IAR ANSI C extensions or reduce the enumeration value to an integer.

No. C39	**Internal Compiler Error: Stack Overflow**

IAR Reference: EW24353

Details

Very deep nestlings of struct 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. C40	**Keyword '__no_bit_access' fails at explicit cast to 16bit data type**

IAR Reference: EW24389

Details

The keyword __no_bit_access does not work correctly with an expression like :

```
*(volatile ushort __no_bit_access *)(ushort)(0xFFF10U)
```

The compiler will use a bit access for an access to above and this causes a problem, if a bit access is not allowed at this address.

Example:

```
#define MYREGISTER *(__no_bit_access volatile unsigned short *)(unsigned
short)(0xFFF10U)


void test (void)
{
  MYREGISTER = (unsigned short) (((unsigned short)(MYREGISTER)) & ~(0x80u));
}
```

Workaround
None. Will be fixed in next update.

No. C41	**Internal Compiler Error at Function defined by Macros**

IAR Reference: EW24361

Details

An internal compiler error may occur, if a function is defined by several macros.

Example:

```
#define main()
int main

#define mainbody () { return 0; }
mainbody
```

Workaround
Define both macros before using them:

```
#define main()
#define mainbody () { return 0; }

int main
mainbody
```

No. C42	**Wrong Code generated for indirect Access to Structure Member**

IAR Reference: EW24383

Details

An expression (e) containing an auto variable (v) and one or more indirect accesses could be optimized incorrectly, if the expression was preceded by a statement where v is assigned the result of a function call, and e contains the only use of v (before v is assigned another value) and if the compiler optimization 'high size' or 'high balanced/speed without function inlining is used.

Example:

```
#include <stdio.h>

typedef struct {int a; int b;} SP;
SP st1 = {1,2}, st2 = {1,2};

typedef struct {void *p;} VP;
VP st2x = {&st2};

unsigned int ex(SP * p1, SP * p2){
   p2->b++;
   return (p1->a)+(p2->b);
}

int sub( void )
{
  int16_t ret;
  VP *px = (VP*) &st2x;
  SP *ps = (SP*) px->p;
  ret  = ex(&st1, px->p);
  ret |= ps->b << 8;
  return(ret);
}

int ans;

int main (void)
{
  ans = sub();
  printf("ret = %x\n", ans);
}
```

Workaround
Use not the above listed optimization setting.

No. C43	**Internal Compiler Error in case of using nested Boolean Expressions**

IAR Reference: EW24511

Details

In case of using optimization level low nested boolean expressions may in rare cases cause an illegal state error.

Example:

```
extern int foo (int);


int test (int a, int b, int c)
{
  foo (1 > (2 > c));
  return (a);
}
```

Workaround
Avoid nested Boolean expression or increase optimization level.

No. C44	**Internal Compiler Error after Error Pe066**

IAR Reference: EW24357

Details

The compiler can produce an internal error after the error, Error [Pe066]: *enumeration value is out of "int" range*, is produced is the option `–strict` is used.

Example:

```
  enum E { A = 0x80000000, B = 0 };

  int test ( void)
  {
    if (sizeof (E) != 4)
      return 1;
    else
      return 0;
  }
```

Workaround
Use C++ compiler (-> compiler option *--ec++*) or enable ANSI C extensions (option *–e*)

No. C45	**Near-Call in Floating Point Library causes a Linker Error**

IAR Reference: EW24573

Details

Due to a wrong near-function call in the floating-point library, a linker error can occur if the compiler option '--generate_far_runtime_library_calls' is used:

```
Error[e18]: Range error, Limit exceeded

  Where $ = ?F_ADD + 0xFFFFFFD8  [0x10019]
            in module "?FLOAT_ADD_SUB" (C:\Program Files (x86)\IAR
Systems\Embedded Workbench 6.5_EWRL78_1305\rl78\LIB\dlrl78fn2nf.r87),
            offset 0x19 in segment part 2, segment XCODE
  What: __iar_norm_arg [0x104CC]
  Allowed range: 0x0 - 0xFFFF
  Operand: __iar_norm_arg [0x104cc]
           in module ?SUBN_ARG (C:\Program Files (x86)\IAR
Systems\Embedded Workbench 6.5_EWRL78_1305\rl78\LIB\dlrl78fn2nf.r87),
           Offset 0x0 in segment part 2, segment XCODE
```

Example:

```
unsigned int   i1;
         int   i2;
         int   i3;
         float f1;

void test( void )
{
  i3 = (int)(i1 – f1 * (i2 – 600))
}
```

Workaround

Replace the standard library by a customer one, where the issue is fixed. If further support is needed please contact the Renesas Software-Tool-Support-Team

No. C46	**Wrong Code generated while Copying a 1-Bit Bitfield**

IAR Reference: EW24645

Details

Assigning a value from one 1-bit bitfield to another 1-bit bitfield can fail if the byte offset of the bitfield in of struct is not zero and an optimization level medium or higher is used.

Example

```
typedef struct
{
 unsigned long u32var1;
 unsigned char u1var6_1:1;
 unsigned char u1var6_2:1;
 unsigned char u1var6_3:1;
 unsigned char u1var6_4:5;
}s1_T;

void test(s1_T * in, s1_T * out)
{
  out->u1var6_1 = in->u1var6_1;
  out->u1var6_2 = in->u1var6_2;
  out->u1var6_3 = in->u1var6_3;
  out->u1var6_4 = in->u1var6_4;
}
```

Workaround
Lower optimization level to medium or low.

No. C47	**Keyword 'const' disables #pragma default_variable_attribute Directive**

IAR Reference: EW24683

Details

Using the keyword 'const' disables the #pragma default_variable_attribute directive.

Example

```
#pragma default_variable_attributes = __root
  const int c3=0x33;
#pragma default_variable_attributes =
```

In above example the variable c3 is defined without object attribute 'root'

Workaround
Use extended keyword instead of #pragma directive to define an attribute.

No. C48	**MISRA C 2004 Rule 10.6 not triggered**

IAR Reference: EW24733

Details

The compiler does not check MISRA-C 2004 rule 10.6 correctly. It bases the check on the usage of the constant instead of on the type of the constant.

Example:

```
#define UNSIGNED_CHAR_C  0x12
#define UNSIGNED_SHORT_C 0x1234
#define UNSIGNED_LONG_C  0x12345678

unsigned char  var1 = UNSIGNED_CHAR_C;  /* Error [Pm127]: */
unsigned short var2 = UNSIGNED_SHORT_C; /* no error MISRA C 2004 */
unsigned long  var3 = UNSIGNED_LONG_C;  /* no error MISRA C 2004 */
```

In above example error Pm127 should be triggered three times instead of only one.

Workaround
None; it will be fixed in next update.

No. C49	**Wrong Result of signed Integer Division**

IAR Reference: EW24778

Details

The result of signed division might be wrong, if one of the operands is a constant that is not of the same type as the other operand.

Example:

```
signed short v1;
signed short v2;

void test (void)
{
      v1 = 1;
      v2 = (signed short)(((signed long)40000 * v1)/200);
}
```

Workaround
None.

No. C50	**Manual Error in Description of Option '--disable_div_mod_instructions'**

IAR Reference: EW24831

Details

On page 217 for the compiler option --disable_div_mod_instructions it is described "Disabling these instructions will make interrupts faster." That is incorrect. It is the opposite since a library call is done when instruction is disabled.

Example:

Workaround
Manual be corrected in future update of the compiler manual.

No. C51	**Huge constant Data placed in Segment 'NEAR_CONST'**

IAR Reference: EW24860

Details

Although a huge constant data is defined correctly, it is located in segment NEAR_CONST where only near constant data shall be placed.

Example:

```
__huge const unsigned short c1 = 0x1234;
```

Workaround
If possible use far constant data.

```
__far const unsigned short c1 = 0x1234;
```

Will be fixed in next update

No. C52	**Stack Content can be corrupted by ISR**

IAR Reference: EW24898

Details

Due scheduling error in the optimizer, the stack content can be corrupted if stack is used for temporary storage in a function and an interrupt occurs also using temporary storage

Example:

In below sample the address of data located on stack is stored in register HL to access it indirectly. Due to the error the stack pointer is modified to free the stack size before the last access to the data is finished. If now an interrupt using stack area occurs between modification of stack pointer and data processing, the data is corrupted:

```
\   00003D 16                   MOVW      HL, AX              ;; 1 cycle
\   00003E 710103               MOV1      S:0xFFF03.0, CY     ;; 2 cycles
\   000041 A7                   INCW      HL                  ;; 1 cycle
\   000042 1002                 ADDW      SP, #0x2            ;; 1 cycle
```

If an interrupt using stack memory occurs here, data used in the next indirect memory access are corrupted:

```
\   000044 71B4                 MOV1      CY, [HL].3          ;; 1 cycle
\   000046 710103               MOV1      S:0xFFF03.0, CY     ;; 2 cycles
```

The correct code should be:

```
\   000040 16                   MOVW      HL, AX              ;; 1 cycle
\   000041 A7                   INCW      HL                  ;; 1 cycle
\   000042 71B4                 MOV1      CY, [HL].3          ;; 1 cycle
\   000044 710103               MOV1      S:0xFFF03.0, CY     ;; 2 cycles
\   000047 1002                 ADDW      SP, #0x2            ;; 1 cycle
```

Workaround
Avoid optimization level high balanced and high speed.

No. C53	**MISRA C Rule 10.1 triggered by Mistake**

IAR Reference: EW24883

Details

In below sample MISRA-C:2004 rule 10.1 is falsely triggered for an implicit cast from a _Bool type to an integer of float type.

Example:

```
#include <stdbool.h>

bool boTest;

void test (void);

void test (void)
{
  if (boTest == false) {
  /* Error Pm128: illegal implicit conversion from underlying MISRA
     type  “_Bool” to “int” (MISRA C 2004 rule 10.1
  */
     …
  }
}
```

Workaround
This issue will be fixed in EWRL78 V2.10. Until then please suppress this error message for the corresponding code lines:

Example:

```
void workaround (void)
{
  #pragma diag_suppress=Pm128
  if (boTest == false) {
  #pragma diag_default=Pm128
     …
  }
}
```

No. C54	**Internal Error at Comparison of near Pointer**

IAR Reference: EW24995

Details

Comparing of a near data pointer with (void near*) -1 can generate an internal compiler error at using optimization level 'low':

```
Tool Internal Error:
Internal Error: [CMPW]:
Diagnostics: Immediate out of range
P0: 1048575
P1: 0
```

Example:

```
static volatile unsigned short int  * ptr;

unsigned  char foo1 ( void )
{
  unsigned char status = 0x40u;

  ptr = ( unsigned short int *) 0xFFFFu;

      if ( ptr != ( unsigned short int *) 0xFFFFu)
      {
        status =   0x42u;
      }
  return status;
}
```

Workarounds

1) Increase optimization level to 'medium' or higher.

   or

2) Instead of casting "-1" to a near pointer, cast the near pointer to (un)signed short:

```
if ( (unsigned short) ptr == 0xFFFF )
{
   ...
}
```

No. C55	**Internal Error at Bitfield Assignment**

IAR Reference: EW24994

Details

An assignment, where a signed / unsigned bitfield member is assigned a constant value, followed by an assignment to an signed / unsigned bitfield member in the same variable, can in some cases trigger an internal error at using a high optimization level:

```
Internal Error: [CoreUtil/General]: integral and fatal error detected, aborting.
```

Example:

```
void test ( void )
{
   struct str {
      struct nest_str {
          signed int b1 : 3 ;
        unsigned int    : 1 ;
        unsigned int b2 : 2 ;
      } bf ;
      int dmy ;
   } s = { { 0, 0 }, 0 } ;

   s.bf.b1 = 1 ;
   s.bf.b2 = 1 ;

   if ( s.bf.b1 != 1 || s.bf.b2 != 1 ) {
      asm("nop");
   }
   else {
      asm("nop");
      asm("nop");
   }
}
```

Workarounds
Lower optimization level.

No. C56	**Internal Error at Switch Statement**

IAR Reference: EW24996

Details

Tail recursive calls before a switch statement can trigger an internal error:

```
Internal Error: [CoreUtil/General]: Access violation (0xc0000005) xxxxxxxx
Fatal error detected, aborting.
```

Example:

```
int val;

void sub( int arg )
{
   if( arg ) {
      sub( arg - 1 ) ;

      switch( arg ) {
         case 1 :
            val = 1 ;
            break ;
         case 2 :
            val = 2 ;
            break ;
         default :
            break ;
      }
   }
}
```

Workarounds
Lower optimization level.

No. C57	**Wrong code generated for inline String Literal Copying**

IAR Reference: EW24999

Details

Copying a string literal to a char array using strcpy generate faulty inline code if the literal starts at an odd address. Inline code is typically generated at optimization level high balanced and high speed.

Example:

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

typedef struct {
      char mem;
      char name[10];
} ST01;

ST01 g01;


void test ( void );

void test ( void )
{
      ST01 b01;

      strcpy( b01.name, "abc");
      strcpy( g01.name, "abc");
}
```

Workarounds
Lower optimization level or use optimization high size.

No. C58	**Wrong Code generated for Multiple Bitfield Assignments of Constant**

IAR Reference: EW25000

Details

Bitfield variables can in some rare cases be optimized incorrectly on high optimization level, if there are multiple assignments to two or more bitfield members and at least one of them is assigned a constant.

Example:

```
#include <stdio.h>

struct srt_dat_t {
        unsigned int      bit1 : 3 ;
        unsigned int           : 3 ;
        unsigned int      bit2 : 5 ;
} ;

void func( struct srt_dat_t arg1 );

void func( struct srt_dat_t arg1 )
{
    arg1.bit1 = !0 ;
    arg1.bit2 = !0 ;

    ++arg1.bit1;
    ++arg1.bit2;

    if (!((arg1.bit1 == 2)  &&  (arg1.bit2  == 2))) {
           printf( "\targ1.bit1[2]--->[%d]\n", arg1.bit1 ) ;
           printf( "\targ1.bit2[2]--->[%d]\n", arg1.bit2 ) ;
    }
}
```

Workarounds

Replace pre-increment by simple addition by one:

```
   arg1.bit1 = arg1.bit1 + 1;
   arg1.bit2 = arg1.bit2 + 1;
```

No. C59	**Wrong Code generated for multiple Bitfield Assignment in one Statement**

IAR Reference: EW25002

Details

Assignments where multiple bitfields are assigned in one can be optimized incorrectly on high optimization.

Example:

```
void test ( void )
{
  SRT_DAT srt1;

  srt1.bt1 = srt1.bt4 = srt1.bt6 = srt1.bt7 = 1;

  if (!(srt1.bt1==1 && srt1.bt4==1 && srt1.bt6==1 && srt1.bt7==1 )) {
    printf( "%-12s %04d:NG\n", __FILE__, __LINE__ ) ;
  }else{
    printf( "%-12s %04d:OK\n", __FILE__, __LINE__ ) ;
  }
  printf( "\tsrt1.bt1[1]--->[%d]\n", srt1.bt1 ) ;
  printf( "\tsrt1.bt4[1]--->[%d]\n", srt1.bt4 ) ;
  printf( "\tsrt1.bt6[1]--->[%d]\n", srt1.bt6 ) ;
  printf( "\tsrt1.bt7[1]--->[%d]\n", srt1.bt7 ) ;
}
```

Workarounds

Use separated statements:

```
void test ( void )
{
  SRT_DAT srt1;

  srt1.bt1 = srt1.bt4 = srt1.bt6 = 1;
  srt1.bt7 = 1;

  if (!(srt1.bt1==1 && srt1.bt4==1 && srt1.bt6==1 && srt1.bt7==1 )) {
    printf( "%-12s %04d:NG\n", __FILE__, __LINE__ ) ;
  }else{
    printf( "%-12s %04d:OK\n", __FILE__, __LINE__ ) ;
  }
  printf( "\tsrt1.bt1[1]--->[%d]\n", srt1.bt1 ) ;
  printf( "\tsrt1.bt4[1]--->[%d]\n", srt1.bt4 ) ;
  printf( "\tsrt1.bt6[1]--->[%d]\n", srt1.bt6 ) ;
  printf( "\tsrt1.bt7[1]--->[%d]\n", srt1.bt7 ) ;
}
```

No. C60	**Wrong Code generated for Calculation depending on Overflow of smaller Datatype**

IAR Reference: EW25004

Details

On high optimization levels loops containing an expression where the result of the expression depends on unsigned overflow of a smaller type can in some case be optimized incorrectly.

Example:

```
#include <stdio.h>

unsigned char globalTMP ;

int main (void)
{
  unsigned int i, res1 ;
  globalTMP = 0;
  res1 = 0;

  for( i = 64 ; i < 65 ; i++ ) {
    globalTMP = i * 4 ;
    res1 = i * 4 + globalTMP ;
  }
  printf("res = %d\n",res1);
  return(0);
}
```

Workarounds
Lower optimization level.

No. C61	**Wrong Code generated for Return-Value including Assignment**

IAR Reference: EW25006

Details

Instead of assigning a correct return value, the input parameter value is used as return value in the following example.

Example:

```
int b = 1;

int func( int a )
{
  return( a = b++ );
}
```

Workarounds
Use separated statements:

```
int func( int a )
{
  a = b++;
  return(a);
}
```

No. C62	**Inserted NOP after DIVWU/DIVHU Instruction moved**

IAR Reference: EW25080

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 instruction scheduler will in some cases move an instruction in between the DIVHU/DIVWU instruction and the NOP. This happens only using optimization level high.

Example:

Workarounds
Disable scheduling by using compiler option `--no_scheduling` .

No. *C63*	**User defined Stack Size overwritten by Default Size**
	IAR Reference: EW25088 *Details* For RL78 Core S1 and S2 devices the user defined stacksize is overwritten by default value after re-opening a project. *Workarounds* None.

No. *C64*	**Wrong Code generated for direct Access to of Hardware Multiplier / Divider Register**

IAR Reference: EW25100

Details

For RL78 Core S2 devices wrong code may be generated for direct access to register MDAH and MDAL of the Hardware Multiplier/Divider directly. Access via included ASM functions or replacement of runtime library functions are not affected.

Example

```
#include <stdint.h>
#include <stdio.h>
#include <ior5f109aa.h>
#include <ior5f109aa_ext.h>

uint32_t foo1( uint32_t value )
{
   uint32_t result;
   MDUC  = 0x80;
   MDAH  = (uint16_t)(value >> 16);
   MDAL  = (uint16_t)(value & 0xFFFF);
   MDBH  = 0;
   MDBL  = 1000;
   DIVST = 1;
   while( DIVST == 1 )  {
   }
   result  = ((uint32_t)MDAH) << 16;
   result += (uint32_t)MDAL;
   return result;
}
```

Workaround

Use dummy function to read result and make sure that function-inlining is disabled:

```
#pragma optimize=no_inline
uint32_t dummy(void)
{
   uint32_t result;
   while( DIVST == 1 ) {
   }
   result  = ((uint32_t)MDAH) << 16;
   result += (uint32_t)MDAL;
   return result;

}
uint32_t foo1( uint32_t value )
{
   MDUC   = 0x80;
   MDAH   = (uint16_t)(value >> 16);
   MDAL   = (uint16_t)(value & 0xFFFF);
   MDBH   = 0;
   MDBL   = 1000;
   DIVST  = 1;
   result = dummy2();
   return result;
}
```

No. *C65*	**Internal Compiler Error using different I/O Register Definitions in different Modules**

IAR Reference: EW25225

Details

In case of using different definitions for the same I/O register in different modules and enabling multiple file compilation, an internal compiler error occurs.

Example

Module 1:

```
__sfr  __no_init volatile unsigned char MK2L @ 0xFFFD4u;
void f1 (void)
{
  MK2L = 0xFF;
}
```

Module 2:

```
#include <ior5f109ge.h>
extern void f1 (void);
void main (void)
{
  f1 ();
  while(1) {
    MK2L = 0x00;
  }
}
```

Workaround

Use only one common I/o register definition:

Module 1:

```
#include <ior5f109ge.h>

void f1 (void)
{
  MK2L = 0xFF;
}
```

No. *C66*	**Compiler Error Pe147 triggered by Mistake**

IAR Reference: EW25227

Details

The compiler emits a spurious error when processing an out of class definition of a const or volatile member function of a class with a class memory.

Example

```
struct __far S
{
  int fun() const;
  int mS;
};

int
S::fun() const
{
  return mS;
}
```

Workaround
None.

No. C67	**Internal Compiler Error using Datatype 'long long' as Switch-Expression**

IAR Reference: EW25270

Details

In case of using datatype long long (=64 bit) for a switch expression, an internal compiler error occurs.

Example

```
long long int  my_very_long_int;

void test ( void )
{
  switch (my_very_long_int) {
  case 1:
    break;
  case 2:
    break;
  }
}
```

Workaround

Use a smaller datatype for the switch expression

No. C68	**Internal Compiler Error using explicit double Casting**

IAR Reference: EW25540

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. C69	**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. C70	**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. C71	**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. C72	**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. C73	**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. C74	**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. C75	**Switch state in recursive function generates an internal error**

IAR Reference: EW26549/EWRL78-585

Details

On optimization level -Om or higher the Compiler generates an internal error in case a function 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. C76	**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

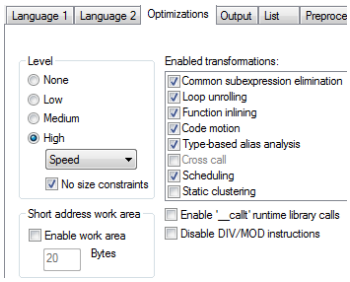
No. C77	**Sign-extending a signed int/short register variable to a long can destroy a variable**

IAR Reference: EWRL78-597

Details

Sign-extending a signed int/short register variable to a long can destroy the register variable if it is located in register AX.

Example



File: file1.c

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

void main(void);
sint16 Mag3DPosDet_CalcSphericalCoordinateTheta(sint16, sint16, sint16);

sint16 X, Y, Z, result;

void main (){
  X = 72;
  Y = -258;
  Z = -130;
  result = Mag3DPosDet_CalcSphericalCoordinateTheta(X, Y, Z);
  if ( result == 1158 ){
    printf("OK”);
  }else{
    printf("FAILED”);
  }
}
```

File: file2.c

```
#include <math.h>
#include "type.h"

sint16 Mag3DPosDet_CalcSphericalCoordinateTheta(sint16, sint16, sint16);
sint16 Mag3DPosDet_CalcSphericalCoordinateTheta(sint16 sComponentX, sint16 sComponentY,
sint16 sComponentZ)
{
    float32 fParam;  uint32 ulHelp;

    ulHelp = (uint32)((((sint32)sComponentX) * ((sint32)sComponentX)) +
                                    (((sint32)sComponentY) * ((sint32)sComponentY)));
    if(ulHelp > 0x00ul){
        fParam = sqrtf(ulHelp);
        fParam = (float32)sComponentZ / fParam;
        fParam = atanf(fParam) * (float32)(57.29577951);
    }
    else{
        fParam = (float32)(900) / ((float32)(10u));
    }
   return ((sint16)(900) - (sint16)(fParam * (float32)(10u)));
}
```

Workaround

None

No. C78	**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. C79	**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. C80	**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. C81	**%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. C82	**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. C83	**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. C84	**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. C85	**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. C86	**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. C87	**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. C88	**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. C89	**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. C90	**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. C91	**long long int switch-statement causes internal error**

IAR Reference: EWRL78-642

Details

Switch statements on type long long is not supported by the compiler.

Example

```
#include <stdio.h>
int sub( unsigned long long data );
int sub( unsigned long long data )
{
        switch( data ) {
                case 0 :
                        return( 10 ) ;
                case 1 :
                        return( 20 ) ;
                default : break ;
        }
        return( 0 ) ;
}
int main( void ){
        printf( "%-12s %04d:OK\n", __FILE__, __LINE__ ) ;
}
```

Workaround

None

No. C92	**Operation with a long long int type member of structure causes internal error**

IAR Reference: EWRL78-639

Details

If the operand is a member of a structure, and has long long int type, the multiple arithmetic operations causes the tool internal error.

Example

```
#include <stdio.h>

typedef struct s_tag {
        long long int mem01;
        long long int mem02;
} STRCT_A;

STRCT_A stdata;

long long int sub(long long int arg );
long long int sub(long long int arg )
{
        return arg;
}

int main( void )
{
        long long int result;
        long long int data01;
        long long int data02;
        int flg=0;

        data01 = 33 ;
        data02 = 3 ;

        stdata.mem01 = sub(data01) ;
        stdata.mem02 = sub(data02) ;

        result = stdata.mem01 - stdata.mem02 ;
        if ( result != 30 )     flg ++;

        result = stdata.mem01 * stdata.mem02 ;
        if ( result != 99 )     flg ++;

        result = stdata.mem01 / stdata.mem02 ;
        if ( result != 11 )     flg ++;

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

        return( 0 ) ;
}
```

Workaround

Disable Compiler optimization cse (--no_cse)

No. C93	**An extraneous memory read can occur when you read a volatile bitfield**

IAR Reference: EWRL78-707

Details

Reading of a volatile bitfield might lead to an extraneous memory read.

Example

```
typedef struct
{
  unsigned char no0:1;
  unsigned char no1:1;
  unsigned char no2:1;
  unsigned char no3:1;
  unsigned char no4:1;
  unsigned char no5:1;
  unsigned char no6:1;
  unsigned char no7:1;
} __BITS8;

__saddr __no_init volatile union { __BITS8 P2_bit; } @ 0xFFF02;

int main( void )
{
  P2_bit.no7 = ~P2_bit.no7;
  return 0;
}
```

The extraneous memory read can be found within the disassembly listing:

```
        RSEG CODE:CODE:NOROOT(0)
main:
        PUSH    BC                  ;; 1 cycle
        MOV1    CY, S:0xFFF02.7     ;; 1 cycle
        MOV1    CY, S:0xFFF02.7     ;; 1 cycle
        NOT1    CY                  ;; 1 cycle
        MOV1    S:0xFFF02.7, CY     ;; 2 cycles
        CLRW    AX                  ;; 1 cycle
        POP     BC                  ;; 1 cycle
        RET                         ;; 6 cycles
```

Workaround
None

No. C94	**Optimizer considers all long long constants as equal**

IAR Reference: EWRL78-760

Details

The optimizer considers all labels of long long constants (i.e. internally generated long long constants) to be equal which can cause cross call and cross jumping to fail.

Workaround
None

No. C95	**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.

I) Description of Operating Precautions for Linker XLINK

No. D5	**ELF Output File Format: Error e113 'Illegal ELF Register'**
	IAR Reference EW24254 *Details* The usage of the compiler option '—worksegment' causes an 'illegal ELF register error if the output file format ELF is selected: `Fatal Error[e113]: Corrupt input file: "Illegal ELF-register." in module xxx ( <path>\xxx.r87 )` *Workaround* Please avoid compiler option '—worksegment' if a linker output file in ELF format is necessary.

No. D6	**Erroneously Error e16 'Segment too long' is generated (I)**
	IAR Reference EW24343 *Details* When placing an empty segment (= size 0 bytes) in a placement range of 0 bytes using the notation START:+SIZE, erroneously error message e16 'Segment too long' is generated even though the segment actually fits: `Error[e16]: Segment xxx (size: 0 align: 0) is too long for segment definition. At least 0 more bytes needed. The problem occurred while processing the segment placement command` *Workaround* Use a placement range greater than 0 bytes.

No. D7	**Erroneously Error e16 'Segment too long' is generated (II)**

IAR Reference EW10555

Details

When last segment of multi segment definition command is empty (= size 0 bytes) erroneously error message e16 'Segment too long' is generated even though the segment actually fits:

```
Error[e16]: Segment xxx (size: 0 align: 0) is too long for segment
definition. At least 0 more bytes needed. The problem occurred while
processing the segment placement command
```

In version V 6.0.3.49 or later an improved error message is generated:

```
Error[e189]: Unable to place the empty segment xxx (align 0). At the
moment of placement there were no available addresses where the segment
could be placed. Try changing the order the segments are placed in The
problem occurred while processing the segment placement command
"-Z(DATA)ONE,TWO,THREE=00000-0FFFF, where at the moment of placement
the available memory ranges were "-none-"
```

Example:

```
-Z(DATA)ONE,TWO,THREE=00000-0FFFF
```

Segment ONE uses the last available byte in the range; segment TWO and THREE are empty.

Workaround
Rearrange the -Z line so that the last listed segment is the one which size is greater than zero.

No. D8	**Range Error using far Runtime-Library Calls**

IAR Reference EW25288

Details

Several support routines for long operations can generate an out of range error at linktime when using far runtime library calls:

```
Error[e18]: Range error, Limit exceeded

  Where $ = ?L_XOR_L03 + 0x1F  [0x28049] in module "?LONG_XOR_L03"
            (C:\Program Files (x86)\IAR Systems\...\dlrl78fn2nf.r87),
           offset 0x1F in segment part 2, segment XCODE
  What: (?L_F_DEALLOC_L06 - (?L_XOR_L03 + 0x1E)) - 3 [0xFFFF7F95]
           Allowed range: 0xFFFF8000 - 0x7FFF
           Operand: ?0SL_F_DEALLOC_L06 [0x1ffe0]
           in module ?LONG_FLOAT_DEALLOC_L06
           (C:\Program Files  (x86)\IAR Systems\...\dlrl78fn2nf.r87),
           Offset 0x0 in segment part 2, segment XCODE
  Operand: ?L_XOR_L03 [0x28048]
           in module ?LONG_XOR_L03
          (C:\Program Files (x86)\IAR Systems\...\dlrl78fn2nf.r87),
           Offset 0x1e in segment part 2, segment XCODE
```

Workaround

Please check first if using far runtime library calls is necessary.
If yes, rename the code segment of runtime library `dlrl78fn2nf.r87` to any other name than XCODE and define the new segment in a specific 32KB area inside the far memory area.

No. D9	**Negative Value for N/A (alignment)**

IAR Reference EW25394

Details

When producing a module summary (-xe) the value of N/A (alignment) can become negative if the --segment_mirror option was used with the @-modifier on a segment with content. This is an incorrect sum in the map file and has no effect on the generated code. This problem has been present since the introduction of --segment_mirror in XLINK 5.4.0.28.

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

Module                     CODE   DATA  CONST
------                     ----   ----  -----
                          (Rel)  (Rel)  (Rel)
?CSTARTUP                    47
…
file                         58      6      6
N/A (alignment)                     -6
----------                  ---      -      -
Total:                      580      4      6
  + common                    2
```

Workaround
None. Fixed in XLINK version V6.3.0

No. D10	**Unused Addresses in Common Segments not filled correctly**

IAR Reference EW25592

Details

When generating more than two output files (e.g. one UBROF output file and additional output files in one of the simpler output format including, but not limited to, intel-hex and Motorola-s-records), XLINK fails to correctly generate filler bytes for COMMON segments.

Workaround
None.

No. D11	**Comand Line Segment Alignment ignored**

IAR Reference EW25374

Details

In case of using command line segment alignment -Z(SEGTYPE)SEGNAME|ALIGNMENT|, the specified alignment is ignored and the segment retains its natural alignment.

Workaround
Update to XLINK V6.2.268 or later

No. D12	**Symbol division results in a "division by zero" error**

IAR Reference EWRL78-714

Details

Using a division operator in the -D command results in an error:
-Ddiv=2
-Dresult=(4/div)

```
Error:
Tool Internal Error:
Internal Error: In function:
Diagnostic: Division by zero.

P0: 0 P1: 0
Internal Error: In function:
Diagnostic: Division by zero.
```

Workaround
This problem is solved in the released XLINK V6.5.4 which can be downloaded either from the Technical Note at the IAR web site or from the Renesas ToolWeb.

No. D13	**End address of checksum is wrong when using the -M option**

IAR Reference EWRL78-735

Details

The checksum end address is wrong when the XLINK -M option is used.

Workaround
None

No. D14	**Segment alignment fails by using the -Z option**

IAR Reference EWRL78-748

Details

When using the alignment specification (|alignment|) suffix for sequential segment placement (example: -Z(DATA)MYDATA|3| should 8-byte align the start address and size of the segment MYDATA), the size of the segment is not aligned if the segment has the SORT property. SORT is used on some data segments to sort them in alignment order (this minimizes size lost to alignment issues). Please refer to your Assembler Reference Guide for details on SORT.

Workaround
None

This problem is fixed in the XLINK version 6.6.2.104. Latest linker version can be retrieved from the IAR website here:
https://www.iar.com/support/tech-notes/linker/latest-version-of-xlink-linker/

No. D15	**End address of SADDR region is wrong**

IAR Reference -

Details

In all linker configuration file templates (*.xcl) 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:

```
-Z(DATA)SADDR_I,SADDR_Z,SADDR_N=FFE20-FFEF7
```

change to

```
-Z(DATA)SADDR_I,SADDR_Z,SADDR_N=FFE20-FFEDF
```

J) Description of Operating Precautions for Debugger C-SPY

No. E5	**All C-SPY Drivers: Structure not displayed Watch Windows**

Details

A struct variable of a typedef struct having the same name as the struct can not be displayed in the C-SPY watch window.

Example

```
typedef struct stTest{
  int i;
}tstTest;

volatile tstTest stTest;
```

Workarounds

1) Don't define a data type name:

```
typedef struct {
  int i;
}tstTest;
```

2) Use different names for data type and data object:

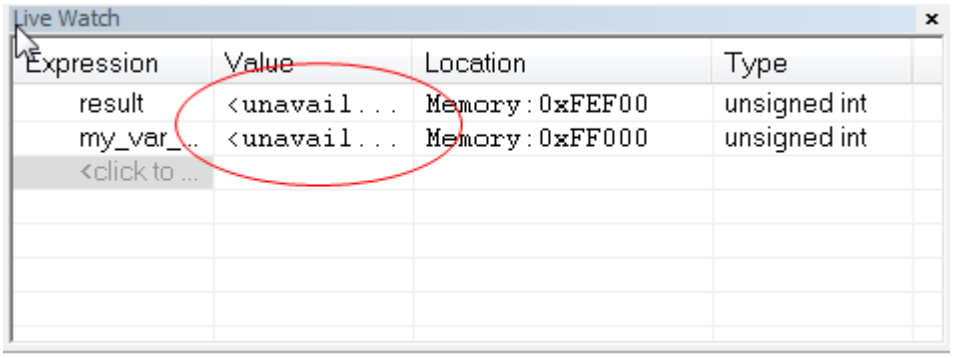
```
typedef struct stTest1{
  int i;
}tstTest;

volatile tstTest stTest;
```

No. E9	**E1 C-SPY Driver: No automatic Mapping for Variables added to Live Watch Window**

Details

Variables added to Live Watch Window are not updated automatically, because the automatic mapping doesn't work. Instead of the value <unavailable> is displayed.



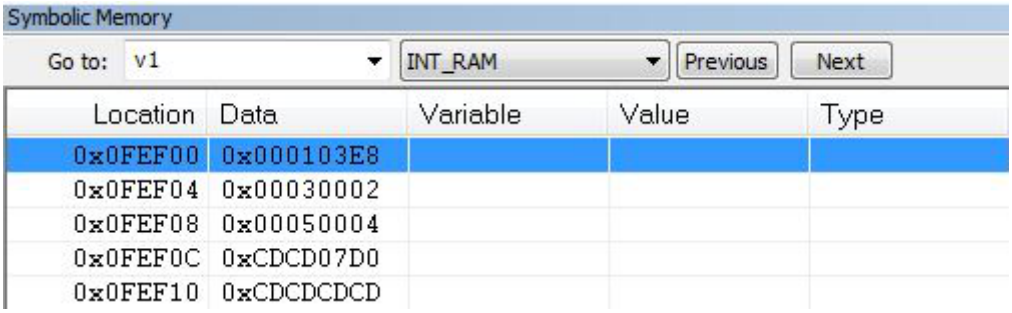
Workaround

As a temporary workaround until the next update patch is available please open the Live Memory Window and select the corresponding memory area for the variables listed in Live Watch Window.

No. E10	**All C-SPY Drivers: Symbols not listed in Symbolic Memory Window**

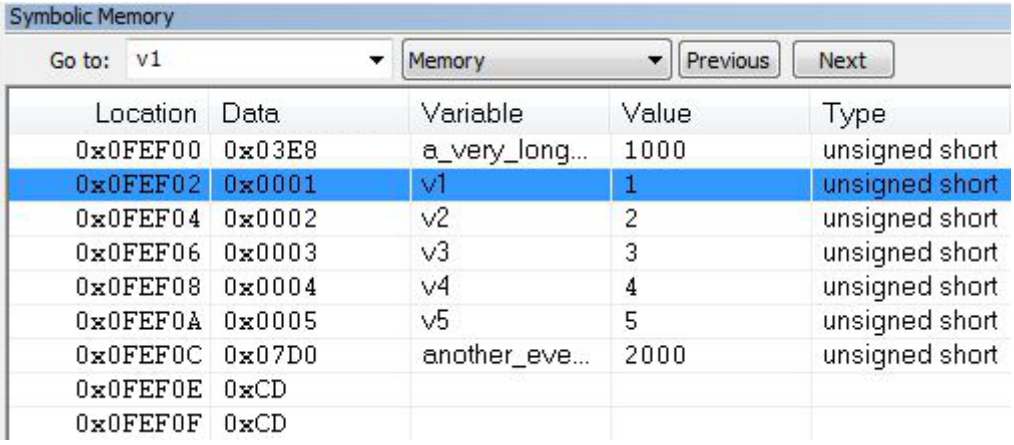
Details

Symbols are not listed in the Symbolic Memory Window, if another zone than 'Memory is selected. In this sample the zone 'INT_RAM is selected:



Workaround

Use only the memory zone 'Memory':



No. E12	**C-SPY IECUBE Driver: Pseudo Emulation of Temperature Sensor does not work**

Details

The Pseudo Emulation function to change the temperature sensor value does not work.
The default value for 25° is always used and can't be modified by the user.

Workaround
None.

No. E13	**C-SPY Simulator Driver: Display Problem in Timeline Window**
	Details Function entries and exits are not always shown correctly in the Timeline Window. It is only a display problem. Example: If function exit is missing and the function is called again later, a new row for the function is created. *Workaround* None. The problem will be fixed in next update V1.30.1 (schedule: e/o February 2013)

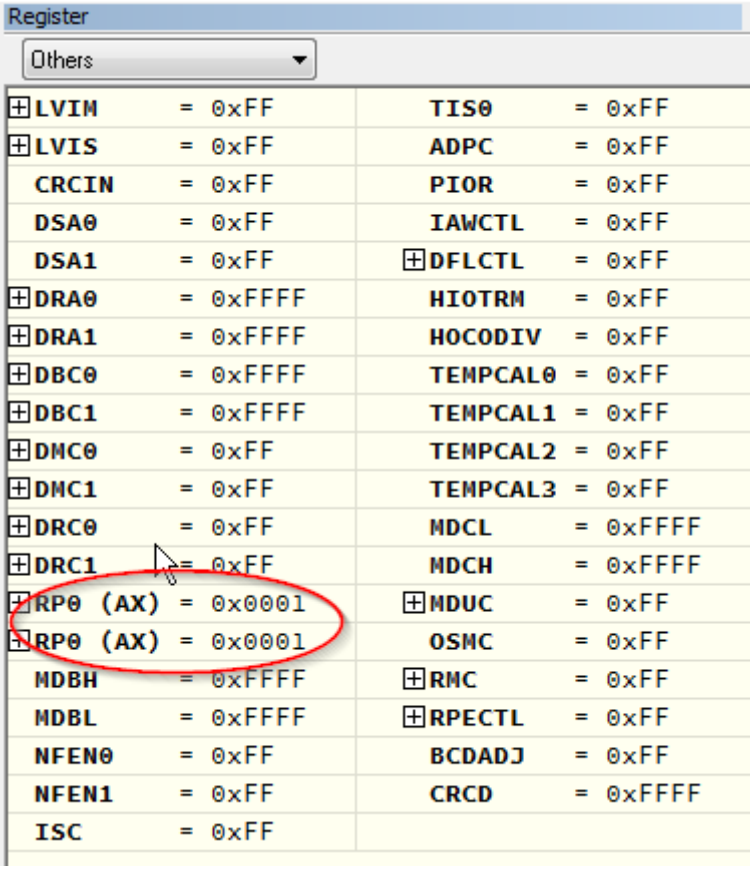
No. E14	**C-SPY E1 Driver: Wrong Manual I/O Register Modification**
	Details If an I/O register allowing byte and word access (e.g. register TDR01) is manually modified in Register Window a wrong value is written Example: Instead of 0x1234 a wrong value of 0x1200 is written to TDR1 Writing any value to low byte always clears the high byte. Writing any value to high byte always clears the low byte. *Workaround* None.

No. E16	**All C-SPY Drivers: Registers MDAL and MDAH not displayed in Register Window**

IAR Reference EW24033

Details

Due to a missing definition in the SFR file (-> subfolder rl78\config\debugger\ior5fxxxxx.sfr) the registers MDAL and MDAH of all RL78 Core0 (= RL78 Core S1) devices are not displayed in the Register Window. Instead of the listed registers the CPU register AX is displayed twice:



Workaround

Add the missing entries for registers MDAL and MDAH manually
Example for ior5f100le.sfr:

```
sfr = "MDAL",           "Sfr", 0xFFFF0, 2, base=16                          ;; Others
sfr = "MULA",           "Sfr", 0xFFFF0, 2, base=16                          ;; Others

sfr = "MDAH",           "Sfr", 0xFFFF2, 2, base=16                          ;; Others
sfr = "MULB",           "Sfr", 0xFFFF2, 2, base=16                          ;; Others
```

If you need further details, please contact the Renesas SW-Tool-Support-Team.

No. E17	**C-SPY E1 Driver: Unknown Break Error**
	IAR Reference EW24022 *Details* If you are using the E1 emulator and single-step over a for-loop or step into a function, the error "Break reason: Unknown (hwbrfact: 0x00000000)" is sometimes generated in Debug Log Window: **Mon Jul 08, 2013 11:04:36: Break reason: Unknown (hwbrfact: 0x00000000).** *Workaround* None.

No. E18	**C-SPY E1 Driver: Application doesn't start after Debug Session**
	IAR Reference EW23929 *Details* After a successfully closed E1 debug session, an application on RL78/F13 and RL78/F14 series doesn't start after power up of the target hardware without connected E1 emulator. *Workaround* None. The problem will be fixed in next EWRL78 SP V1.30.5

No. E19	**C-SPY E1 Driver: Crash at Reaching a Software Breakpoint**
	IAR Reference EW23929 *Details* The C-SPY debug session crashes if a software breakpoint reached and if the Flash-Selfprogramming-Library feature FSL_ChangeInterruptTable/FSL_RestoreInterruptTable is used. The problem only occurs on RL78/F13 and RL78/F14 series and for software breakpoints defined on code lines between function call of FSL_ChangeInterruptTable () and FSL_RestoreInterruptTable (). *Workaround* None. The problem will be fixed in next EWRL78 SP V1.30.5

No. E20	**All C-SPY Drivers: Debug Session did not Start**
	IAR Reference EW24226 *Details* In rare cases after trying to start a debug session of modified application via "Download and Debug", a build will be performed, but the debug session won't start. *Workaround* Please clean the project and start the debug session again.

No. E21	**IECUBE and E1 C-SPY Drivers: Data Flash Memory Window cannot be opened**
	IAR Reference EW24236 *Details* The Data Flash Memory Window cannot be opened by clicking the correspondent menu entry. *Workaround* None. Will be fixed in next update.

No. E22	**IECUBE and E1 C-SPY Drivers: Data Flash Memory Content cannot be changed in Memory Window**
	IAR Reference EW24237 *Details* Although the correct memory zone "EEPROM" is selected, the content of the Data Flash Memory is neither displayed correctly nor can be modified in C-SPY Memory Window. *Workaround* None. Will be fixed in next update.

No. E23	**E1 C-SPY Driver: IDE hangs due to Missing Frames in Trace Buffer**
	IAR Reference EW24263 *Details* Due to a problem in the algorithm for filling in missing frames between the branches in the trace buffer, the IDE may hang for a certain code example when using OCD trace. *Workaround* Uncheck the feature 'Fill in missing frames' in trace setup dialogue. Will be fixed in next update.

No. E24	**IECUBE C-SPY Driver: Wrong Time Stamp Information**
	IAR Reference EW24319 *Details* Due to usage of a wrong trace time base, for certain trace settings wrong time stamps are displayed in Trace Window. *Workaround* Please use 'Run Break Timer' or an event controlled timer for execution time measurement. Will be fixed in next update.

No. E25	**E1 C-SPY Driver: Data Sample Graph is not updated**
	IAR Reference EW24594 *Details* Variables displayed in the Sampled Graphs Window are not updated unless they are present in the Live Watch Window at the same time. *Workaround* Add variable to Live Watch Windows. Will be fixed in next update.

No. E26	**E1 C-SPY Driver: Debug Session closed after Error 'Flash macro service ROM accessed or stepped in'**
	IAR Reference EW24790 *Details* The debug session is closed after error 'Flash macro service ROM accessed or stepped in' occurs. The error occurs, if a single step action (step in, step over, step out) shall be executed 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. *Workaround* None. Will be fixed in future update, so that the debug session is continued.

No. E27	**E1 C-SPY Driver: RL78 device feature "RAM guard" doesn't work in case of single step execution on assembler instruction level**

IAR Reference ---

Details

The RAM area protected by the RL78 RAM guard feature can be unexpected re-written in case of single step execution on the assembler instruction level.

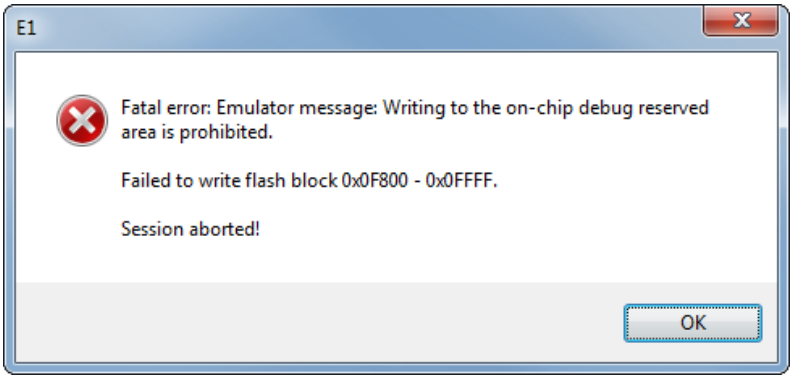
Workaround
Instead of using the single step on assembler level, please use
- single step execution on C level or
- RUN mode with/without breakpoints

No. E28	**E1 C-SPY Driver: Wrong Address area displayed in Error Message**

IAR Reference ---

Details

In case of using the E1 OCD emulator, the highest 256 bytes of the Flash memory must be reserved as described in the RL78 E1 Manual Addendum. In case of a violation of this requirement, a debug session cannot be started and an error message occurs instead.
In this error message instead of the highest 256 bytes the address range of the corresponding Flash memory block is displayed.



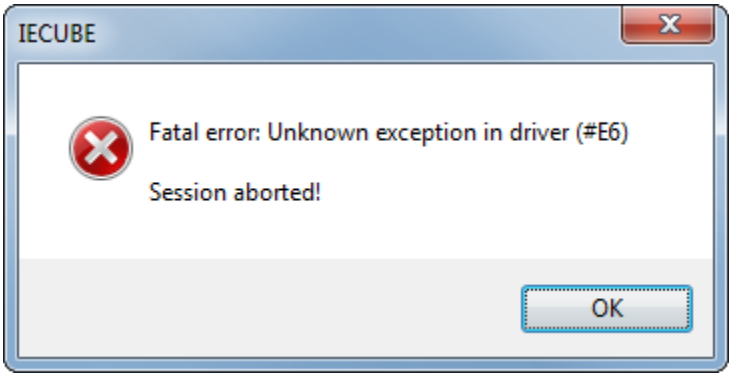
Workaround
Not necessary as the error message is correct.

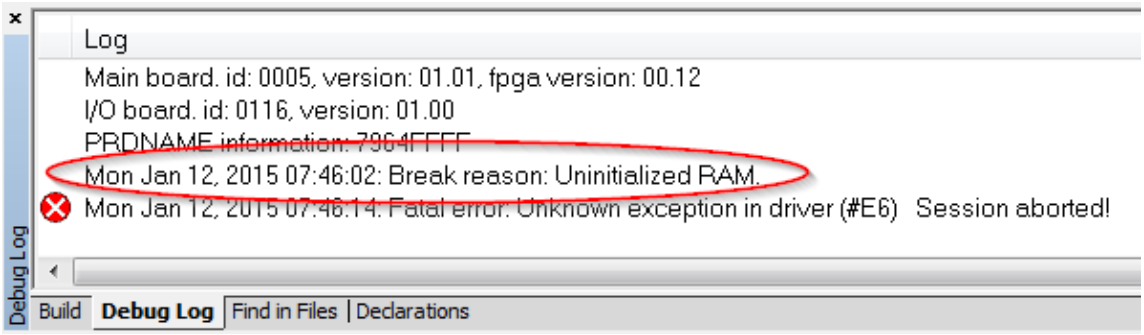
No. E29	**IECUBE C-SPY Driver: Debug Session closed after Fail-Safe-Break**

IAR Reference EW25151

Details

The Debug session is closed if an IECUBE fail-safe-break (e.g. read from uninitialized RAM) occurs. The fail safe reason is listed in Debug Log Window.





Workaround
None. The issue will be fixed in next update.

No. E30	**E1 C-SPY Driver: Debug Session closed after Error 'Flash macro service ROM accessed or stepped in' (II)**

IAR Reference EW25668

Details

A warning message is displayed when single step is not allowed during flashing and C-SPY stops execution with a "failed to run" message.

Same reason as described in issue E26. The correction of issue E26 did not handle the case where the breakpoint was placed on a jump instruction which means that 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. E31	**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. E32	**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. E33	**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)

K) Valid Specification

Item	Date published	Document No.	Document Title
1	April 2014	UIEEW-7	IAR Embedded Workbench IDE Project Management and Building Guide
2	March 2014	CRL78-3	IAR C/C++ Compilers Reference Guide for RL78
3	February 2011	ARL78-1	IAR Assemblers Reference Guide for RL78
4	January 2013	UCSRL78-3	IAR C-SPY Debugging Guide for RL78
5	March 2013	XLINK-600	IAR Linker and Library Tools Reference Guide
6	January 2011	EWMISRAC1998-4	IAR Embedded Workbench MISRA C 1998 Reference Guide
7	January 2011	EWMISRAC2004-3	IAR Embedded Workbench MISRA C 2004 Reference Guide

L) Revision

Edition	Date published	Document No.	Comment
1	26-04-2011	R20UT0521ED0000	First release.
2	27-06-2011	R20UT0521ED0001	EWRL78 update 1.10.2 Items C1 and E2 added
3	21-07-2011	R20UT0521ED0100	Items C2 and E3 added
4	08-08-2011	R20UT0521ED0101	EWRL78 update V1.10.3 Item B2 added
5	16-08-2011	R20UT0521ED0102	EWRL78 update V1.10.4 Items C3 and C4 added, item E3 updated Link to current document version changed.
6	13-09-2011	R20UT0521ED0103	Item B2 updated, items C5 and C6 added
7	13-10-2011	R20UT0521ED0104	Items C7, C8, E4, E5, and E6 added
8	28-10-2011	R20UT0521ED0105	EWRL78 update V1.10.5 Item E7 added
9	09-12-2011	R20UT0521ED0106	Items B3, B4 ,C9, D1 and C10 added
10	10-02-2012	R20UT0521ED0107	Items A3 and E8 added, specification update MISRA C 1998 and 2004 Reference Guide
11	27-02-2012	R20UT0521ED0108	Item C11 added
12	03-04-2012	R20UT0521ED0109	Item D2 added New Renesas Order Codes since 01.04.2012
13	16-04-2012	R20UT0521ED0110	EWRL78 update V1.20.1, specification update Embedded Workbench, C Compiler and Linker Reference Guide, item C6 updated, item C13 added
14	24-05-2012	R20UT0521ED0111	Items A4 and E9 added
15	18-06-2012	R20UT0521ED0112	Items C14, D3 and E10 added
16	01-08-2012	R20UT0521ED0113	Items C15 and C16 added
17	13-08-2012	R20UT0521ED0114	EWRL78 SP update V1.20.3 added Item E11 added
18	17-09-2012	R20UT0521ED0115	Item C17, C18 and E12 added
19	19-10-2012	R20UT0521ED0116	Item C19 added
20	31-10-2012	R20UT0521ED0117	EWRL78 Update V1.20.4
21	30-01-2013	R20UT0521ED0118	Item E13 added
22	26-02-2013	R20UT0521ED0119	Item B5, C20 and D4 added
23	12-03-2013	R20UT0521ED0120	EWRL78 update V1.30.2, specification update Embedded Workbench, C-Spy Debugger and Linker Reference Guide, item C21 added, items C1, E1 and E2 removed
24	03-04-2013	R20UT0521ED0121	XLINK update V5.6.0.36, item D1 removed, previous Renesas order codes removed

Edition	Date published	Document No.	Comment
25	15-05-2013	R20UT0521ED0122	Items A5, C22, C23 and E14 added
26	05-06-2013	R20UT0521ED0123	Items C24, C25 and E15 added
27	14-06-2013	R20UT0521ED0124	Update EWRL78 V1.30.3 Items C2,C3 and C4 removed
28	08-07-2013	R20UT0521ED0125	Items B6, E16 and E17 added
29	03-09-2013	R20UT0521ED0126	Item A6, C26 added
30	09-09-2013	R20UT0521ED0127	Item C27 added
31	26-09-2013	R20UT0521ED0128	Items C28, C29 added, item B6 updated
32	09-10-2013	R20UT0521ED0129	Item C30, C31, E18, and E19 added.
33	14-10-2013	R20UT0521ED0130	Update EWRL78 V1.30.5 Items C5 - C9, C11, C12, D2, and E3 removed Items C29 and C31 update
34	29-10-2013	R20UT0521ED0131	Items D5, E20, E21, and E22 added
35	20-11-2013	R20UT0521ED0132	Items E23 and E24 added
36	25-11-2013	R20UT0521ED0133	Item A7 added
37	06-12-2013	R20UT0521ED0134	Items C32 – C39 and D6 added Item D3 removed, XLINK Update V5.8.0.42
38	02-01-2014	R20UT0521ED0135	Items C40, C41 and C42 added
39	11-02-2014	R20UT0521ED0136	Items A8 and D7 added
40	10-04-2014	R20UT0521ED0137	Update EWRL78 V1.40.1 Items B7, C43, C44, C45 and E25 added Items C14, C15, C16, E4, E6 and E15 removed Items C34 and D7 updated
41	12-05-2014	R20UT0521ED0138	Item C46 added
42	21-05-2014	R20UT0521ED0139	Item C47 added
43	11-06-2014	R20UT0521ED0140	Item C48 added
44	25-06-2014	R20UT0521ED0141	Item C49 added
45	23-07-2014	R20UT0521ED0142	Items C50 and E26 added, Specification Update
46	07-08-2014	R20UT0521ED0143	Items A9, C51 and C52 added
47	22-09-2014	R20UT0521ED0144	Update EWRL78 V1.40.3 Item C53 added Items B2, B4, C17, C19, E7 and E9 removed Specification Update
48	17-10-2014	R20UT0521ED0145	Item A10, C54 and E27 added
49	29-10-2014	R20UT0521ED0146	Update EWRL78 V1.40.5 Item D4 removed, item C55 - C61 added
50	12-11-2014	R20UT0521ED0147	Update EWRL78 V1.40.6 Items C62, C63 and E28 added Status change item C57 (only partly solved in V1.40.5)
51	25-11-2014	R20UT0521ED0148	Item C64 added
52	12-01-2015	R20UT0521ED0149	Items A11 and E29 added Item E11 removed
53	17-02-2015	R20UT0521ED0150	Items A12, C65, and C66 added

Edition	Date published	Document No.	Comment
54	16-03-2015	R20UT0521ED0151	Items A13, C67, and D8 added
55	18-06-2015	R20UT0521ED0152	Item D9 added, item E28 updated, email address of Software-Tool-Support updated
56	10-08-2015	R20UT0521ED0153	Items C68 , D10 and D11 added, XLINK update 6.3.3.74
57	15-09-2015	R20UT0521ED0154	Item E30 added
58	01-02-2016	R20UT0521ED0155	Item C46 updated (fixed from version 1.40.3), item C56 updated (sample code updated), item E31 added
59	16-03-2016	R20UT0521ED0156	Item C69 added
60	27-04-2016	R20UT0521ED0157	Item C70 added
61	14.07.2016	R20UT0521ED0158	Item C71 added
62	23.11.2016	R20UT0521ED0159	Item E32 added
63	06.02.2017	R20UT0521ED0160	Item C72 added
64	13.03.2017	R20UT0521ED0161	Item A14 added
65	21.04.2017	R20UT0521ED0162	Item E33 added Item C73 added
66	09.06.2017	R20UT0521ED0163	Item C74 added Item C75 added Item C76 added Item C77 added Item C78 added Item C79 added Item C80 added Item C81 added Item C82 added Item C83 added Item C84 added Item C85 added
67	11.09.2017	R20UT0521ED0164	Item C86 added Item C87 added Item C88 added Item C89 added Item C90 added Item C91 added Item C92 added
68	21.03.2017	R20UT0521ED0165	Item C93 added
69	08.06.2018	R20UT0521ED0166	Item D12 added
70	13.08.2019	R20UT0521ED0167	Item C94 added Item D13 added Item D14 added Item D15 added
71	19.03.2021	R20UT0521ED0168	Item C95 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.

Please note that EWRL78 V1.xx had been updated to EWRL78 V4.xx already. Due to major internal differences between these versions, two different customer notifications are published.

Renesas Electronics Europe

R20UT0521ED0168
March 2021