

MULTI Compiler v2023.5 Release Notes for Embedded ARM



**Green Hills Software
30 West Sola Street
Santa Barbara, California 93101
USA
Tel: 805-965-6044
Fax: 805-965-6343
www.ghs.com**

LEGAL NOTICES AND DISCLAIMERS

GREEN HILLS SOFTWARE MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. Further, Green Hills Software reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Green Hills Software to notify any person of such revision or changes.

Copyright © 1983-2023 by Green Hills Software. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission from Green Hills Software.

Green Hills, the Green Hills logo, CodeBalance, GMART, GSTART, INTEGRITY, MULTI, and Slingshot are registered trademarks of Green Hills Software. AdaMULTI, Built with INTEGRITY, EventAnalyzer, G-Cover, GHnet, GHnetLite, Green Hills Probe, Integrate, ISIM, u-velOSity, PathAnalyzer, Quick Start, ResourceAnalyzer, Safety Critical Products, SuperTrace Probe, TimeMachine, TotalDeveloper, and DoubleCheck are trademarks of Green Hills Software.

All other company, product, or service names mentioned in this book may be trademarks or service marks of their respective owners.

For a listing of Green Hills Software's periodically updated patent marking information, please visit http://www.ghs.com/copyright_patent.html.

PubID: release_notes_arm-744926

Branch: <http://toolsvc/branches/release-branch-2023-5-comp>

Date: September 1, 2023

Contents

1. Changes in Version 2023.5	1
System Requirements	2
Windows	2
Linux	2
Product Compatibility	4
Feature Status Definitions	5
New Features and Enhancements	5
New Option to Set Default Buffer Size for stdio Libraries	5
Annex K Functions Available in All Language Dialects	5
Changes to Linker -C and -D Options	5
New Options to Treat Warnings as Errors	6
New Option to Control Whether dblink Errors Result in a Build Failure	6
Improved Debugging for C11, C++11, C++14, and C++17	6
New Option to Allow Flexible Array Members in All Dialects of C and C++	7
MIRROR Attribute Added to Memory Map Definitions	7
Newly Supported ARM CPUs	7
Changes in Behavior	7
Map Files Exclude Non-Allocated Sections	7
Libraries Reject Zero as Second Byte of Multibyte Character in EUC and Shift-JIS	8
Increased Instrumentation with -check=memory,nilderef	8
-O2 or -O3 imply -OV on --neon targets	8
Removed Features	9
Deprecated Features	9
Code Generation Defects	9

2. Changes in Version 2023.1 11

New Features and Enhancements	12
New Supported Host: Windows 11	12
New DoubleCheck Assert Fail Intrinsic	12
Improved DWARF Information with -delete	12
gaddr2line DWARF Support	12
New gbincmp Options	12
libc++ Now Provided as a Standard C++ Library Alternative	13
New Attribute set_offset	13
Changes in Behavior	13
New Version of ind_io_arrays.c	13
Change in Linkage for Const-Qualified Variable Templates (C++ CWG Defect 2387)	14
BUFSIZ increased to 8192 on INTEGRITY	15
Removed Features	15
Deprecated Features	15
ARM Cores	15
Code Generation Defects	16

3. Changes in Version 2022.1 17

New Features and Enhancements	18
New Option to Constrain Functions	18
New gstack Options	18
New Option for Floating-Point Divide Conversions	18
New Optimization Attribute and #pragma	18
New Pragma and Attribute Syntax for Confirming Section Placement of Variables	19
Default Optimization Levels	19
New CPU Support	19
New DoubleCheck Warning	20
New Option to Remap Paths in DWARF Information	20
Improved GNU Extended asm Code Generation	20
New Support for Splitting Relocatable Objects Across Memory Blocks	21
New POSIX Wide Character Conversion Functions	21

Changes in Behavior	21
Change to Defaults Shown for Optimization Options in Builder ...	21
Decreased Ability to Toggle Loop Optimizations	22
Optimization Strategies Selected for -O1, -O2, and -O3	22
Using -Olimit= Without Debugging Information No Longer Requires -glimits	22
Parallel Execution Policies No Longer Defined in std::execution	23
Options --fast=all and --small=all Now Apply to Only One Function	23
Change in Value for std::numeric_limits<floating-point-types>::has_signaling_NaN	23
Change in -MMD Option	24
Change in Parsing for Illegal UTF-8 Characters in Comments	24
The --thread_safe_initializations Option is Now Subsumed by -single_threaded	24
Assembler Rejects VFPv2, VFPv4, and VFPv5 Instructions with Earlier FPUs	24
DoubleCheck Warning #15	25
Behavior Change of unsigned Bitfields in Certain Situations	26
Removed Features	27
Deprecated Features	29
Code Generation Defects	29

4. Changes in Version 2021.1 **31**

New Features and Enhancements	32
New Option to Annotate Call/Return Instructions	32
New Option to Mitigate Against Forward-Edge CFI Attacks	32
New Option to Customize Arena malloc	32
New Option to Include Line Numbers in Temporary Preprocessor Output	32
New Support for Splitting Sections Across Memory Blocks	32
New Option for Improved Header Trace Diagnostics	33
New Predefined Macro for Secure Coding Standards	33
New Predefined Macro to Represent Toolchain Version	33
New Attribute to Cancel Generation of Non-Assembly Code	33

Changes in Behavior	34
Default align_val_t Overloads of operator new/delete Now Follow Proper Forwarding Order	34
Change in -globalreg Limitations	34
String Literals in Different Sections No Longer Merged Together	34
Options --section_prefix and --section_suffix No Longer Affect Special Sections	34
Deducing Types with_INITIALIZER List Constructors	35
Change in Defaults for Inlined C String Functions	35
Removed Features	35
Deprecated Features	36
Code Generation Defects	36

5. Changes in Version 2020.5 **37**

New Features and Enhancements	38
New DoubleCheck Features	38
New Linker Option to Prevent Function Deletion	38
New Documentation for Writing Secure Code	38
New Support for CERT C Rules	38
New Support for Binary Code Generation with DWARF Generation	39
New Option to Compile in Strict UAL Thumb Syntax	39
New UAL Thumb-1 Assembler Directive	39
New Option to Instrument Function Prologues and Epilogues	39
New Support for Half-Precision Floating-Point Type	40
New CPU Support	40
New gstack Option to Set All Stack Usages at Once	41
New Support for Subnormal Results in expf()	41
Changes in Behavior	41
Stricter Type Checking for Strong Function Pointers	41
Placement of String Literal Data in Object Files	42
gstack Returns Non-Zero Status if User Attention is Required	43
Change in gbuild Evaluation of :reference Path	43
Change in SMI and SMC Intrinsics	43
Deprecated Features	43

Code Generation Defects	44
6. Changes in Version 2020.1	45
Licensing	46
New Features and Enhancements	46
C11, C17, and C18 Support	46
C++17 Support	46
Improved Deferral of Function Template Parsing	46
Changes in Behavior	47
New Default Setting for Auto-Interrupt Table	47
Error for -fNaN_cmp_unordered with -fsoft or -fnone	47
ax D Modifier Change	47
Discretionary Errors Now Given in C++ Standard Library Headers	47
Linker Directives Files with Preprocessor Directives May Trigger Warnings	48
Removal of Non-Standard Members in Unordered Containers	48
Removed Features	48
Deprecated Features	48
Code Generation Defects	49
7. Changes in Version 2019.5	51
New Features and Enhancements	52
Support for ARMv6 Intrinsics Extended to ARMv6M	52
Linker-Defined Symbols Initialized to Thumb Functions	52
New CPU Support	52
New VFP v5 Support	53
New Linker Option for Expanded Section Layout Information	53
New Ability to Customize the Value of FOPEN_MAX	53
New Option to Align Standalone Objects	53
New gsrec Option for Selective Data Output	53
New Attribute for Referencing Absolute Variables in PID Programs	54
New #pragma Directive to Influence Loop Unroll Factor	54
New Option to Control Number of Callers Tracked with Run-Time Memory Checking	54

Improved Support for Arena and Real-Time malloc	54
Improved Support for Input Files in gasmlist	54
Changes in Behavior	55
Increased Number of Callers Tracked with Run-Time Memory	
Checking	55
Modified Returns for isnormal and isfinite	55
L'\x80' and L'\xff' No Longer Accepted as Wide Characters in	
Shift-JIS or EUC Modes	55
Removed Features	55
Deprecated Features	56
Code Generation Defects	56

8. Changes in Version 2019.1 **57**

New Features and Enhancements	58
Updated Compiler Front-End	58
New Option for Single-Threaded Performance	58
New Arena malloc	58
New Support for Synchronization and Hint Intrinsics on the	
Cortex-M23	58
New Support for v8 Instruction Set Intrinsics	59
New Support for v8M Intrinsics	59
Improved Assembler/Disassembler Support for the v8M Baseline	
Instruction Set	60
New Option to Conform to ARM Procedure Call Standard	60
Improved Wildcard Matching in Linker Directives Files	60
New Option to Disable ARM Cortex-M DSP Extensions	60
New Linker Option to Identify Separately Linked Absolute	
Images	61
New Linker Options to Provide Errors for Symbols Importable from	
Multiple Files	61
New Option to Assign Nameless Variables to Individual Data	
Sections	61
C++ Class Array Construction No Longer References	
new/malloc	61
New Attribute and Option to Enforce Section Order	62
Duplicate Strings Removed from DWARF Information	62
Improved Memory Functions for INTEGRITY Applications	62

Changes in Behavior	63
EDOM and ERANGE Not Defined in <math.h>	63
Changes to Alignment Reduction Behavior with __attribute__((aligned(x)))	63
Updated Aggregate Parameter Alignment Attributes	63
Assembler Now Enforces Range Limit of Thumb CBZ/CBNZ Instruction	63
When std::terminate() is Called, std::uncaught_exception() Now Returns "false"	64
Real-Time malloc is No Longer Experimental	64
New malloc Default	64
Scoped enum Without Explicit Base Type Always Given Base Type of int	64
Deprecated Features	64
Removed Features	65
Code Generation Defects	65

9. Changes in Version 2018.5 67

New Features and Enhancements	68
Support for MISRA 2012	68
New Option for Enhanced Loop Unrolling	68
Updated Support for C++11 and C++14	68
Improved gsrec Capabilities	68
New Option and Attribute to Perform Address Space Layout Randomization	69
Enhanced Line Information in Conditional and Logical Operators	69
Expanded asarm Support for Pseudoinstructions and Alternative Syntax	69
Support for ARMv8M Security Instructions	70
-no_data_in_text Supported on the Cortex-M23	70
New CPU Support	70
Changes in Behavior	70
Change in Accepted Multibyte Characters for Shift-JIS and EUC	70
Allocation of Uninitialized Global Variables Now Defaults to --no_commons	70

Change in servcode Utility Support	72
Removed Features	72
Code Generation Defects	72

10. Changes in Version 2018.1 73

New Features and Enhancements	74
C++14 Support	74
New Version of malloc	74
New Option to Control Vectorization of Floating-Point Calculations	74
New Predefined Symbols for Optimization Strategies	74
New Intrinsic to Mitigate Against Spectre Attacks	75
Improved Support for v6M Intrinsics	75
Improved Allocation of Data Into Program Sections	75
Changes in Behavior	76
INTEGRITY Small Block malloc Moved to libfmalloc	76
Updated __MULTI_HOST__ Macro	76
New -delete Behavior	76
New Default Individual Sections Behavior	76
Deprecated Features	76
Removed Features	77
Code Generation Defects	77

11. Changes in Version 2017.5 79

New Features and Enhancements	80
64-Bit Toolchain Availability	80
Updated Compiler Front-End	80
Improved Implementation of C rand() Function	80
Improved Preprocessor Options	80
New CPU Support	80
Improved C++11 Support	81
Increased Floating-Point Execution Speed	81
Improved Memory Function Speed	81
Improved #pragma ghs section	81
Enhanced Options for Retaining Temporary Preprocessor Files	81

Changes in Behavior	82
Updated alignof() Extension Behavior in C++11	82
Changes in Use of <code>__asm__</code> Keyword	82
Change in Linker Directive Files	82
Temporary Demo Incompatibility	83
Deprecated Features	83
Removed Features	83
Code Generation Defects	83
 12. Changes in Version 2017.1	 85
New Features and Enhancements	86
New CPU Support	86
New Option for ax Librarian	86
Improved INTEGRITY Support for <code>--large_vtbl_offsets</code>	86
Changes in Behavior	87
Modification of memmove Behavior	87
The Compiler Now Generates Template Information Files in Fewer Cases	87
Modification of Boolean Comparison Behavior	87
Change in Rebuilding Behavior of <code>gbuild</code>	87
New Option Name for <code>-display_error_number</code>	88
Removed Features	88
Code Generation Defects	88
 13. Changes in Version 2016.5	 89
New Features and Enhancements	90
C++11 Support	90
New Options Related to C++11	90
Support for Disabling Run-Time Type Information (RTTI) in C++11 Mode	91
Automatic NEON SIMD Vectorization	91
Data Alignment Support for 16 and 32 Bytes	91
Fast Simulator Improvements	91
Argument Type Checking	92
New Floating-Point Rigor Option	92
Option to Add Extra Dot to Individual Section Names	92

New Option to Set Lower Bound of p_align	92
Changes in Behavior	93
Default C Language Dialect is Now C99	93
Changes in fenv.h	93
Changes in abort() to Conform to C++ Standard	93
Changes in Standard C++ Behavior	93
C++ Binary Compatibility Across Compiler Versions	94
C++ Stream Locking	94
Treatment of Duplicate Inline Functions and Linkonce Template Instantiations	95
Stricter Diagnosis of Constructor Inaccessibility	95
Changes in Allowed Extensions	96
Modifying Libraries with ax Librarian	96
Changes in Symbol Names for Function Scope Local Static Variables	96
Thumb2 Assembler Encoding Change	97
Using -asm Now Implies -noobj	97
Deprecated Features	97
Removed Features	98
Code Generation Defects	98

14. Changes in Version 2015.1 99

New Features and Enhancements	100
Project Build Configurations	100
New CPU Support	100
Automated Setup of Multicore Targets	100
Improved Configurability of the Tools for INTEGRITY	100
Using Interprocedural Optimizations with Makefiles	101
Duplicate Symbol Detection in Linker and Archiver	101
Preserving Temporary Preprocessor Files	101
Support for Stripping Debug Information From Relocatable Objects and Libraries	102
Section Sorting in Linker	102
Generating Both Numerically and Alphabetically Sorted Symbols in Map File	102
Annotating ELF Files with Toolchain Version Information	102
Fast Simulator Improvements	102

Preventing Literal Data in Text	103
Link-time Global Variable Type Checking	103
Aligned Data Sections	103
Individual Function and Variable Sections	104
64-Bit Linker and dblink on Windows and Linux Hosts	104
Options to Modify Message Severity Supported for asm and elxr	104
Specify the Output Name of a Dependency File	104
Additional Documentation of Diagnostic Messages for Assemblers and elxr	104
Ability to Specify Diagnostics by Symbolic Tag Names	105
Support for %= Format String in GNU Extended Assembly	105
Changes in Behavior	105
Diagnostic Messages When Building INTEGRITY 5.x and 10.x	105
bcmp and bcopy now treat their size arguments as unsigned	106
Removed and Deprecated Features	106

15. Changes in Version 2014.1 107

New Features and Enhancements	108
New CPU Support	108
Cross-Project Implicit Dependency Analysis	108
New Undefined Behavior Checking	108
Range-Based For Loops	109
auto Type Keyword	109
Improved Inlining in C++	109
Ability to Check for Use of Floating Point	110
Intermodule Variable Allocation	110
Calls to pow(x, y) Now Optimized in Certain Cases	110
Generating a Target-Walkable Stack Now Supported in Thumb Mode	110
Ability to Control Generation of Hardware Divide Instructions ...	111
VFP v4 Now Supported	111
Inlining Constant Math Functions	111
Changes in Behavior	111
Changes in the Behavior of Some #pragma Directives	111
index() and rindex() Prototypes Now Hidden	112

Change in Default Behavior for Virtual Function Table Size	112
New C++ Restrictions Regarding <code>--large_vtbl_offsets</code>	112
Link-Once Template Instantiation Now Enabled by Default	113
Limited Support for C-Like Structure Packing in C++	113
New Default for Generation of Hardware Divide Instructions	114
<code>-gs</code> No Longer Implies <code>-gtws</code>	114
Instantiate Extern Inline Now Enabled by Default	114
<code>sqrt()</code> and <code>sqrtf()</code> Now Return NAN for Negative Input	115
Removed and Deprecated Features	115
16. Changes in Version 2013.5	117
Product Compatibility	118
New Features and Enhancements	118
Static Assertions	118
Specify a Compiler Inside a Project File	118
Ability to Specify the Size of <code>wchar_t</code>	119
Stack Smashing Protection	119
Memory Clearing Optimization Control	119
Updated <code>gstack</code> Utility Program for Better C++ Analysis	120
New Attribute and Intrinsic to Remove Specific Functions from Profiling Reports	120
Changes in Behavior	120
GNU Compatibility Updated for Better Compatibility with Version 4.3	120
17. Changes in Version 2013.1	123
Product Compatibility	124
New Features and Enhancements	124
New CPU Support	124
Relaxed Size Restrictions	124
<code>protrans</code> Supports Converting Section Dumps to <code>.pro</code> Files	125
Removed and Deprecated Features	125
18. Changes in Version 2012.5	127
Product Compatibility	128

New Features and Enhancements	128
Updated gstack Utility Program	128
New CPU Support	128
New Block Profiling System	129
New ax Modifier	129
New Linker Options to Control Linker p_align Behavior	129
-fsingle Supported on More Targets	129
New Stack Check Attribute	129
Changes in Behavior	130
Clean Builds Recommended	130
Error Compiling ind_initmem.c	130
Vector's Clear Method No Longer Deallocates	130
Implicit Inclusion is No Longer Enabled By Default	131
Eclipse Plug-in Option Type Changes	131
ghide No Longer Hides COMMON Symbols	131
Removed and Deprecated Features	131

19. Changes in Version 2012.1 133

Product Compatibility	134
New Features and Enhancements	134
Separate Releases Allow Greater Flexibility When Upgrading	134
Faster Builds	134
New CPU Support	135
New Flash Support	135
Define Macros in Any Project File	135
New Layout for Better Debugging When Running Out ROM	135
New Coding Standard Features	136
UTF-8 Support	136
ARM NEON Support	136
Unused Virtual Function Deletion Now Supported for ARM Targets	136
Changes in Behavior	137
New Licenses Required	137
Clean Builds Recommended	137
gbuild Defaults to Parallel Builds	137
Different malloc() Implementation Used By Default in Stand-Alone Projects	137

__MULTI_DIR__ Customization File Macro Changes	137
Pointers Are Now Unsigned by Default	138
memmove Moved to libind.a	138
Removed and Deprecated Features	138

20. Changes in Version 5.4 139

Product Compatibility	140
New Features and Enhancements	140
UTF-8 Support	140
Updated MULTI Eclipse Plug-In	140
Removed Features	140
Driver No Longer Accepts .bld Files	140

21. Changes in Version 5.2 141

Product Compatibility	142
New Features and Enhancements	142
New -Omoredebug Optimization Strategy	142
Support for Mixed Endianness	142
Updated Compiler Front-End	143
Updated Fast Flash Programmer	143
General Improvements	143
Changes in Behavior	143
New Warnings Enabled by Default	143
Changes to Line Continuations in GNU C Mode	144
Changes in std::string	144
Options Enabled by Default	144
Map File Format Changes	145
Changes to the Connection Editor for simarm	145
Loops with an Iteration Variable Shorter than int May Execute Slower than in Previous Releases	145
Mixed Endian Access in ARMv6	146
Big Endian VFP Doubles	146
Thumb-2 Relocation Number Changes	146
__alt_init Hook Removed From Stand-Alone Run-time Library ..	146
Removed Features	147

Deprecated Features and Discontinued Support	148
Deprecated K&R C Support	148
Deprecated -gnu Option	148
22. Changes in Version 5.0.6	149
New Features and Enhancements	150
Improved Code Generation	150
Improved Dependency Analysis	150
DoubleCheck Static Source Code Analyzer	150
New Fast Simulator Available	150
Improved Simulator Speeds	151
Fast Flash Programmer	151
Changes in Behavior	151
Changes to Driver Option Defaults	151
Options Enabled by Standard C++ Dialects	152
Libind.a Splitup	153
Building System Libraries from 4.x Distributions	153
Global Const Variables Placed in Read-Only Data	153
Debugging Code Optimized for Speed	153
Libraries for PIC Programs	154
Assembler Evaluation of Local Thumb Labels	154
GNU Assembler Support in MULTI 5	154
Support for XScale Extensions to the ARM Architecture	155
Allocation of Memory Pages at Run Time	155
Unbundled Features	155
Unbundled in 5.0.6	155
Deprecated Features and Discontinued Support	156
Deprecated Options	156
The ghexfile Utility is No Longer Available	157
Manual Template Instantiation Mode Options No Longer Supported	157
Programming flash from grun	157
Legacy Assembler Syntax Disabled by Default in MULTI 5	157
23. Known Issues	159
Code Generation Defects	160

Installation and Licensing	160
Name of Install Directory Cannot Contain Spaces	160
Compiler and Probe Missing From ginstall	160
Empty Parent Directory Deletion During Uninstall on Windows	
Hosts	160
Host Support	161
UNC Paths Are Not Supported in the MULTI Debugger	161
UNC Paths to Virtual Machines Are Not Supported	161
Toolchain Issues	161
INTEGRITY Fails to Build	161
NXP MAC71xx Processors and ARM/Thumb Mixed Code	161
gstack Provides Warnings for Green Hills Library Functions	162
gstack Provides Warnings for INTEGRITY	162
Use of -a, -p, or -pg in INTEGRITY KernelSpace Requires	
-Onomemfuncs	162
Cryptographic Extension Not Supported	162
Building Projects with Absolute Path Output Locations	162
Building a Project with Multiple Instances of a File	163
Error Associated with Calling sqrt	163
Toolchain Components Failing to Launch	163
Increased Compile Time and Host Memory Usage with Complex	
Templates	163
Toolchain File Size Limitation	164
Target Connection Issues	164
Flash Driver Library libflash.a Limited to Four CFI Drivers	164
Debugging Issues	164
Auto Trace Collection Reduces Performance	164
Target Agent May Cause Failures When Programming Flash	164
References Not Available for Inlined Functions	165
Cannot Debug Executables in Eclipse Projects with a Space in the	
Name	165
TimeMachine is Not Compatible with NEON Instructions	165
TimeMachine Does Not Track VFP Registers	165
Jazelle Mode Trace is Not Supported	166
Raw Trace is Not Displayed When Collecting Data Only Trace ...	166
Trace on Cogent CSB637/937	166
Tracing INTEGRITY Applications on AT91RM9200 Targets	166

C11 Debugging Issues	167
C++11 Debugging Issues	167
Memory Leak Detection May Result in False Positives/Negatives	167
Stack Trace Causes Incorrect Disassembly	167
File System Issues	168
Two Dots (..) Not Supported After Symbolic Links in Paths	168
NFS May Cause Poor Performance	168
Miscellaneous	168
Links to Connecting Book	168

Chapter 1

Changes in Version 2023.5

Contents

System Requirements	2
Product Compatibility	4
Feature Status Definitions	5
New Features and Enhancements	5
Changes in Behavior	7
Removed Features	9
Deprecated Features	9
Code Generation Defects	9

System Requirements

This section details the system requirements for the current compiler release.

The MULTI IDE may have different system requirements than the Compiler. For information about IDE requirements, see the release notes for the MULTI IDE.

Note that if you intend to build INTEGRITY, please contact Green Hills Support.

Windows

Running a full installation on Windows requires:

- A modern, multi-core, 64-bit x86 processor.
- One of the following:
 - Windows 8 (64-bit).
 - Windows 10 (64-bit).
 - Windows 11 (64-bit).
- 4 GB of RAM (16 GB is recommended).
- 10 GB of free disk space per installation (100 GB of free space and a solid state drive are recommended).
- A monitor running at a display resolution of 1024x768 or higher. (Two high-resolution monitors are recommended.)
- A DVD-ROM drive or access to the Green Hills Support site [<https://support.ghs.com>]. If neither is available, contact Green Hills Support.
- The ability to write to the Windows System directories. On most Windows systems, you must also be a member of the Administrators group. End users must have full access to installed files.

Linux

Running a full installation on Linux requires:

- A modern, multi-core, 64-bit x86 processor.

- One of the following:
 - Ubuntu 22.04.x LTS (64-bit).
 - Ubuntu 20.04.x LTS (64-bit).
 - Ubuntu 18.04.x LTS (64-bit).
 - Ubuntu 16.04.x LTS (64-bit).
 - Ubuntu 14.04.x LTS (64-bit).
- Note: Support for Ubuntu 14.04.x is deprecated and may be removed in a future release.

**Note**

Ubuntu is distributed with "HWE" kernels that are updated frequently and are not supported by Green Hills Software. If you encounter a toolchain issue while using an HWE kernel, we strongly advise trying a non-HWE kernel prior to contacting Green Hills Support.

- CentOS 7.x (64-bit).
 - CentOS 6.x (64-bit).
 - Note: Support for CentOS 6.x is deprecated and may be removed in a future release.
- 32-bit compatibility libraries are required. If you are using Ubuntu 14 or newer, run:

```
sudo dpkg --add-architecture i386
sudo apt-get update
sudo apt-get install libc6:i386 libncurses5:i386
sudo apt-get install libstdc++6:i386 libx11-6:i386 lib32z1
sudo apt-get install linux-libc-dev:i386
sudo apt-get install libxcursor1:i386
sudo apt-get install libc6-dev-i386
```

If you are using CentOS, run:

```
sudo yum install glibc.i686 glibc-devel.i686
sudo yum install libX11.i686 zlib.i686 libstdc++.i686
```

If you find that the tools are not working, especially because of a licensing issue, you may need to install additional 32-bit compatibility libraries. For

example, if you are using LDAP with SSS, you must install **libnss_sss**, which is usually part of the sssd-client package.

- The GNU C library (**glibc**) version 2.11 - 2.35.
 - Note: Support for **glibc** versions 2.11 - 2.16 is deprecated and may be removed in a future release.
- 4 GB of RAM (16 GB is recommended).
- 10 GB of free disk space per installation (100 GB of free space and a solid state drive are recommended).
- A monitor running at a resolution of 1024x768 or higher. (Two high-resolution monitors are recommended.)
- A graphical display running the X Window System.
- A mounted DVD-ROM drive or access to the Green Hills Support site [<https://support.ghs.com>]. If neither is available, contact Green Hills Support.
- An Ethernet device.
- Write permissions to the installation directory.

Product Compatibility

Green Hills Compiler 2023.5 is officially supported with:

- MULTI version 8.x.
- INTEGRITY version 11.0.4, 11.2.4, 11.4.4, 11.4.6, 11.7.8, 11.7.9, and 19.1.
- INTEGRITY IoT version 2017.0.
- u-velOSity version 2.6.4, 2.7.1, and 2.7.2.
- Green Hills Debug Probe software version 6.6.4.

For information about compatibility with newer versions of INTEGRITY, see the INTEGRITY release notes.



Note

If your MULTI IDE is newer than the compiler, see the IDE release notes for up-to-date product compatibility.

Feature Status Definitions

Toolchain features may have differing levels of support:

- **Supported** — Unless otherwise stated, toolchain features are considered supported by Green Hills.
- **Unsupported** — This feature exists in the toolchain but is not considered supported. It is offered as-is.
- **Deprecated** — This feature is currently supported but will be unsupported or removed in a future release.
- **Removed** — This feature is no longer available for the associated toolchain release and all subsequent releases unless otherwise stated.

New Features and Enhancements

New Option to Set Default Buffer Size for stdio Libraries

A new option (`-stdio_bufsiz=size`) allows you to set the default size of buffers used by the `stdio` libraries. For more information, see “Default buffer size for stdio libraries” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Annex K Functions Available in All Language Dialects

The bounds-checking functions described in C11 Annex K are now provided in all dialects of C and C++. (Previously they were only available in C11, C++11, and later dialects.)

Changes to Linker -C and -D Options

The `-C` and `-D` linker options now support expanded expressions. In addition to immediate values as before, these options will now accept arithmetic expressions consisting of immediates and/or linker constants.

New Options to Treat Warnings as Errors

A new option (**--quit_after_build_warnings**) allows you to treat warnings as errors in many toolchain components, similar to the existing driver option **--quit_after_warnings**. Individual options have also been added to control this functionality for each program separately. To clarify its scope, **--quit_after_warnings** has been renamed to **--quit_after_compiler_warnings**. For more information, see “Quit Building if Warnings are Generated” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5* and “Quit Building if Compiler Warnings are Generated” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Option to Control Whether dblink Errors Result in a Build Failure

A new option (**--fail_build_on_dblink_error**) allows you to specify that errors generated by **dblink** should result in a build failure. For more information, see “Fail Build if dblink Errors are Generated” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Improved Debugging for C11, C++11, C++14, and C++17

When used with Compiler 2023.5, the MULTI 8.x Debugger supports a number of new features, including:

- C++11 lambda functions
- C++14 variable templates
- Variables with thread-local storage in u-velOSity and INTEGRITY user space
- `char16_t`, `char32_t`, and `wchar_t` base types

The following options are not required and should not be used when debugging with MULTI 8.x:

- **--dbo_version=**
- **-dblink.args=-dloVersion=**
- **-dblink.args=-dnmVersion=**

Note: The debugger cannot debug `thread_local` variables in kernel space. If you attempt to display them in the Debugger, it will falsely report that they are optimized away.

New Option to Allow Flexible Array Members in All Dialects of C and C++

A new option (**--flexible_array_members**) allows you to use C99 conformant flexible array members in all dialects of C and C++. For more information, see “Allow Flexible Array Members in All Dialects” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

MIRROR Attribute Added to Memory Map Definitions

The `MIRROR` attribute has been added to memory map definitions, which can be used to represent overlapping regions of memory. For more information, see “Defining a Memory Map with the `MEMORY` Directive” in Chapter 8, “The `elxr` Linker” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Newly Supported ARM CPUs

The following CPUs are now supported:

- Cortex-M85 (**-cpu=cortexm85**)
- Cortex-X1C (**-cpu=cortexx1c**)
- Cortex-X3 (**-cpu=cortexx3**)
- Neoverse-V2 (**-cpu=neoversev2**)

Changes in Behavior

Map Files Exclude Non-Allocated Sections

By default when the linker emits a map file, it will now exclude unallocated sections (sections without the `SHF_ALLOC` flag set). Their addresses do not correspond to target memory and so they may appear as if they overlap allocated sections in

memory. To restore the old behavior of including these sections in the map file, pass the **-Mc** option to the driver.

Libraries Reject Zero as Second Byte of Multibyte Character in EUC and Shift-JIS

When the target character encoding is set to EUC or Shift-JIS (option **-kanji=euc**, **-kanji=shiftjis**, or **-kanji=euc/shiftjis**), the libraries no longer allow multibyte characters in which the second byte is zero. For example, if `mbtowc` is called on a string consisting of the bytes `0x81` and `0x00`, it will return `-1` (indicating invalid input); previously it would have returned `2` and produced the wide character `0x8100`. Likewise, the function `wctomb` rejects wide characters in which the low byte is `0` (except the null wide character `L'\0'`). Note that no legal non-null character in EUC or Shift-JIS ends with a zero byte; the libraries used for EUC and Shift-JIS are more permissive than either character encoding. This change is to better conform with the standard, which forbids a non-null multibyte character from containing a zero byte.

Increased Instrumentation with **-check=memory,nilderef**

When both **-check=memory** and **-check=nilderef** are enabled, the compiler now adds both types of instrumentation to the program. Previously, **-check=nilderef** was ignored in this case. This results in detecting more types of NULL dereferences than before, at the cost of increased executable size. Additionally, some NULL dereferences will now be detected by both types of instrumentation, resulting in two run-time errors being reported. The old behavior can be restored by disabling **-check=nilderef** wherever **-check=memory** is enabled, using either **-check=nonilderef** or **#pragma ghs check=nonilderef**. For more information, see “Run-Time Memory Checks” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

-O2 or **-O3** imply **-OV** on **--neon** targets

On ARM targets where **--neon** is implicitly enabled by default, **-O2** and **-O3** now enable **-OV**. In v2022.1 and v2023.1, **-OV** was only enabled by **-O2** or **-O3** if **--neon** was explicitly enabled by the user.

Removed Features

The following features are no longer available:

- The function `gamma ()`
- The **gproxy** utility

Deprecated Features

The following features are deprecated and may be removed in a future release:

- The option **--quit_after_warnings**. For the same behavior, use **--quit_after_compiler_warnings**.
- The option **-memory_access**.

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 160.

Chapter 2

Changes in Version 2023.1

Contents

New Features and Enhancements	12
Changes in Behavior	13
Removed Features	15
Deprecated Features	15
Code Generation Defects	16

New Features and Enhancements

New Supported Host: Windows 11

Windows 11 is now a supported host for the compiler.

New DoubleCheck Assert Fail Intrinsic

A new `__double_check_assert_fail()` intrinsic is now supported. For more information, see “Intrinsic Functions” in Chapter 14, “Macros, Pragma Directives, and Intrinsics” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Improved DWARF Information with `-delete`

Using the `-delete` option no longer causes the generated DWARF debugging information to be out of sync with the program in memory.

`gaddr2line` DWARF Support

The `gaddr2line` utility program can now use either the Green Hills `.dbo` information or DWARF information to correlate instruction addresses with source positions.

Two new options can be used to control this behavior: if `--dbo` is used, `gaddr2line` will only attempt to use the `.dbo` information. If `--dwarf` is used, it will only consider the program's DWARF information. If neither of these options is used, `gaddr2line` will only use the DWARF information if the `.dbo` information is missing.

See “The `gaddr2line` Utility Program” in Chapter 10, “Utility Programs” in *MULTI: Building Applications for Embedded ARM v2023.5* for more information.

New `gbincmp` Options

The `gbincmp` utility now supports three new options: `-[no]empty`, `-[no]empty_not_found`, and `-[no]prefix`. For more information, see “The `gbincmp` Utility Program” in Chapter 10, “Utility Programs” in *MULTI: Building Applications for Embedded ARM v2023.5*.

libc++ Now Provided as a Standard C++ Library Alternative

The Green Hills Compiler now offers the choice between two different implementations of the standard C++ libraries.

In addition to the existing Dinkumware standard C++ libraries, a Green Hills specific version of the libc++ libraries is now also provided and may be selected with the **--libcxx** option. Both libraries are intended to conform to the C++ standard and the choice between the two is primarily personal preference, although the choice must be consistent throughout your application.

There are some restrictions to the usage of the libc++ libraries, the most notable of which is that they may only be used with C++11 or higher. For more information, see “Specifying C++ Libraries” in Chapter 12, “Green Hills C++” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Attribute `set_offset`

The attribute `set_offset` can now be added to struct members, allowing a custom offset to be specified without the need for padding. This may be particularly useful when reading only a few fields in a file header or network packet, or when dealing with sparse device registers. Note that this attribute cannot reorder fields or cause them to overlap.

Changes in Behavior

New Version of `ind_io_arrays.c`

The file **`ind_io_arrays.c`**, used to customize the value of `FOPEN_MAX`, has been updated. If you have an existing project that includes a copy of that file, you will need to replace it with the new version. Failure to do so will result in an error similar to the following:

```
[elxr] (error #412) unresolved symbols: 1
      __ghs_alternate_file_arrays_v2      from myprogram.o
```

For more information, see “Customizing the Value of FOPEN_MAX” in Chapter 15, “Libraries and Header Files” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Change in Linkage for Const-Qualified Variable Templates (C++ CWG Defect 2387)

The compiler now implements the resolution to C++ Core Working Group issue 2387, which changes the default linkage of variable templates that are const-qualified.

Previously, an instantiation of a variable template would have internal linkage if the variable had a non-volatile const-qualified type and did not have an explicit 'extern' keyword on the declaration. It did not matter whether the 'const' keyword was specified explicitly in the declaration or whether the 'const' type was specified via a template argument.

Now, however, a variable template will never implicitly have internal linkage because of a const-qualified type (although it may have internal linkage for other reasons, such as being declared in an unnamed namespace).

```
template <typename T>
T const templ_var1 = 0;

template <typename T>
T templ_var2 = 0;

/* As part of the example, take the addresses
   of the variable templates to force their
   implicit instantiation. */

/* templ_var1<int> used to have internal (i.e.
   file) linkage, now has external linkage */
int const *ptr1 = &templ_var1<int>;

/* templ_var2<const int> used to have internal
   linkage, now has external linkage */
int const *ptr2 = &templ_var2<const int>;

/* templ_var2<int> used to have external linkage
```

```
(no change). */  
int const *ptr3 = &templ_var2<int>;
```

BUFSIZ increased to 8192 on INTEGRITY

The default buffer size used by the **stdio** library, `BUFSIZ`, has been increased to 8192 on INTEGRITY (from its old value of 512). This is to improve performance when doing large reads or writes, by reducing the number of input/output system calls required. Note that the **stdio** library allocates its buffers on the heap, so this will result in increased heap usage. Additionally, Tasks that allocate buffers of size `BUFSIZ` on the stack may need to have their `StackLength` attribute increased. Note also that the function `setbuf` sets a stream buffer which is implicitly of size `BUFSIZ`. Calls to that function which pass a buffer of hard-coded size may need to be changed to use `setvbuf`, which allows specifying the buffer size explicitly.

Removed Features

The following features are no longer available:

- INTEGRITY 5.x and 10.0.2 are no longer supported.

Deprecated Features

ARM Cores

The following cores are deprecated and may be removed in a future release:

- ARM7TM **arm7tm** `__ARM7tm__` Thumb-1, ARMv4
- ARM8 **arm8** `__ARM8__` ARM only, ARMv4
- ARM9 **arm9** `__ARM9__` Thumb-1, ARMv4
- ARM9E **arm9e** `__ARM9E__` Thumb-1, ARMv5
- ARM10 **arm10** `__ARM10__` Thumb-1, ARMv5
- StrongARM **strongarm** `__StrongARM__` ARM only, ARMv4
- XScale **xscale** `__XScale__` Thumb-1, ARMv5

- ARM11 **arm11** __ARM11__ Thumb-1, ARMv6
- ARM11T2 **arm11t2** __ARM11T2__ Thumb-2, ARMv6T2
- PJ4 **pj4** __PJ4__ Thumb-2, ARMv7-A
- PJ4v6 **pj4v6** __PJ4v6__ Thumb-2, ARMv6T2

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 160.

Chapter 3

Changes in Version 2022.1

Contents

New Features and Enhancements	18
Changes in Behavior	21
Removed Features	27
Deprecated Features	29
Code Generation Defects	29

New Features and Enhancements

New Option to Constrain Functions

A new option (**-constraint_rules_file**) allows function use to be restricted, such as for secure coding purposes. For more information, see “Constraint Rules” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New gstack Options

The **gstack** utility now supports three new options, all of which alter the information provided in **gstack** reports: **-list_funcs**, **--link_names_only**, and **--show_disamb**. For more information, see “gstack Options” in Chapter 10, “Utility Programs” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Option for Floating-Point Divide Conversions

A new option (**-const_fdiv_to_fmul**) allows the compiler to convert a floating-point divide by constant into an equivalent multiply, and is implied by **-frigor=fast**. For more information, see “Divide By Constant Becomes Multiply By Reciprocal” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Optimization Attribute and #pragma

A new attribute (`__attribute__((ghs_optimize(Ostrategy)))`) and `#pragma (#pragma ghs Ostrategy)` allow configuring the high-level optimization strategy for individual functions. These should be used instead of the old `#pragma ghs OL`, `#pragma ghs OS`, and `#pragma ghs ZO` (which are now deprecated). The new attribute and `#pragma` should more closely match the corresponding command line options. For more information, see “Green Hills Extension Pragma Directives” in Chapter 14, “Macros, Pragma Directives, and Intrinsics” in *MULTI: Building Applications for Embedded ARM v2023.5* and “Attributes” in Chapter 11, “Green Hills C” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Pragma and Attribute Syntax for Confirming Section Placement of Variables

The `#pragma ghs section` directive and `__attribute__((section(...)))` notations now accept an additional syntax that causes the compiler to issue an error if an affected variable would be placed into a different type of section than the type specified by the `#pragma` directive or attribute.

This behavior can be enabled using the `require` and `endrequire` argument in `#pragma ghs section`, or using the two-argument form of `__attribute__((section(...)))`. For more information, see “Green Hills Extension Pragma Directives” in Chapter 14, “Macros, Pragma Directives, and Ininsics” in *MULTI: Building Applications for Embedded ARM v2023.5* and “Attributes” in Chapter 11, “Green Hills C” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Default Optimization Levels

The options **-O1**, **-O2**, **-O3**, **-Os**, **-Oz**, **-Og**, and **-Od** have been added to provide the ability to select a default program characteristic to optimize for. For more information, see “Default Optimization Level” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New CPU Support

The following CPUs are now supported:

- Cortex-A78AE (**-cpu=cortexa78ae**)
- Cortex-A78C (**-cpu=cortexa78c**)
- Cortex-A710 (**-cpu=cortexa710**)
- Cortex-M55 (**-cpu=cortexm55**)
- Kryo560-Gold (**-cpu=kryo560g**)
- Kryo560-Silver (**-cpu=kryo560s**)
- Kryo570-Gold (**-cpu=kryo570g**)
- Kryo570-Silver (**-cpu=kryo570s**)
- Kryo670-Gold (**-cpu=kryo670g**)

- Kryo670-Silver (**-cpu=kryo670s**)
- Kryo670-Prime (**-cpu=kryo670p**)
- Kryo680-Gold (**-cpu=kryo680g**)
- Kryo680-Silver (**-cpu=kryo680s**)
- Kryo680-Prime (**-cpu=kryo680p**)
- Neoverse-N1 (**-cpu=neoversen1**)
- Neoverse-N2 (**-cpu=neoversen2**)
- Neoverse-V1 (**-cpu=neoversev1**)

New DoubleCheck Warning

DoubleCheck warning #22 is now available to detect dead writes to function parameters. For more information, see “Source Analysis Error and Warning Messages” in Chapter 5, “The DoubleCheck Source Analysis Tool” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Option to Remap Paths in DWARF Information

A new option (**-remap_dwarf_path**) allow the tools to alter paths in the generated DWARF debugging information, which makes it easier to generate identical ELF files in separate builds. For more information, see “Remap Path in DWARF Information” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Improved GNU Extended asm Code Generation

When using inline assembly with the GNU extended `asm` syntax, register parameters and return values will be allocated more efficiently to eliminate unnecessary move instructions.

New Support for Splitting Relocatable Objects Across Memory Blocks

When a section is designated in the link map to be split across multiple memory blocks, it may now split up different instances of that section coming from the same relocatable object.

New POSIX Wide Character Conversion Functions

The library functions `mbsnrtowcs` and `wcsnrtombs`, from POSIX, are now supported. These functions convert between wide and multibyte strings. For more information, see the POSIX standard and “Standard C Functions and Dependencies” in Chapter 15, “Libraries and Header Files” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Changes in Behavior

Change to Defaults Shown for Optimization Options in Builder

Most options in the **Advanced → Optimization Options** category now always show a default value of **Allowed** in the Builder, regardless of optimization strategy. Previously, the default was dependent on the optimization strategy. The new behavior is required because of how these options interact with the new optimization attribute and `#pragma` (see “New Optimization Attribute and `#pragma`” on page 18).

As an example, **Pipeline Instruction Scheduling** is performed under **-Ospeed**, but not **-Onone**. In previous releases, passing **-Onone** on the command line prevented pipelining for the entire file. In contrast, it is now possible for pipelining to be performed within such a file – for example, if it contains a function declared with `#pragma ghs Ospeed`. However, if **-Onopipeline** is also passed, then pipelining will be suppressed for the entire file, regardless of optimization attributes or `#pragma` directives. The old behavior could be implemented by having **-Onone** imply **-Onopipeline**, but the new behavior cannot, because **-Onone -Onopipeline** now has a different effect than **-Onone** by itself.

The names for these option settings have been changed to clarify their behavior: **Allowed** (instead of **On**) to indicate that the compiler may perform the optimization, and **Disabled** (instead of **Off**) to indicate that the compiler may not perform the

optimization anywhere in the file. The underlying driver options are the same, and any existing projects that pass these options will still pass the same options after upgrading. For more information, see “Optimization Options” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Decreased Ability to Toggle Loop Optimizations

It is no longer possible to toggle all loop optimizations as a unit. The compiler will choose appropriate loop optimizations to perform according to the optimization strategy. Individual loop optimizations which had separate options (such as loop unrolling) can still be manually disabled using those options, but cannot be manually enabled if not included in the optimization strategy. For example, it is no longer possible to manually enable loop unrolling under **-Ogeneral**. (Previously, this would have required passing **-OL** to enable loop optimizations.) To gain the speed benefits of loop optimizations without the size costs of applying them globally, we recommend selectively applying speed optimizations to those functions that make up an appreciable fraction of execution time. This can be done using the `ghs_optimize` attribute, `#pragma ghs Ospeed`, or the option **--fast=**. See “Optimize All Appropriate Loops” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*, “New Optimization Attribute and #pragma” on page 18 and “Optimize Specific Functions for Speed” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Optimization Strategies Selected for -O1, -O2, and -O3

The options **-O1**, **-O2**, and **-O3** have been updated to better suit desired performance characteristics. For more information, see “Default Optimization Level” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Using -Olimit= Without Debugging Information No Longer Requires -glimits

The options **-Olimit=peephole** and **-Olimit=pipeline** no longer require passing **-glimits** in order to have an effect when not generating debug information. The option **-glimits** is still recognized for backward compatibility, but it has no effect. For more information, see “-Olimit= Without Debug Information” in Chapter 3,

“Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Parallel Execution Policies No Longer Defined in std::execution

The `std::execution` namespace no longer defines the `parallel_policy` or `parallel_unsequenced_policy` types, nor the `par` and `par_unseq` variables. The goal of this change is to make it clearer that the parallel behaviors denoted by these entities are not supported. If your application requires the existence of these entities to build (but does not require parallel behavior), you can define them again by defining the macro `__GHS_KEEP_UNSUPPORTED_PARALLEL_ALGORITHM_DEFS` prior to the inclusion of `<execution>`.

Options `--fast=all` and `--small=all` Now Apply to Only One Function

When the string `all` is passed to the `--fast` or `--small` option, it now indicates to the compiler that the function named `all()` should be optimized in the indicated manner. Previously, it would result in every function being optimized in that manner. For more information, see “Optimize Specific Functions for Speed” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5* and “Optimize Specific Functions for Size” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Change in Value for

`std::numeric_limits<floating-point-types>::has_signaling_NaN`

The value of `std::numeric_limits<T>::has_signaling_NaN` is now `false` when `T` is a floating point type (such as `float` or `double`). Previously, this value was `true`. This prior value reflected the fact that signaling NaNs had their own representation and could be differentiated from quiet NaNs. However, with the exception of C++ functions dedicated to signaling NaNs, neither the C nor C++ libraries would specifically detect or create signaling NaNs. When a signaling NaN was passed as input to a library function it would be treated the same as a quiet NaN. Additionally, most hardware will silently ignore signaling NaNs or otherwise convert them to quiet NaNs when in default configurations. For these reasons, the value of `std::numeric_limits<T>::has_signaling_NaN` has been changed to `false` to more accurately reflect current behavior.

Change in -MMD Option

The option **-MMD** now omits the standard C headers in addition to the standard C++ headers and headers from system `#include` directories.

Change in Parsing for Illegal UTF-8 Characters in Comments

When the input source encoding is set to UTF-8 (e.g. **-kanji=utf8**), the compiler will now give an error if an illegal UTF-8 character is present in the comments of a source file. The error given is: `error #7: unrecognized token.`

The --thread_safe_initializations Option is Now Subsumed by -single_threaded

The **Thread Safe Initializations** option has been subsumed by the **Program is Single Threaded** option. The behavior previously enabled by **--no_thread_safe_initializations** is now enabled by **-single_threaded**, while **-no_single_threaded** corresponds to **--thread_safe_initializations**. This is true regardless of the C++ dialect selected.

As a consequence of this change, the **--thread_safe_initializations** behavior is now the default even in C++03 (assuming **-no_single_threaded** is in effect). As before, **--thread_safe_initializations** will not enforce thread safety unless the `_mtx_*` functions are defined. For more information, see “Program is Single-Threaded” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Assembler Rejects VFPv2, VFPv4, and VFPv5 Instructions with Earlier FPUs

In previous releases, the assembler would allow the following instructions to be used with a coprocessor version VFPv3 or higher:

```
vsel
vminnm
vmaxnm
vcvta
vcvtm
```

```
vcvtn  
vcvtp  
vrinta  
vrintm  
vrintn  
vrintp  
vrintx  
vrintz
```

These instructions are now restricted to coprocessors that allow for VFPv5 instructions. Note also the following:

- The instructions `vfma`, `vfms`, `vfma`, and `vfms` now require VFPv4.
- The `vmov` instructions to or from two general-purpose registers now require VFPv2.
- Targets that supported the v8.2 instruction set now also require a VFPv5 FPU to use the other VFPv5 instructions.

For more information, see “Floating-Point Coprocessor Version” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

DoubleCheck Warning #15

DoubleCheck warning #15 now detects dead writes to variables that are not function parameters, instead of all variables. Additionally, it is now off by default but may be re-enabled with `-double_check.enable=15`. This change is in tandem with the new DoubleCheck warning number #22 described in the New Features above. If your project previously used an ignore option or `#pragma` for #15, you may need to remove the ignore, replace #15 with #22, or ignore #22 as well, depending on your project's specifics.

Behavior Change of unsigned Bitfields in Certain Situations

The behavior of unsigned bitfields whose values are used in certain situations has changed. These situations only arise when:

- Both of the following are true:
 - The bitfield has an unsigned base type whose size is at least the size of an `int`.
 - The width is less than the number of bits in an `int`.
- And when the value of the bitfield is read in one of the following contexts:
 - The first operand of an assignment or compound assignment operator.
 - The operand of unary prefix or unary postfix increment or decrement.
 - The second operand of a comma operator.

Previously, the type of the value returned by the operator would have the unsigned base type of the field. Now, the type of the value will be a signed integer type (the result of integral promotion), which is consistent with how the bitfield's value is treated in ordinary read contexts. An exception is that the postfix increment and decrement cases in C++ continue to have the unsigned base type of the field. The new behavior is in accordance with the intent of the language standards.

This may cause a change of behavior in cases such as the following:

```
struct foo {
    unsigned int field: 4;
};
bool func(void)
{
    int negative = -1;
    struct foo bar;
    bar.field = 0;
    // Previously, the += operator's value would be taken to have
    // an unsigned type, so the usual arithmetic conversions would
    // convert negative to UINT_MAX, and the comparison
    // 0 <= UINT_MAX would return true.
    // Now, the += operator's value will be taken to be signed, and
    // the comparison 0 <= -1 will return false.
    return (bar.field += 0) <= negative;
```

```
// Note: If the code had been written:  
// bar.field += 0;  
// return bar.field < negative;  
// The code would have returned false both before and after  
// this change.  
}
```

Note that the behavior of assignment to a bitfield itself has not changed – what has changed is the observed type in these contexts (that is, the type of the assignment expression when it is used as part of a larger expression).

Furthermore, integral promotion on unsigned bitfields that have a width larger than the size of an `int` but less than the size of a `long long` has changed. Previously, these fields were given a type of `signed long long`. These types now have a type of `unsigned long long`. This behavior is also in accordance with the intent of the language standards.

Removed Features

The following features are no longer available:

- The `#pragma ghs OM`.
- The macro `__Char_Is_Signed__`.
- The macro `__Char_Is_Unsigned__`.
- The macro `__GHS_Inline_Memory_Functions`.
- The macro `__Int_Is_32`.
- The macro `__Int_Is_64`.
- The macro `__LL_BIT`.
- The macro `__LL_Is_64`.
- The macro `__Long_Is_32`.
- The macro `__Long_Is_64`.
- The macro `__MISRA_i`, where *i* is a MISRA 1998 rule.
- The macro `__ONLY_STANDARD_KEYWORDS_IN_C`.
- The macro `__Ptr_Is_32`.
- The macro `__Ptr_Is_64`.

- The macro `__Reg_Is_32`.
- The macro `__Reg_Is_64`.
- The macro `__WChar_Is_Int__`.
- The macro `__WChar_Is_Long__`.
- The macro `__WChar_Is_Short__`.
- The macro `__msw`.
- The option **--bool**.
- The option **--legacy** for the **gstack** utility.
- The option **--no_large_vtbl_offsets** when used with INTEGRITY versions 11 and later.
- The option **--saferc**. Note that this means that MISRA 1998 rules can no longer be enforced.
- The option **--unique_strings**.
- The option **-C (HTML File Compression)**.
- The option **-OM**.
- The option **-Oconstprop**.
- The option **-Omemfuncs**.
- The option **-dyload**.
- The option **-funroll-loops**.
- The option **-fvolatile**.
- The option **-h (Alternate Header Name)**.
- The option **-irel**.
- The option **-libbsp**.
- The option **-o (Alternate Source Name)**.
- The option **-p (Alternate Compression Table)**.
- The option **-pic_compat**.
- The option **-tekhex** for the **gsrec** utility.
- The preprocessing directive `#assert`.
- The preprocessing directive `#import`.
- The preprocessing directive `#include_next`.

- The use of conditional `#pragma` directives (such as `ifnooptimize`).
- The following library functions: `mktemp()`, `bcmp()`, `bcopy()`, `bzero()`, `fprintf()`, `getw()`, `truncate()`, `putw()`, `rand_r2()`, `setlinebuf()`.
- The following non-standard C++ headers: **`fstream.h`**, **`iomanip.h`**, **`iostream.h`**, **`new.h`**, **`rope`**, **`stdiostream.h`**, **`stl.h`**, **`strstream.h`**.
- Specifying a C++ mangled name for *foo* in `#pragma weak foo`.

Deprecated Features

The following features have been deprecated and will be removed in a future release:

- The `#pragma ghs OL`, `#pragma ghs OS`, and `#pragma ghs ZO`. Use the new `#pragma ghs Ospeed`, `#pragma ghs Osize`, and `#pragma ghs Onone` (respectively) instead. See “New Optimization Attribute and `#pragma`” on page 18.
- The function `gamma()`.
- The option **`-glimits`**.
- The option **`-OL`**.
- The option **`-OL=function`**.
- The option **`-Oloop`**.
- Assembly comments beginning with a number character (`#`).

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 160.

Chapter 4

Changes in Version 2021.1

Contents

New Features and Enhancements	32
Changes in Behavior	34
Removed Features	35
Deprecated Features	36
Code Generation Defects	36

New Features and Enhancements

New Option to Annotate Call/Return Instructions

A new option (**-annotate_call_return**) allows you to annotate call/return instructions for improved debugging. For more information, see “Annotate Call/Return Instructions” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Option to Mitigate Against Forward-Edge CFI Attacks

A new option (**-indirect_function_call_checks**) allows you to mitigate against forward-edge CFI attacks using indirect function-call checks. For more information, see “Indirect Function Call Checks” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Option to Customize Arena malloc

A new option, **-malloc_num_arenas**, allows for overriding the number of arenas used by the arena `malloc` on supported operating systems. For more information, see “Number of Arenas for Malloc” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Option to Include Line Numbers in Temporary Preprocessor Output

A new option (**-keepcppfileswithlines**) causes `#line` directives to be emitted in the preprocessed output files produced by **-keepcppfiles**. For more information, see “Line Numbers in Temporary Preprocessor Output” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Support for Splitting Sections Across Memory Blocks

The linker now supports the ability to split sections across multiple memory blocks using a section map. For more information, see “Splitting a Section Across Multiple Memory Blocks” in Chapter 8, “The elxr Linker” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Option for Improved Header Trace Diagnostics

A new option (`--diagnostic_header_trace`) allows diagnostic messages emitted from header files to be preceded by a trace of the directives to `#include` said header files. For more information, see “Show Header Trace for Diagnostics” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Predefined Macro for Secure Coding Standards

A new predefined macro (`GHS_STANDARDS`) is available to aid compliance with secure coding standards. For more information, see “Green Hills Secure Coding Options Files” in Appendix D, “Secure Coding Practices” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Predefined Macro to Represent Toolchain Version

A new predefined macro (`GHS_TOOLS_VERSION`) is available to represent the toolchain version in use. For more information, see “Predefined Macros” in Appendix F, “Customization Files” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Attribute to Cancel Generation of Non-Assembly Code

The new attribute `naked` is now supported, which can be used to tell the compiler to not generate a prologue, epilogue, or other non-asm code for a given function. This can be used to hand-write the body of a function in assembly, but define the function in a C module. For more information, see “Attributes” in Chapter 11, “Green Hills C” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Changes in Behavior

Default `align_val_t` Overloads of `operator new/delete` Now Follow Proper Forwarding Order

The default versions of `operator new/delete` (and their array versions) now follow proper forwarding when using an `align_val_t` parameter. For example, the default version of `operator new[](size_t, align_val_t)` will now forward to `operator new(size_t, align_val_t)`, as specified by the C++17 standard.

For backwards compatibility, the behavior of overloads without an `align_val_t` parameter is unchanged and still follows the C++03 specification (pre-LWG 206). This may change in future releases. Please keep this in mind if you replace these global operators with your own versions, as it may affect which overloads you must replace for correct behavior.

Change in `-globalreg` Limitations

The `-globalreg` option can no longer be set higher than 1 when building a C++ project. For more information, see “Global Registers” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

String Literals in Different Sections No Longer Merged Together

Previously, the compiler could merge together duplicate string literals even if they were allocated in different sections (for example, due to `#pragma ghs section` directives). Now the compiler will keep duplicate strings separate if they have any difference in their section mapping. This includes cases where the difference in section mapping has no effect on the section targeted for string allocation.

Options `--section_prefix` and `--section_suffix` No Longer Affect Special Sections

The names of special sections, such as those used to record profiling data with `-coverage`, are no longer affected by `--section_prefix` and `--section_suffix`. For more information about these options, see “Prepend String to Every Section Name”

in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5* and “Append String to Every Section Name” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*. options.

Deducing Types with Initializer List Constructors

When performing template argument deduction against a single-element initializer list, the compiler will now correctly deduce the expression as a copy constructor call rather than a call to a constructor taking an initializer list. This matches the behavior described by the C++ p0702r1 paper.

```
#include <initializer_list>

template<typename T> struct X
{
    X() {}
    X(std::initializer_list<T>);
    X(const X& other) = default;
};

X xi = {0};
X xxi = {xi};    // Previously deduced as type X<X<int>>, now X<int>
```

Change in Defaults for Inlined C String Functions

Previously, the option **Inlined C String Functions** defaulted to **Off (-Onostrfuncs)** when using the optimization strategies **-Ogeneral** and **-Ospeed**. This is no longer the case: these strategies now enable the option **-Ostrfuncs**. In addition, this change coincides with an improvement in the speed of the `strcmp()` function when inlined.

Removed Features

The following features are no longer available:

- The option **-bsp=s32v234_evb** (NXP S32V234-EVB).
- The option **-bsp=davincitms355** (TI DaVinci (DM355)).

Deprecated Features

The following features have been deprecated and will be removed in a future release:

- The option **-Onomemory**.
- The option **--section_prefix**.
- The option **--section_suffix**.
- The option **--thread_safe_intializations**.
- The **gstack** option **--legacy**.
- The **gsrec** option **-tekhex**.

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 160.

Chapter 5

Changes in Version 2020.5

Contents

New Features and Enhancements	38
Changes in Behavior	41
Deprecated Features	43
Code Generation Defects	44

New Features and Enhancements

New DoubleCheck Features

A new option, **-double_check.enable**, is available to undo the effect of any previously applied ignore option(s) as well as to enable some checkers that are off by default. In addition, the existing **-double_check.ignore** option now supports the argument `all`. For more information, see “DoubleCheck Errors to Enable” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5* and “Ignoring and Enabling Source Analysis Errors and Warnings” in Chapter 5, “The DoubleCheck Source Analysis Tool” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Linker Option to Prevent Function Deletion

When using **-delete**, a new linker option (**-keep_from**) prevents the deletion of all functions in a specified section. For more information, see “Linker-Specific Options” in Chapter 8, “The elxr Linker” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Documentation for Writing Secure Code

A new appendix has been added to the documentation that provides guidance on writing software for safety/security-critical systems. This includes a summary of compiler features and recommended coding practices to help prevent bugs and security vulnerabilities. For more information, see Appendix D, “Secure Coding Practices” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Support for CERT C Rules

The toolchain now supports the 2016 Edition of the SEI CERT C Coding Standard, a set of secure coding rules and recommendations for the C programming language. For more information, see “Supporting CERT C Rules” in Appendix D, “Secure Coding Practices” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Support for Binary Code Generation with DWARF Generation

The compiler now supports binary code generation when DWARF information is being generated. (Binary code generation can significantly decrease compile time.) Enabling **-dual_debug** with **-g** or **-G** will no longer disable binary code generation as it would in previous releases. For more information, see “Binary Code Generation” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5* and “Generate MULTI and DWARF Information” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Option to Compile in Strict UAL Thumb Syntax

A new option (**-strict_thumb1_ual_syntax**) allows you to generate assembly code that is strictly compliant with the ARM Unified Assembler Language (UAL). This is useful when compiling in Thumb mode on a processor that does not support Thumb-2. For more information, see “Disable Pre-UAL Thumb Syntax” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New UAL Thumb-1 Assembler Directive

A new assembler directive (`.thumb1`) allows the assembler to accept strictly UAL compliant Thumb-1 assembly. For more information, see `.thumb1` in “Miscellaneous Directives” in Chapter 7, “Assembler Directives” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Option to Instrument Function Prologues and Epilogues

A new option (**-instrument_prologue_epilogue**) now allows you to combine instrumentation and code snippets to assist with tasks such as logging functions or stack overflow detection. For more information, see “Instrument Prologue/Epilogue” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Support for Half-Precision Floating-Point Type

A new half-precision floating point type, `__fp16`, is now supported. The `__fp16` data type is capable of representing `NAN` and `INFINITY`, as well as finite and subnormal numbers. It can be used for reducing memory usage in cases where reduced range and precision are acceptable. For more information, see “Half-Precision Floating-Point Support” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New CPU Support

The following CPUs are now supported:

- Cortex-A77 (**-cpu=cortexa77**)
- Cortex-A78 (**-cpu=cortexa78**)
- Cortex-X1 (**-cpu=cortexx1**)
- Kryo465-Gold (**-cpu=kryo465g**)
- Kryo465-Silver (**-cpu=kryo465s**)
- Kryo468-Gold (**-cpu=kryo468g**)
- Kryo468-Silver (**-cpu=kryo468s**)
- Kryo470-Gold (**-cpu=kryo470g**)
- Kryo470-Silver (**-cpu=kryo470s**)
- Kryo475-Gold (**-cpu=kryo475g**)
- Kryo475-Silver (**-cpu=kryo475s**)
- Kryo485-Gold (**-cpu=kryo485g**)
- Kryo485-Silver (**-cpu=kryo485s**)
- Kryo490-Gold (**-cpu=kryo490g**)
- Kryo490-Silver (**-cpu=kryo490s**)
- Kryo495-Gold (**-cpu=kryo495g**)
- Kryo495-Silver (**-cpu=kryo495s**)
- Kryo585-Gold (**-cpu=kryo585g**)
- Kryo585-Silver (**-cpu=kryo585s**)

New *gstack* Option to Set All Stack Usages at Once

A new option for ***gstack*** (***-constant_stack_usage***) allows you to set the individual stack usage for every function to a single value. For more information, see “The *gstack* Utility Program” in Chapter 10, “Utility Programs” in *MULTI: Building Applications for Embedded ARM v2023.5* and “Shadow Stack” in Appendix D, “Secure Coding Practices” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Support for Subnormal Results in *expf()*

On configurations where subnormal values are supported and not flushed to zero, *expf()* now returns a subnormal value when the correct result is subnormal rather than returning 0.

Changes in Behavior

Stricter Type Checking for Strong Function Pointers

The compiler now performs stricter type checking for strong function pointers when taking the address of a function. Specifically, types with `__attribute__((strong_fptr))` are now taken into consideration when they occur as parameter or return types for a function type. This includes the function type for the function whose address is being taken and the function type of the object to which the address is assigned. If the strong function pointer type does not match the destination type, this will either result in an error (typically for non-relaxed *strong_fptrs*) or the compiler will apply a conservative analysis for that strong function pointer (for relaxed *strong_fptrs*).

```
typedef void (*FPTR) (void);
__attribute__((strong_fptr)) typedef void (*STRONG_FPTR) (void);
__attribute__((strong_fptr("relax"))) typedef void (*RELAX_STRONG_FPTR) (void);

void foo(STRONG_FPTR);
void bar(RELAX_STRONG_FPTR);

void example()
{
    void (*fptr1) (FPTR) = &foo; // Now rejected because parameter type mismatch

    void (*fptr2) (FPTR) = &bar; // Allowed because RELAX_STRONG_FPTR is relaxed,
```

```
                                // but causes analysis to be very conservative.  
}
```

Types with `__attribute__((strong_fptr("relax")))` are allowed to mismatch if the mismatch occurs within immediate parameter types or return types of the top-most function type. Parameter types or return types that are more deeply nested are not allowed to mismatch, even if they are relaxed strong function pointers.

By default, the compiler will enforce this stricter checking when the **-act_like** version is set to 2020.5 or greater. For other **-act_like** values, the compiler will allow many of these conversion mismatches for backwards compatibility, but will perform an extremely conservative **gstack** analysis. Note that if you are compiling code for INTEGRITY, **-act_like** may be automatically set to an appropriate version. You should be cautious about changing this value.

Placement of String Literal Data in Object Files

In prior versions, string literals inside function templates or inline functions were allocated in `.ghs.linkonce` sections that corresponded to their containing function. For example:

```
.ghs.linkonce.7..rodata.foo
```

These `.ghs.linkonce` sections were collapsed to their destination section (such as `.rodata`) at link time. Now the majority of these string literals will be allocated directly in their destination section instead. Additionally, the ordering of string literals in their respective object file section (for example, `.rodata`) may be substantially different than previous releases. This is true even for string literals in non-template and non-inline functions.

A side effect of these changes is that the compiler is now better at merging duplicate string literals from different functions, which may result in fewer string literals being allocated in the program.

Section remappings (for example, via the `#pragma ghs section` directive) are still honored for string literals. Note, however, that a duplicate string literal may not be allocated at all, and may refer to an identical string literal allocated elsewhere.

gstack Returns Non-Zero Status if User Attention is Required

The **gstack** utility will now return a non-zero exit status if it detects a problem that requires user attention. For example, **gstack** will return such a status if a program is missing the information needed to compute an accurate upper bound on the program's stack usage, or if the program computes an upper bound that exceeds the user-specified stack limit. Previously, **gstack** would return an exit status of zero in these cases.

Change in gbuild Evaluation of :reference Path

The **gbuild** utility now honors conditional control statements when evaluating the project path specified using the **:reference** option in a **Reference** project. Thus, if a path entry is conditionally disabled, the reference will not be resolved, and **gbuild** will output an error message such as `Error: Unable to resolve reference`. For example, the following error may manifest for INTEGRITY 11.7.8 when building **multivisor_mono_common_refs.gpj**:

```
Error: Unable to resolve reference:
servers.gpj;servers_ext.gpj;ghswitch_server.gpj;ghswitch_module.gpj
```

If this occurs, either upgrade to INTEGRITY 11.7.9 or contact Green Hills Support.

Change in SMI and SMC Intrinsics

The SMI and SMC ARM intrinsics now produce the SMC mnemonic instead of SMI. The argument must be representable with only 4 bits instead of 16.

Deprecated Features

The following features have been deprecated and will be removed in a future release:

- The functions `flockcreate()` and `flockdestroy()`.
- The compatibility mode simulator **simarm_compat**.

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 160.

Chapter 6

Changes in Version 2020.1

Contents

Licensing	46
New Features and Enhancements	46
Changes in Behavior	47
Removed Features	48
Deprecated Features	48
Code Generation Defects	49

Licensing

Beginning with Compiler 2020.1, licensing details such as local cache behavior and seat usage are specified in the licensing agreement itself. One or more instances of a new licensing binary **lic_lics** will run on hosts where the toolchain is used.

New Features and Enhancements

C11, C17, and C18 Support

The compiler now supports C11, C17, and C18. These language dialects include threads, atomics, bounds-checking library functions, aligned memory, and generics. Bear in mind that common data is not supported in C11 or later. This applies both to the option **--commons** and to the `common` attribute. For more information, see “ISO C11, ISO C17, and ISO C18” in Chapter 11, “Green Hills C” in *MULTI: Building Applications for Embedded ARM v2023.5*.

C++17 Support

The compiler now supports C++17. C++17 includes hexadecimal floating-point literals, `if constexpr` statements, and enhanced support for overaligned types with `new` expressions. As with all supported C++ variations, files built or linked in C++17 mode should not be mixed with files built or linked using other dialects. For more information about C++17, see “C++17” in Chapter 12, “Green Hills C++” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Improved Deferral of Function Template Parsing

The option **--defer_parse_function_templates** is now more effective at deferring the instantiation of inline template functions.

Changes in Behavior

New Default Setting for Auto-Interrupt Table

The **Auto-Interrupt Table** option now defaults to off (**-no_auto_interrupt_table**). If you are using a non-Cortex M processor along with C language interrupt routines, you may need to pass **-auto_interrupt_table** to maintain the prior behavior.

Error for **-fNaN_cmp_unordered** with **-fsoft** or **-fnone**

A fatal error is now given for **-fNaN_cmp_unordered** in conjunction with the options **-fnone** (no floating point) or **-fsoft** (software floating point). Note that **-fsoft** is enabled by default on some processors and may be set explicitly on most processors. Previously, **-fNaN_cmp_unordered** was ignored in these cases.

ax D Modifier Change

Using the **ax** librarian's **D** modifier now sets a consistent file mode independent of the host or file system permissions. Previously, the mode of a file in an archive would depend on said properties.

Discretionary Errors Now Given in C++ Standard Library Headers

Previously, the compiler suppressed discretionary error diagnostics that were caused by code in the C++ Standard Library headers. (That is, errors with a **-D** suffix on their diagnostic number.) Now, however, the compiler will emit diagnostic messages with discretionary error severity even if they arise from C++ Standard Library headers. This can cause code to be rejected with Compiler 2020.1 even though it compiled successfully with previous releases. This should only occur with programs that are invalid since most such diagnostic messages result from violating library feature requirements.

For example: Access errors are a common discretionary error (such as from accessing a private or protected member of a class). While it is invalid to use `std::make_unique` to construct an object via a private constructor, the compiler would have previously allowed it because the access error would have been suppressed. Now the access error diagnostic will be given and compilation will fail.

Linker Directives Files with Preprocessor Directives May Trigger Warnings

The linker will now output a warning when passed linker directives files containing preprocessor directives such as `#if` or `#include`. Previously, these directives were ignored because the `#` character also signifies the beginning of a comment.

Preprocessing of linker directives files is handled by the driver when **-preprocess_linker_directive** is in effect. In this circumstance, the compiler will be used as a preprocessor and the resulting linker directives files will not have such directives.

Removal of Non-Standard Members in Unordered Containers

The unordered containers (`std::unordered_[multi]set` and `std::unordered_[multi]map`) previously defined several non-standard members:

- The member types `reverse_iterator`, `key_compare`, and `value_compare`.
- The member functions `rbegin()`, `crbegin()`, `rend()`, `crend()`, `lower_bound()`, `upper_bound()`, `key_comp()`, and `value_comp()`.

These member types and functions are no longer defined by default. If your program makes use of them, you can define the

`__GHS_UNORDERED_CONTAINER_EXTENSIONS` macro to go back to the old behavior. (Note that these non-standard members, as well as the `__GHS_UNORDERED_CONTAINER_EXTENSIONS` macro, are deprecated.)

Removed Features

The following features are no longer available:

- Support for 32-bit hosts.

Deprecated Features

The following features have been deprecated and will be removed in a future release:

- The option **-Onoconstprop**.

- The functions `mktemp()` and `setlinebuf()`.

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 160.

Chapter 7

Changes in Version 2019.5

Contents

New Features and Enhancements	52
Changes in Behavior	55
Removed Features	55
Deprecated Features	56
Code Generation Defects	56

New Features and Enhancements

Support for ARMv6 Intrinsic Extended to ARMv6M

The following existing intrinsics are now enabled for Cortex-M0 and Cortex-M1: REVSH, REV, REV16, SEV, WFE, WFI, YIELD. For more information, see “v6 Instruction Set Intrinsics” in Chapter 14, “Macros, Pragma Directives, and Intrinsics” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Linker-Defined Symbols Initialized to Thumb Functions

Any absolute symbol that is both declared in a linker directives file and initialized to the value of a thumb function will now be marked as a thumb function. When its address is taken from within a program it will have its low bit set to distinguish it as pointing to thumb code.

New CPU Support

The following CPUs are now supported:

- Cortex-A76 (**-cpu=cortexa76**)
- Cortex-A76AE (**-cpu=cortexa76ae**)
- Kryo250-Gold (**-cpu=kryo250g**)
- Kryo250-Silver (**-cpu=kryo250s**)
- Kryo260-Gold (**-cpu=kryo260g**)
- Kryo260-Silver (**-cpu=kryo260s**)
- Kryo280-Gold (**-cpu=kryo280g**)
- Kryo280-Silver (**-cpu=kryo280s**)
- Kryo360-Gold (**-cpu=kryo360g**)
- Kryo360-Silver (**-cpu=kryo360s**)
- Kryo385-Gold (**-cpu=kryo385g**)
- Kryo385-Silver (**-cpu=kryo385s**)
- Kryo460-Gold (**-cpu=kryo460g**)

- Kryo460-Silver (**-cpu=kryo460s**)

New VFP v5 Support

Support for VFP v5 is now available with **-fpu=vfpv5** and **-fpu=vfpv5_d16**. This includes the following floating-point instructions: `VSEL`, `VMAXNM*`, `VMINNM*`, `VCVT*`, and `VRINT*`.

New Linker Option for Expanded Section Layout Information

A new option for map files (**-Ms**) will add information about the correlation between output sections and section inclusion rules in the linker directives file. For more information, see “Display Section Layout in Map File” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Ability to Customize the Value of FOPEN_MAX

It is now possible to customize the value of `FOPEN_MAX`. For more information, see “Customizing the Value of `FOPEN_MAX`” in Chapter 15, “Libraries and Header Files” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Option to Align Standalone Objects

A new option (**-align_standalone_objects**) allows for the direct control of optimizations that can overalign objects. For more information, see “Align Standalone Objects” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New gsrec Option for Selective Data Output

A new option for **gsrec** (**-sec s**) allows you to exclude sections other than *s* from data output. For more information, see “The gsrec Utility Program” in Chapter 10, “Utility Programs” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Attribute for Referencing Absolute Variables in PID Programs

A new attribute (`absolute_label`) allows for creating global variables in linker directives files that are not PID-adjusted. For more information, see “Attributes” in Chapter 11, “Green Hills C” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New #pragma Directive to Influence Loop Unroll Factor

The compiler now supports the `#pragma ghs unroll=n` directive, which can be used to influence the unroll factor the compiler uses when unrolling a loop. This can be useful if the compiler's default choice of unroll factor is suboptimal for a given loop. For more information, see “Green Hills Extension Pragma Directives” in Chapter 14, “Macros, Pragma Directives, and Intrinsic” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Option to Control Number of Callers Tracked with Run-Time Memory Checking

A new option (`-memcheck_num_callers=n`) allows you to specify how many callers should be tracked for each allocation when run-time memory checking is enabled. For more information, see “Number of Callers to Track with Run-Time Memory Checks” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Improved Support for Arena and Real-Time malloc

The **Arena** and **Real-Time** settings for `-malloc_version` are now supported when INTEGRITY TimeMachine instrumentation is retained at link time.

Improved Support for Input Files in gasmlist

The **gasmlist** utility now supports fully linked ELF files with DWARF information, as well as object files with DWARF or Green Hills **.dbo** information. For details, see “The gasmlist Utility Program” in Chapter 10, “Utility Programs” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Changes in Behavior

Increased Number of Callers Tracked with Run-Time Memory Checking

The memory-checking `malloc` library (`-check=alloc/-malloc_version=memcheck`) now tracks 8 callers for each allocation by default, up from the old value of 5. This will lead to increased heap usage when run-time memory checks are enabled. For information about configuring this value, see “Number of Callers to Track with Run-Time Memory Checks” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Modified Returns for `isnormal` and `isfinite`

The functions `isnormal()` and `isfinite()` now return a non-zero integer when their conditions are met, and 0 otherwise. Previously, they returned 1 or 0. The new implementation is more efficient and conforms to the ISO C99 standard. However, any programs that incorrectly depend on them to return exactly 1 will no longer behave as expected.

`L'\x80'` and `L'\xff'` No Longer Accepted as Wide Characters in Shift-JIS or EUC Modes

When target character encoding set to Shift-JIS or EUC (`-kanji=shiftjis`, `-kanji=euc`, or `-kanji=euc/shiftjis`) library functions that convert from a wide character or string to a multibyte character or string now reject the values `L'\x80'` and `L'\xff'`. These are not valid wide characters in EUC or Shift-JIS encoding. Previously, attempting to convert these characters resulted in multibyte strings that would be rejected by functions that convert in the other direction.

Removed Features

The following features are no longer available:

- Support for MULTI 6.x.

Deprecated Features

The following features have been deprecated and will be removed in a future release:

- The library functions `bcmp`, `bcopy`, and `bzero`. Use `memcmp`, `memmove`, and `memset` instead.
- The option **-Omemfuncs**.
- Support for 32-bit hosts.

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 160.

Chapter 8

Changes in Version 2019.1

Contents

New Features and Enhancements	58
Changes in Behavior	63
Deprecated Features	64
Removed Features	65
Code Generation Defects	65

New Features and Enhancements

Updated Compiler Front-End

Compiler 2019.1 is based on a newer version of the EDG C++ front-end. The new front-end includes many bug fixes and improvements in standard compliance. The default severity levels of some diagnostics may have changed.

New Option for Single-Threaded Performance

A new optimization option (**-single_threaded**) can improve performance in single-threaded programs. For more information, see “Program is Single-Threaded” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Arena malloc

A new version of `malloc` can divide the heap into separate arenas, allowing less contention when allocations occur in multi-threaded applications. For more information, see “Malloc Version” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Support for Synchronization and Hint Ininsics on the Cortex-M23

The following intrinsics are now supported on the Cortex-M23:

```
int __LDREX(const int *Rn);
int __STREX(int Rm, int *Rn);
unsigned char __LDREXB(const unsigned char *Rn);
unsigned short __LDREXH(const unsigned short *Rn);
int __STREXB(unsigned char Rm, unsigned char *Rn);
int __STREXH(unsigned short Rm, unsigned short *Rn);
void __CLREX(void);
void __SEV(void);
void __WFE(void);
void __WFI(void);
void __YIELD(void);
```

For more information, see “Synchronization Primitives” in Chapter 14, “Macros, Pragma Directives, and Intrinsics” in *MULTI: Building Applications for Embedded ARM v2023.5* and “Hint Intrinsics” in Chapter 14, “Macros, Pragma Directives, and Intrinsics” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Support for v8 Instruction Set Intrinsics

The following intrinsics are now available for v8 or higher architectures:

```
int __LDA(const int *Rn);
int __LDAB(const unsigned char *Rn);
int __LDAH(const unsigned short *Rn);
int __LDAEX(const int *Rn);
int __LDAEXB(const unsigned char *Rn);
int __LDAEXH(const unsigned short *Rn);
void __STL(int Rm, int *Rn);
void __STLB(unsigned char Rm, unsigned char *Rn);
void __STLH(unsigned short Rm, unsigned short *Rn);
int __STLEX(int Rm, int *Rn);
int __STLEXB(unsigned char Rm, unsigned char *Rn);
int __STLEXH(unsigned short Rm, unsigned short *Rn);
```

For more information, see “v8 Instruction Set Intrinsics” in Chapter 14, “Macros, Pragma Directives, and Intrinsics” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Support for v8M Intrinsics

The following intrinsics are now supported on v8M processors:

```
int __TT(const void *Rn);
int __TTT(const void *Rn);
int __TTA(const void *Rn);
int __TTAT(const void *Rn);
```

For more information, see “Cortex-M Instruction Set Intrinsics” in Chapter 14, “Macros, Pragma Directives, and Intrinsics” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Improved Assembler/Disassembler Support for the v8M Baseline Instruction Set

The toolchain provides improved support for assembling and disassembling the v8M instruction set. For example, the assembler and disassembler have been updated to recognize the same set of system registers that are supported with the `__MSR()` and `__MRS()` compiler intrinsics. For more information, see “Cortex-M Instruction Set Intrinsics” in Chapter 14, “Macros, Pragma Directives, and Intrinsics” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Option to Conform to ARM Procedure Call Standard

The new `-arm_aapcs` option allows you to compile code that fully conforms to the ARM Procedure Call Standard. For more information, see “ARM Procedure Call ABI Version” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Improved Wildcard Matching in Linker Directives Files

The toolchain's link stage has been made faster for certain projects with detailed linker directives files. These detailed files typically contain large numbers of complex wildcard patterns in their section inclusion rules. As a side effect of this improvement, some ambiguous cases in which multiple rules apply may be prioritized differently. In general, the toolchain will prioritize more specific rules over less specific rules. Ties between equally specific rules will be resolved by their order in the linker directives file. Note that this new implementation will make subtler distinctions about rule specificity. To restore the old behavior for backwards compatibility, pass `-lnk=-old_section_wildcard_matching`. For more information, see “Linker-Specific Options” in Chapter 8, “The elxr Linker” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Option to Disable ARM Cortex-M DSP Extensions

The new `-disable_cortexm_dsp` option allows you to stop the generation of Cortex-M DSP instructions on supported processors. For more information, see “Disable Arm Cortex-M DSP Extensions” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Linker Option to Identify Separately Linked Absolute Images

A new option (**-only_data_absolute**) will produce errors any time a function is imported from a separately linked absolute image. For more information, see the table “Linker-Specific Options” in Chapter 8, “The elxr Linker” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Linker Options to Provide Errors for Symbols Importable from Multiple Files

When importing symbols from a separately linked absolute image, a new option (**-ambiguous_absolute_error**) will produce errors for symbols that can be imported from multiple import files. A companion option (**-allow_ambiguous_abs**) allows you to exclude specific symbols from these errors. For more information, see the table “Linker-Specific Options” in Chapter 8, “The elxr Linker” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Option to Assign Nameless Variables to Individual Data Sections

When used in conjunction with **-individual_data_sections**, a new option allows you to control whether compiler-created objects will be assigned to individual data sections (**-individual_data_sections_for_nameless_vars**). This new behavior is enabled by default when using **-individual_data_sections**. For more information, see “Individual Variable Sections for Nameless Variables” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

C++ Class Array Construction No Longer References new/malloc

To ease the development of C++ applications that do not reference `malloc`, the compiler no longer generates calls to the `__vec_new` run-time library function (or related functions) when emitting code to construct arrays with automatic or static storage duration. Instead, the compiler will generate calls to `__vec_new_no_alloc` (or related functions) that do not directly reference `new/malloc`.

Previously, although `__vec_new` did not actually call `new/malloc` for arrays with automatic or static storage duration, it did have a direct reference to `new` (and thus `malloc`) that would not be optimized away.



Note

When writing a C++ application that does not reference `malloc`, considering using the **-delete** option, as it may optimize away some indirect references to `malloc`.

New Attribute and Option to Enforce Section Order

A new section attribute (`MONOTONIC`) can now be used in linker directives files to prevent reordering of sections for which multiple memory blocks are specified. For more information, see “Using Section Attributes” in Chapter 8, “The elxr Linker” in *MULTI: Building Applications for Embedded ARM v2023.5*.

A new option (**-monotonic_block_order**) can also be used to instruct the linker to treat all sections as `MONOTONIC` by default. For more information, see “Monotonic Block Order” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Duplicate Strings Removed from DWARF Information

The toolchain now removes duplicate strings from the DWARF information generated by **-dual_debug** (see “Generate MULTI and DWARF Information” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*). This can significantly reduce the size of DWARF information in cases of numerous duplicated strings. For example, if the same header is included in multiple modules, only one instance of each string from the header is now included in the DWARF output.

Improved Memory Functions for INTEGRITY Applications

The toolchain can now link in its versions of memory functions when building INTEGRITY applications. This may improve run-time performance when these functions are being used. For more information, see “Enable Toolchain Versions of Certain Memory Functions” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Changes in Behavior

EDOM and ERANGE Not Defined in <math.h>

To improve conformance with the C standard, the macros `EDOM` and `ERANGE` are now defined only in `<errno.h>`. In prior versions, they were also defined in `<math.h>`. Note that this change in behavior may cause build errors if your code uses `EDOM` or `ERANGE` without including `<errno.h>`. If upgrading to 2019.1 coincides with an error like the following:

```
error #20: identifier "EDOM" is undefined
```

add this line to the error-triggering file:

```
#include <errno.h>
```

Changes to Alignment Reduction Behavior with `__attribute__((aligned(x)))`

The compiler will now ignore and give a warning for instances of `__attribute__((aligned(x)))` that attempt to reduce the alignment of fields, classes, structs, or unions. This behavior is designed to minimize accidental under-alignment of entities and is more consistent with the GNU compiler. Purposeful under-alignment can still be achieved with structure packing.

Updated Aggregate Parameter Alignment Attributes

Previously, under the **-align8** calling convention, structs aligned to 8 bytes or more (by an attribute or `#pragma`) were allocated to even-numbered integer registers. The compiler will now base this process on the natural alignment of a struct (derived from its most-aligned member), rather than adjustments to the struct as a whole. This change reflects recent clarifications in the AAPCS.

Assembler Now Enforces Range Limit of Thumb CBZ/CBNZ Instruction

The assembler now gives an error when CBZ or CBNZ destinations are out of range. Previous versions would silently replace an out-of-range CBZ with a two-instruction

compare-and-branch sequence. This was not always equivalent, as it had the side effect of setting condition codes (unlike CBZ).

When `std::terminate()` is Called, `std::uncaught_exception()` Now Returns "false"

The C++ standard now directly conveys that an exception is considered implicitly handled when `std::terminate()` is called. This means that said exception is no longer "uncaught" and thus `std::uncaught_exception()` should return `false`. The compiler now implements this behavior.

Real-Time `malloc` is No Longer Experimental

The real-time `malloc` option `-malloc_version=realtime` is now fully supported.

New `malloc` Default

Arena `malloc` is now the default setting of `-malloc_version` for nearly all single/multi-threaded cases. For more information, see "Malloc Version" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM v2023.5*.

Scoped `enum` Without Explicit Base Type Always Given Base Type of `int`

A scoped `enum` without explicit base type is now correctly assigned a base type of `int`, even when `-short_enum` is enabled. Prior releases used a smaller type in some cases. Note that this change creates incompatibility with object files built with prior releases.

Deprecated Features

The following features have been deprecated and will be removed in a future release:

- Support for MULTI 6.x.

Removed Features

The following features are no longer available:

- The prelinker. Because the prelinker is no longer used by the toolchain, **-prelink_against** is no longer supported.
- The **--no_link_once_templates** option. (The **--link_once_templates** option is now on by default.)
- The **--implicit_include** option.
- The **--export** option (for C++ export templates).
- The **stabs2dbo** utility.

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 160.

Chapter 9

Changes in Version 2018.5

Contents

New Features and Enhancements	68
Changes in Behavior	70
Removed Features	72
Code Generation Defects	72

New Features and Enhancements

Support for MISRA 2012

The toolchain now supports the 2012 version of the Motor Industry Software Reliability Association (MISRA) standards. MISRA 2012 is a set of C guidelines designed to enforce good engineering practices when developing embedded automotive systems. For more information, see “MISRA C 2012” in Chapter 13, “Coding Standards” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Option for Enhanced Loop Unrolling

A new option (**-Ounrollmore**) is available to direct the compiler to unroll larger loops, sometimes with larger unrolling factors. This option replaces **-Ounrollbig**. For more information, see “Unroll Loops More” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Updated Support for C++11 and C++14

The Compiler has improved conformity with the C++11 and C++14 standards. The following features are now supported:

- Namespace members `std::exception_ptr` and `std::nested_exception`.
- Class Members `std::promise::set_exception` and `std::promise::set_exception_at_thread_exit`.
- Exceptions thrown by functions called with `std::packaged_task` and `std::async`.

Improved gsrec Capabilities

The **gsrec** utility now allows output of up to 248 data bytes in a single hex record, and supports padding of multi-core archives. For more information, see “The gsrec Utility Program” in Chapter 10, “Utility Programs” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Option and Attribute to Perform Address Space Layout Randomization

A new option (`--aslr_seed`) and attribute (`RANDOMIZE`) can now be used to jointly perform address space layout randomization (ASLR). ASLR can improve the security of your program. For more information, see “Address Space Layout Randomization” in Chapter 8, “The elxr Linker” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Enhanced Line Information in Conditional and Logical Operators

The compiler now generates better debugging line information for the operands of the conditional (`?:`) and logical (`||`, `&&`) operators. When written on separate lines, these operands can be stepped through individually. For example:

```
void foo(int *p_int)
{
    if (p_int != 0 &&
        *p_int > 10 &&      /* Now has its own breakdot */
        *p_int < 20) {      /* Also now has its own breakdot */
        /* do something */
    }
}
```

There are some limitations for this behavior. Different settings (particularly optimization strategies) may result in fewer or differently placed breakdots. Additionally, there may be code that needs to be evaluated unconditionally at the end of an expression. This code will usually be associated with the last operand of the operator. This is illustrated in the example below; the code for the assignment to `some_int_val` will typically be associated with the `(some_int_value - 1);` line. Since the assignment must happen unconditionally, it may falsely appear that the `(some_int_value - 1);` operand was also evaluated unconditionally.

```
some_int_value = (p_int != 0) ?
    *p_int :
    (some_int_value - 1);    /* May appear to be evaluated even
                             if *p_int is non-NULL */
```

Expanded `asarm` Support for Pseudoinstructions and Alternative Syntax

The `asarm` assembler now supports a number of additional syntax options. These include accepting 2-op shortcuts for 3-op instructions, and recognizing the optional `.w` suffix in ARM mode.

Support for ARMv8M Security Instructions

The security instructions available on the Cortex-M23 and Cortex-M33 are now supported. These are as follows: `TT`, `TTT`, `TTA`, `TTAT`, `BXNS`, and `BLXNS`.

`-no_data_in_text` Supported on the Cortex-M23

The option `-no_data_in_text` is now supported on the Cortex-M23. Using `-no_data_in_text` on the Cortex-M23 requires little-endian mode and `-align8`. For more information, see “Placement of Literal Data in Text” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New CPU Support

The following CPUs are now supported:

- Cortex-M35P (`-cpu=cortexm35p`).
- Cortex-A55 (`-cpu=cortexa55`).
- Cortex-A75 (`-cpu=cortexa75`).
- Kryo (`-cpu=kryo`).

Changes in Behavior

Change in Accepted Multibyte Characters for Shift-JIS and EUC

When target character encoding is set to Shift-JIS or EUC, library functions that convert from multibyte to wide characters/strings now reject the values `\x80` and `\xff`. These values are not valid starting bytes for a multibyte character in either encoding.

Allocation of Uninitialized Global Variables Now Defaults to `--no_commons`

Global, non-`extern` variables without an initializer are now treated as unique definitions, rather than common data. The old behavior can be restored by passing `--commons` (see “Allocation of Uninitialized Global Variables” in Chapter 3,

“Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*). Note that this change can cause multiply-defined symbol errors in code that depended on the prior behavior. For example, variables declared in header files without the `extern` keyword can trigger such errors. Consider the following example files:

test.h

```
int x;
```

test1.c

```
#include "test.h"
int main(void) { }
```

test2.c

```
#include "test.h"
```

When **test1.c** and **test2.c** are built together, it will now result in the following error:

```
[elxr] (error #539) symbol x multiply defined in:
test1.o
test2.o
[elxr] (error) errors during processing
```

To fix the error, build with **--commons** or alter the code to use only a single definition. This can be done by adding `extern` to the declaration in the header file and adding a definition to one of the source files. For example:

test.h

```
extern int x;
```

test1.c

```
#include "test.h"
int x = 0;
int main(void) { }
```

test2.c

```
#include "test.h"
```

Change in servecode Utility Support

The utility used to generate licensing machine identifiers (**servecode**), will no longer generate or report Elan server codes for Compiler 2017.5 or later. If you need a server code for an older Elan-compatible release, use that version's **servecode**.

Removed Features

The following features are no longer available:

- The option **-Ounrollbig**. For a replacement, use **-Ounrollmore** (see “Unroll Loops More” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*).
- The legacy profiling options **-p**, **-pg**, and **-a**. For a replacement, consider **-coverage=*** (see “Profiling - Block Coverage” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*).
- Support for u-velOSity version 2.2.9.

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 160.

Chapter 10

Changes in Version 2018.1

Contents

New Features and Enhancements	74
Changes in Behavior	76
Deprecated Features	76
Removed Features	77
Code Generation Defects	77

New Features and Enhancements

C++14 Support

The compiler now supports C++14. C++14 includes support for variable templates, binary literals, and relaxed `constexpr` restrictions. As with all supported C++ variations, files built or linked in C++14 mode should not be mixed with files built or linked using other dialects. For more information about C++14, see “C++14” in Chapter 12, “Green Hills C++” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Version of malloc

The compiler now includes an experimental version of `malloc` that uses lockless instructions to speed up small block operations under contention. This improved version can also run in bounded time. For more information, see “Malloc Version” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Option to Control Vectorization of Floating-Point Calculations

A new option (**-vectorize_fp**) allows you to specify whether to allow the vectorization of floating-point calculations. Note that **-frigor=accurate** now implies **-no_vectorize_fp** on NEON processors. Previously, **-frigor=accurate** allowed for floating-point vectorization on NEON processors, but such processors lack support for subnormal floating-point values. If you are not using subnormal values, you may re-enable vectorization with **-vectorize_fp**. For more information, see “Vectorize Floating Point” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Predefined Symbols for Optimization Strategies

The Compiler now provides predefined macros for each optimization strategy. For more information, see “Optimization Predefined Macro Names” in Chapter 14, “Macros, Pragma Directives, and Intrinsics” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Intrinsic to Mitigate Against Spectre Attacks

A new intrinsic `__builtin_speculation_safe_load()` creates a barrier to branch prediction with a sequence of instructions that cannot be exploited by a variant 1 Spectre attack. Please refer to your processor manufacturer's documentation on Spectre before employing this intrinsic. For more information, see “Built-In Intrinsics” in Chapter 14, “Macros, Pragma Directives, and Intrinsics” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Note that for Spectre variant 2, ARM recommends against using retpolines. As such, we do not offer the Spectre 2 mitigation option **-indirect_retpolines** on ARM-based systems. For more information, see the relevant ARM FAQ [<https://developer.arm.com/support/security-update/frequently-asked-questions>]

Improved Support for v6M Intrinsics

A set of intrinsics that were previously only supported for ARM v7 are now supported on ARM v6M:

```
void __DMB(void);
void __DMB_OPT(__ghs_c_int__ option);
void __DSB(void);
void __DSB_OPT(__ghs_c_int__ option);
void __ISB(void);
```

For more information, see “v7 Instruction Set Intrinsics” in Chapter 14, “Macros, Pragma Directives, and Intrinsics” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Improved Allocation of Data Into Program Sections

The compiler now consistently allocates constants of `_Complex` type to read-only data sections. Previously, in some cases those constants would be allocated to read-write data sections. For example:

```
float _Complex foo(void) { return -5.0f + I*10.0f; }
```

Changes in Behavior

INTEGRITY Small Block malloc Moved to libfmalloc

On INTEGRITY, the small block `malloc` is now provided in **libfmalloc.a** rather than **libansi.a**. If **-act_like** is set to 2017.5 or earlier, passing `:deflibraries=ansi` will also cause the linker to link against **libfmalloc.a**. This matches the behavior of the Compiler prior to 2018.1 by using an additional library to provide the same version of `malloc` as before.

Updated `__MULTI_HOST__` Macro

In order to accurately reflect differences between the 32-bit and 64-bit toolchain, the `__MULTI_HOST__` macro has new values. For more information, see “Predefined Macros” in Appendix F, “Customization Files” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New -delete Behavior

The deletion of unused functions via **-delete** now takes place before reporting unresolved symbols. This means that programs referencing unresolved symbols that only occur in unused, deleted functions can now link successfully. Previously, they would give an error for unresolved symbols.

New Default Individual Sections Behavior

When individual function sections or individual data sections are enabled, these options now apply to items in link-once sections such as template instances. This new behavior can be disabled with **-no_individual_linkonce_sections**.

Deprecated Features

The following features have been deprecated and will be removed in a future release:

- Using `#pragma weak foo` where `foo` is a C++ mangled symbol name. To make symbols weak, use `__attribute__((weak))` instead.

- The option **-Ounrollbig**.
- The options **-fast_malloc** and **-no_fast_malloc**. Use **-malloc_version=smallblock** and **-malloc_version=legacy** instead.

Removed Features

The following features are no longer available:

- The **-nofloatio** option.
- The options **--one_instantiation_per_object** and **--hybrid_one_instantiation_per_object**.
- The **--instantiation_dir** option.

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 160.

Chapter 11

Changes in Version 2017.5

Contents

New Features and Enhancements	80
Changes in Behavior	82
Deprecated Features	83
Removed Features	83
Code Generation Defects	83

New Features and Enhancements

64-Bit Toolchain Availability

The Green Hills toolchain is now available as a set of 64-bit binaries. For more information, see “System Requirements” on page 2.

Updated Compiler Front-End

Compiler 2017.5 is based on a newer version of the EDG C++ front-end. The new front-end includes many bug fixes and improvements in standard compliance. The default severity levels of some diagnostics may have changed.

Improved Implementation of C `rand()` Function

The `rand()` function is now smaller, more efficient, and provides better statistical output. Note, however, that the implementation uses a linear congruential generator (LCG). This is a common and fast means of generating pseudo-random numbers that requires minimal RAM usage. However, its mathematical properties are such that it should still not be used when secure pseudo-random properties are required.

Improved Preprocessor Options

The option **-dM** may now be used with the **-E** option to generate a list of `#define` symbols. For more information, see “Driver Options for Intermediate Forms of Output” in Chapter 1, “The Compiler Driver” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New CPU Support

The following CPUs are now supported:

- Cortex-M23
- Cortex-M33

Improved C++11 Support

The compiler now supports `std::condition_variable`, along with all library functions related to it (for example, `std::notify_all_at_thread_exit`). The compiler also now supports the use of parameter pack expansions in `alignas` specifiers.

Increased Floating-Point Execution Speed

Improved instruction scheduling has increased the average floating-point execution speed on Cortex-A processors.

Improved Memory Function Speed

The functions `memmove` and `memcpy` have been rewritten for improved speed.

Improved `#pragma ghs section`

The `#pragma ghs section` directive now supports up to 10,000 user-defined sections.

Enhanced Options for Retaining Temporary Preprocessor Files

You can now specify the directory to retain temporary assembly files (`-asmfile_dir`) and temporary preprocessor files (`-cppfile_dir`). For more information, see “Temporary Assembly File Directory” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5* and “Temporary Preprocessor Output Directory” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Changes in Behavior

Updated `alignof()` Extension Behavior in C++11

The C++11 standard only allows the application of the `alignof()` operator to a type-id, but most implementations allow the operator to be applied to an expression as an extension. For compatibility with other compilers, the behavior of a particular form has changed:

```
char x1 alignas(8);
int x2 = alignof(x1);
```

In prior releases, `x2` had the value of 1. It now has the value of 8.

Changes in Use of `__asm__` Keyword

The compiler now rejects many attempts to allocate variables to specific registers using the `__asm__` keyword. On rejection, the compiler may serve any of the following diagnostics:

- 2034 (**ghs_invalid_register_number**): register is not a globalreg-reserved register.
- 2035 (**ghs_cannot_allocate_specified_register**): cannot allocate "xxxx" to specified register.
- 2036 (**ghs_cannot_allocate_specified_caller_saved_register**): cannot allocate "xxxx" to specified caller-saved register.

We strongly recommend that you avoid using this syntax whenever possible. For more information, see “GNU C/C++ Extensions” in Chapter 11, “Green Hills C” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Change in Linker Directive Files

Linker directive files (`.ld`) can now reference symbols imported from external images using the `-A` option.

Temporary Demo Incompatibility

The "Data Graphing (C++)" demo included with MULTI 7.1.6 and earlier is not compatible with this compiler release. Demo compatibility will be restored upon the availability of MULTI 8.

Deprecated Features

The following features have been deprecated and will be removed in a future release:

- The `rand_r2()` function.

Removed Features

The following features are no longer available:

- Support for Solaris-hosted toolchain components.
- The **-host64** option. This option was previously enabled by default on 64-bit operating systems. It allowed the selective use of some 64-bit toolchain components (notably the linker and **dblink**) if your project required more build-time resources than 32-bit binaries could accommodate. If your host machine is running a 64-bit operating system, you should use the newly available 64-bit toolchain (see "System Requirements" on page 2).
- The **grawrite** utility. This utility was designed to write raw binaries to floppy disks on Windows. (Unix-like systems have the **dd** command available for this purpose.) If you need to write raw binaries to floppy disks, free utilities are available online. For more information, contact Green Hills Support.
- Macraigor On-Chip Debugging (**ocdserv**).

Code Generation Defects

For a list of known code generation defects, see "Code Generation Defects" on page 160.

Chapter 12

Changes in Version 2017.1

Contents

New Features and Enhancements	86
Changes in Behavior	87
Removed Features	88
Code Generation Defects	88

New Features and Enhancements

New CPU Support

The following CPUs are now supported:

- Cortex-R8
- Cortex-A32
- Cortex-A35
- Cortex-R52
- Cortex-A72
- Cortex-A73

New Option for ax Librarian

The ax librarian now supports the **D** command modifier for compatibility with the GNU archiver. For more information, see “Librarian Command Modifiers” in Chapter 9, “The ax Librarian” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Improved INTEGRITY Support for `--large_vtbl_offsets`

In previous releases, the compiler did not support the use of `--large_vtbl_offsets` with INTEGRITY version 11.4 or earlier. With Compiler 2017.1, `--large_vtbl_offsets` is supported on all versions of INTEGRITY and is enabled by default. With this change, the use of `--no_large_vtbl_offsets` on INTEGRITY is no longer directly supported. Contact Green Hills Support for assistance if `--no_large_vtbl_offsets` is required on INTEGRITY.

Note that this change accompanies a change in library names. Previously, Compiler-provided run-time libraries that included large virtual table offsets also included “_xvtbl” as part of their filenames. Now that this is the default state, the “_xvtbl” text has been removed. Legacy builds of the run-time libraries without large virtual table offsets are now designated with the addition of “_svtbl” to their names. The compiler driver automatically selects the correct library build variant for the supplied virtual table offset size.

Changes in Behavior

Modification of memmove Behavior

The inline version of `memmove()` has been removed, and thus the function will no longer be affected by **-Omemfuncs**.

The Compiler Now Generates Template Information Files in Fewer Cases

The **.ii** and **.ti** file types are now only generated if the prelinker will be invoked as part of the build process. If you are using a custom build environment with special handling for these files, it may need to be updated to reflect this change.

Modification of Boolean Comparison Behavior

In C, the compiler will no longer output diagnostic 1937 ("possibly unintentional comparison of bool to non-bool") when comparing a `bool` against a literal 0 or 1. This new behavior is useful for the C99 dialect because the `true/false` macros in **stdbool.h** are required to expand to integer literals. Without the change, the compiler would flag expressions such as `(mybool == false)`.



Note

Diagnostic 1937 is normally a remark, but is elevated to a warning by **-ghstd=2010**. Note also that this change in behavior does not apply to C++ because the language has proper `true/false` literals.

Change in Rebuilding Behavior of gbuild

In previous releases, a rebuild of a project containing multiple instances of the same file could cause **gbuild** to unexpectedly produce different code than the prior build (see “Building a Project with Multiple Instances of a File” on page 163). This was caused by using multiple instances of the same file with non-identical options and the same output location. Now, **gbuild** will always build any such file multiple times, once for each set of different options. If the resulting object files are configured to output to the same location, **gbuild** will output the message `"final`

`command has changed`" and any subsequent rebuild will cause this process to repeat and require relinking the application. To revert to the old behavior, pass **-no_rebuild_for_opt_change**.

If you are building INTEGRITY 11.4 or earlier and you see the prior **gbuild** message, it is likely because these versions of INTEGRITY included multiply instanced files in the manner described above. Note that this issue has never been observed to result in a difference in linked INTEGRITY images.

New Option Name for **-display_error_number**

The option **-display_error_number** is now **-display_diag_number**. The name has been changed to more accurately reflect that the option pertains to all diagnostic messages, not simply errors. The old name (**-display_error_number**) will continue to be recognized by the compiler.

Removed Features

The following features are no longer available:

- K&R C support.
- Windows XP support.
- Agilent Processor Probe (**hpserv**) support.
- For INTEGRITY versions after 11, the option **--munch**.
- For INTEGRITY 11 and newer, the option **-generatemonolithdeletions**.
- The `<varargs.h>` Facility.

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 160.

Chapter 13

Changes in Version 2016.5

Contents

New Features and Enhancements	90
Changes in Behavior	93
Deprecated Features	97
Removed Features	98
Code Generation Defects	98

New Features and Enhancements

C++11 Support

C++11 brings numerous improvements, including enhancements to performance, multithreading, and template metaprogramming. As with all supported C++ variations, files built or linked in C++11 mode should not be mixed with files built or linked using other dialects. For more information about C++11, see “C++11” in Chapter 12, “Green Hills C++” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Options Related to C++11

Along with C++11 support, the compiler now provides a number of related options:

- **--user_defined_literals** — Controls support for C++11-style user-defined literals. See “User-Defined Literals” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.
- **--thread_local_storage** — Controls support for the C++11 `thread_local` storage class specifier. (Used to indicate that a variable should reside in thread-local storage.) See “Thread-Local Storage” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.
- **--deprecated_nonconst_string_literal_conv** — Controls support for the deprecated conversion from string literal to `non-const char *` in C++11 mode. See “Allow String Literals to be Assigned to non-const char Pointers” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.
- **--narrowing_conversions** — Controls whether narrowing conversions are allowed in C++11 mode. See “Allow Narrowing Conversions in C++11 Dialects” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.
- **--thread_safe_initializations** — Controls whether initializations are thread-safe.

Support for Disabling Run-Time Type Information (RTTI) in C++11 Mode

Disabling run-time type information is now supported for C++11 mode. For more information, see “Run-Time Type Information Support” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Automatic NEON SIMD Vectorization

For NEON-capable ARMv7 or higher targets, there is now an option to automatically transform loops performing repeated operations on arrays (**-OV**). This setting makes use of the NEON SIMD instruction set to perform multiple operations in parallel. Previously, NEON instructions were only available in compiled code via intrinsics. For more information, see “Automatic Vector Optimization” in Chapter 4, “Optimizing Your Programs” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Data Alignment Support for 16 and 32 Bytes

For non-leaf functions, new modes allow you to maintain the stack at 16 or 32-byte alignments (**-align16/-align32**). In addition, these modes allow you to align sufficiently large NEON vector and matrix data types by the indicated amount, along with optional alignment hints to speed up NEON memory accesses. Note that using **-align16/-align32** with a Green Hills probe requires **mpserv** version 5.4 or greater. For more information, see “Data Alignment” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Fast Simulator Improvements

The following enhancements have been made to the fast simulator for ARM:

- Improvements to ARMv7M NVIC simulation and exception delivery.
- Support for the Generic Timer extension defined by ARMv7 when used with INTEGRITY.

Argument Type Checking

A new option to perform link-time argument type checking is now available. The **-argcheck** option will direct the linker to issue a warning when an externally visible function is called with a type that does not match its definition. For more information, see “Link-Time Argument Type Checking” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Floating-Point Rigor Option

A new option (**-frigor**) allows you to make accuracy/performance tradeoffs when calculating floating-point values. For more information, see “Floating-Point Rigor” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Option to Add Extra Dot to Individual Section Names

For individual section names, the compiler now supports an option to add an extra dot to separate base section names from variable or function names (**-individual_section_name_extra_dot**). For example, the individual section name for function `<func>` in `.text` would be `.text.<func>`. For more information, see “Extra Dot in Individual Section Name” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Option to Set Lower Bound of p_align

The option **-min_p_align** allows you to set a minimum value for the `p_align` field in ELF headers. For more information, see the table in “Linker-Specific Options” in Chapter 8, “The elxr Linker” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Changes in Behavior

Default C Language Dialect is Now C99

The default C language dialect is now ISO C99 (**-c99**). Note that if **-act_like** is set to `2015.1` or earlier, the compiler will still default to ANSI (**-ansi**). For more information, see “C Language Dialect” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Changes in `fenv.h`

The **`fenv.h`** header file no longer defines the floating-point exception macros `FE_DIVBYZERO`, `FE_INEXACT`, `FE_INVALID`, `FE_OVERFLOW`, and `FE_UNDERFLOW`. In addition, the `FE_ALL_EXCEPT` macro is now defined as `0`. This change more clearly indicates the compiler's lack of support for floating-point exceptions.

Changes in `abort()` to Conform to C++ Standard

In previous releases, `abort()` would call `exit()` if the abort signal was not implemented or if it returned. This had the side effect of calling static destructors and `atexit()` functions, which is incompatible with the C++ standard. When faced with the same circumstances, `abort()` now calls `_Exit()`. This will not invoke static destructors or `atexit()` functions.

Changes in Standard C++ Behavior

A number of changes have been made to Standard C++ mode. The primary differences visible to a Standard C++ application are listed below.

- The `std::ios_base::failure` class now uses the C++11 specification, even in Standard C++ mode. This is done to ensure binary compatibility between C++11 and Standard C++ modes.
- The class `std::system_error` is now the parent class of `std::ios_base::failure`. Previously, the parent class was `std::exception`.

- The string returned by the `what()` member function now contains the user-provided message string as a sub-string. Previously, the returned string was identical to the user-provided message string.
- The `char` specialization of `std::char_traits::lt()` now behaves the same as the built-in operator `<` for type `unsigned char` rather than `char`. The old behavior can be re-enabled by building with `-D__CHAR_TRAITS_CHAR_LT_COMPAT`.
- The class function `std::money_get::do_get()` now requires at least one whitespace character where `money_base::space` appears in any of the format pattern's initial elements.
- The first argument to `std::codecvt::length()` is no longer declared as `const`.
- The `std::istream::operator>>(streambuf *)` extraction operator performs unformatted input to conform to the C++11 standard. This means it will update the number of characters successfully read and stored for the `gcount()` member function.

C++ Binary Compatibility Across Compiler Versions

C++ headers and libraries in 2016.5 are not binary compatible with C++ code compiled with previous versions of the toolchain. Any projects with modules or libraries containing C++ should be fully rebuilt prior to linking with the updated toolchain. (Note that INTEGRITY 10 or later always includes the source files necessary to rebuild the C++ libraries.) Attempting to link together modules built with incompatible versions of the toolchain will give an error of the form:

```
[elxr] (error #104) Possible C++ ABI incompatibility between
modules libcxinit.a and hello.o
```

This diagnostic can be temporarily suppressed by passing `-elxr_diag_suppress 104`. However, this is not long-term solution. You should rebuild or obtain updated libraries to avoid this error in the future.

C++ Stream Locking

The `std::basic_streambuf` class no longer implements locking, along with the non-template versions of `streambuf` in the Embedded/Extended Embedded C++

libraries. All `streambuf`-derived classes that do not override the `_Lock()` and `_Unlock()` members are similarly affected. (For example, `std::basic_stringbuf`.) If your program requires locking in any of these classes, you must implement your own subclass that overrides the `_Lock()` and `_Unlock()` members.

Note that the `std::basic_filebuf` class in the C++ library and non-template versions of `filebuf` in the Embedded/Extended Embedded C++ libraries are unaffected by this change.

Treatment of Duplicate Inline Functions and Linkonce Template Instantiations

Previously, the linker would strictly check the sizes of linkonce template instantiation duplicates, along with inline function duplicates. Now the linker will ignore these size differences. To restore the previous behavior, pass the linker option **`-lnk=-strict_linkonce_warning`**.

Stricter Diagnosis of Constructor Inaccessibility

The compiler is now stricter in diagnosing constructor inaccessibility. For example, in permissive dialects, warnings used to be output when referencing inaccessible copy constructors if calls to them would be elided. These cases now result in an error, as illustrated below:

```
struct Foo {
    Foo() { }
    Foo(int a) { }
private:
    Foo(const Foo &other) { }
};

int main()
{
    Foo f = Foo(0); /* Now an error, even in permissive modes
                     and even though the call to the copy
                     constructor will be elided. */
}
```

Changes in Allowed Extensions

The following extension of `basic_string::append` is no longer supported:

```
append(const CharT* first, const CharT* last)
```

An alternative is as follows:

```
append(const CharT* s, size_type count)
```

Modifying Libraries with ax Librarian

When using the **ax** Librarian with the `-rC` argument, the specified old *libraryfile* (if it exists) is now removed before any *memberfiles* are added. If the operation fails before the final *libraryfile* is written, no *libraryfile* will exist. Previously, the old *libraryfile* was not removed prior to adding *memberfiles* and was left unchanged if the operation failed. To revert to the old behavior, pass the **-no_remove_old_archive_with_C** option to the librarian. For more information, see “Modifying Libraries” in Chapter 9, “The ax Librarian” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Changes in Symbol Names for Function Scope Local Static Variables

In previous releases, the symbol name for a local static variable with function scope was `.Ln`, where `.Ln` was a compiler-generated label that did not reference the variable's name. In this release, the symbol name is now a mangled string that incorporates the name of the variable and the name of the function. If more than one local static variable has the same name (as is possible if they are in different scopes), a number is added for differentiation.

This change also affects the individual section names for local static variables with function scope, which now also use the mangled names. For more information, see the example in “Individual Function and Variable Sections” in Chapter 2, “Developing for ARM” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Thumb2 Assembler Encoding Change

The ARM assembler has been modified to conform to the Unified Assembler Language (UAL) syntax. With this modification, 16-bit thumb instructions that explicitly set the `s` bit in opcode are only accepted outside IT blocks. Within IT blocks, 32-bit thumb2 versions of these instructions are used instead. (This is necessary because condition codes are set in the latter case.) To determine if this modification will change the behavior of hand-written assembly code, pass `-asm=-thumb2condcheck`. The `thumb2condcheck` setting will produce syntax errors if conditions are explicitly set within IT blocks.

Using `-asm` Now Implies `-noobj`

Using `-asm=options` now implies `-noobj` if neither `-obj` nor `-noobj` are currently specified. Previously, compiling C or C++ files using `-asm=options` bypassed the assembler unless `-noobj` was specified or implied by another option. To restore the old behavior, specify both `-obj` and `-asm=options`.

Deprecated Features

The following features have been deprecated and will be removed in a future release:

- The **stabs2dbo** utility.
- The option to disable `bool` type support in C++. When this option is removed, `bool` type support will be obligatory.
- The option to turn off link-once template instantiation (`--no_link_once_templates`).
- The options `--one_instantiation_per_object` and `--hybrid_one_instantiation_per_object`. For the purpose of preventing multiple definition errors with template instantiations when building libraries with C++ code, the `--link_once_templates` option may be used as a replacement. In addition, the prelinker is also deprecated, as its sole purpose was to support the above deprecated options along with `--export`, which is already deprecated.
- Support for INTEGRITY version 5.x and earlier.

Removed Features

The following features have been removed:

- Instantiation of the C++ template `std::complex<T>` for types `T` other than `float`, `double`, and `long double`.

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 160.

Chapter 14

Changes in Version 2015.1

Contents

New Features and Enhancements	100
Changes in Behavior	105
Removed and Deprecated Features	106

New Features and Enhancements

Project Build Configurations

Compiler 2015.1 makes it much easier to set up multiple build configurations of the same project. In previous releases, you might have created multiple Top Projects—each of which included shared source—to manually set up multiple build configurations. Now you can easily create your own build configurations within a single project structure. For more information, see the description of **defineConfig** in “Group 1 Directives” in Appendix C, “Green Hills Project File Format” in *MULTI: Building Applications for Embedded ARM v2023.5*.



Note

This feature is only available with MULTI 7.x and newer.

New CPU Support

The following CPUs are now supported:

- Cortex-M7 (`cortexm7`)

Automated Setup of Multicore Targets

Two new options, **-ghsmc_file** and **-ghsmc_core_count**, enable the automated setup of multicore targets. Note that these options are only available for MULTI 7 or newer. For more information, see “Multi-Core Configuration File” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*. See also “Multi-Core Core Count” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Improved Configurability of the Tools for INTEGRITY

The options **-rtos_library_directory** and **-rtos_include_directory** have been added to improve configurability of the tools for INTEGRITY. These options will be used by a future release of INTEGRITY.

Using Interprocedural Optimizations with Makefiles

Interprocedural optimizations are now officially supported for programs built using makefiles. See “Interprocedural Optimizations” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*, and see Example 1.4. Using Interprocedural Optimizations With Makefiles in *MULTI: Building Applications for Embedded ARM v2023.5*.

Duplicate Symbol Detection in Linker and Archiver

The linker now has an option to detect symbols that are present in more than one library being linked. While legal, and often useful, symbols being defined in multiple libraries can also be the source of subtle problems. The options **-link_reject_duplicate_lib_syms**, **-link_allow_duplicate_ghs_syms**, **-link_allow_duplicate_linkonce_syms**, and **-duplicate_whitelist** are added to control the behavior of the linker. For more information, see “Report an Error When Multiple Libraries on a Link Line Contain a Definition of the Same Symbol” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

The archiver now has an option to detect multiple definitions of a symbol when creating a library. The options **-reject_duplicates**, **-allow_cxx_duplicates**, and **-allow_linkonce_duplicates** are added to control the behavior of the archiver. For more information, see “Error on Duplicate Symbols in an Archive” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Preserving Temporary Preprocessor Files

Two new options, **-keepcppfiles** and **-keepasmfiles** can be used to retain temporary preprocessor output and temporary assembly files, respectively. See “Temporary Preprocessor Output” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5* and “Temporary Assembly Files” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5* for more information.

Support for Stripping Debug Information From Relocatable Objects and Libraries

A new option, **-stripreloc** has been added to the **gstrip** utility. When this option is passed, **gstrip** can be run on relocatable object files and libraries, in addition to executable files. When stripping relocatable object files and libraries, **gstrip** will preserve any symbols necessary for linking. See “The gstrip Utility Program” in Chapter 10, “Utility Programs” in *MULTI: Building Applications for Embedded ARM v2023.5* for more information.

Section Sorting in Linker

The linker now supports the new `SORT` and `SORT_BY_NAME` commands in linker directives files to allow sections included with wildcards to be sorted by name. See “Using Wildcards in Section Inclusion Commands” in Chapter 8, “The elxr Linker” in *MULTI: Building Applications for Embedded ARM v2023.5* for more information.

Generating Both Numerically and Alphabetically Sorted Symbols in Map File

A new option, **-Man** is now supported by the linker to produce both a list of symbols sorted alphabetically (as with the **-Ma** option) and numerically (as with the **-Mn** option). See “Map File Sorting” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5* for more information.

Annotating ELF Files with Toolchain Version Information

A new option, **-version_info** is now supported to add information about the compiler and assembler version and command line options to the `.comment` section of the output file. See “Write Version Info to .comment Section” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5* for more information.

Fast Simulator Improvements

The fast simulator for ARM has seen the following improvements:

- Support for additional CP15 registers defined by ARMv7
- Support for address translation according to ARM VMSAv7
- Minimal support for LDREX/STREX instructions
- Updated exception handling support
- Better support for Thumb supervisor mode instructions

Preventing Literal Data in Text

A new option, **-no_data_in_text**, is now supported on ARMv7 and higher processors to prevent generating literal data in executable sections by the tools. The run-time libraries for these architectures for both ARM and Thumb mode are now built with this option. See “Placement of Literal Data in Text” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5* for more information.

Link-time Global Variable Type Checking

Two new options, **-globalcheck=[full|normal|none]** and **-globalcheck_qualifiers**, are now supported to perform link-time checks of global variable types. When **-globalcheck** is enabled, the compiler generates information about global variable data-types at their definition and uses. The linker checks that the types are compatible with respect to size and signedness. With **-globalcheck_qualifiers**, the qualifiers (such as `const` and `volatile`) are also checked. See “Link-Time Global Variable Type Checking” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5* and “Link-Time Global Variable Type Qualifiers” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5* for more information.

Aligned Data Sections

A new option, **-split_data_sections_by_alignment**, is now supported to allocate variables into aligned data sections. See “Aligned Data Sections” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5* for more information.

Individual Function and Variable Sections

Individual function and variable sections are now supported for assigning functions (including static functions), global variables and static variables (including function scope local static variables) in default sections or custom sections to their own individual sections by the compiler. These individual sections can then be placed at arbitrary addresses via linker directives file or be merged back to the sections they derived from implicitly at link time. For more information, see “Individual Function and Variable Sections” in Chapter 2, “Developing for ARM” in *MULTI: Building Applications for Embedded ARM v2023.5*.

64-Bit Linker and dblink on Windows and Linux Hosts

The `elxr` linker and `dblink` debug information linker are now supported in 64-bit mode on 64-bit Windows and Linux hosts.

Options to Modify Message Severity Supported for asm and elxr

`asm` and `elxr` now support setting discretionary diagnostic messages to different levels of severity. For more information, see “Toolchain Messages” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Specify the Output Name of a Dependency File

A new option, `-MF`, has been added to specify the output name of a dependency file. This name will be used for files created by the `-MD` or `-MMD` options. For more information, see “Generating Dependency and Header File Information” in Chapter 1, “The Compiler Driver” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Additional Documentation of Diagnostic Messages for Assemblers and elxr

Additional documentation of diagnostic messages has been added to the *MULTI: Toolchain Error Messages and Other Diagnostics* book.

Ability to Specify Diagnostics by Symbolic Tag Names

Several constructs that previously required a diagnostic to be specified by number now accept symbolic tag names as well. These constructs include:

- The `--diag_[suppress|remark|warning|error]` compiler options. (See “Varying Message Severity” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.)
- The `ghs nowarning #pragma` directive. (See “Green Hills Extension Pragma Directives” in Chapter 14, “Macros, Pragma Directives, and Intrinsics” in *MULTI: Building Applications for Embedded ARM v2023.5*.)

For example, these two options are now equivalent:

- `--diag_error=9`
- `--diag_error=nested_comment`

See the *MULTI: Toolchain Error Messages and Other Diagnostics* book for the new symbolic tag names.

Support for %= Format String in GNU Extended Assembly

The compiler now accepts the `%=` special format string in GNU Extended Assembly. The string expands to a number that is unique to each specific instantiation of an `asm` statement. For more information, see Chapter 18, “GNU Extended Assembly” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Changes in Behavior

Diagnostic Messages When Building INTEGRITY 5.x and 10.x

In some rare cases, building INTEGRITY 5.x or 10.x from source may generate additional diagnostic messages. To disable these messages, see “Varying Message Severity” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5* or contact Green Hills Support.

bcmp and bcopy now treat their size arguments as unsigned

The library functions `bcmp()` and `bcopy()` now treat a size argument larger than `INT_MAX` as representing a large size, rather than a negative one. This will cause a long run time due to iteration over large arrays. Previously, both functions would do nothing if their size arguments were larger than `INT_MAX`.

Removed and Deprecated Features

The following feature has been deprecated:

- Support for the `--prelink_objects` option. As an alternative for build systems that use `clearmake`, please see the `--link_once_templates` option.

Chapter 15

Changes in Version 2014.1

Contents

New Features and Enhancements	108
Changes in Behavior	111
Removed and Deprecated Features	115

New Features and Enhancements

New CPU Support

The following CPUs are now supported:

- Cortex-A12 (`cortexa12`)
- Cortex-A17 (`cortexa17`)
- Cortex-A53 (`cortexa53`)
- Cortex-A57 (`cortexa57`)

Cross-Project Implicit Dependency Analysis

Implicit dependency analysis is now supported across Top Projects. As a result, building a Top Project that depends on libraries or objects in another Top Project now builds the defined dependencies. Previously, you had to remember to build these dependencies yourself. This feature is especially useful for INTEGRITY users whose applications depend on source code in the INTEGRITY project tree. For more information, see “Implicit Dependency Analysis” in Chapter 10, “Utility Programs” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Undefined Behavior Checking

The compiler now checks for undefined behavior resulting from improperly sequenced expressions. By default, these diagnostics are issued as warnings, but they may be issued as errors via the `--diag_error=2017,2018` option. Alternatively, these diagnostics may be suppressed altogether via the `--diag_suppress=2017,2018` option.

For more information, see “Sequence Related Undefined Behavior Checking” in Chapter 11, “Green Hills C” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Range-Based For Loops

This release adds support for C++11-style range-based for loops. For more information, see “Range-Based For Loop Support” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

auto Type Keyword

This release allows you to use the `auto` keyword as a type specifier, as opposed to a storage class, for C++11-style type deduction. For more information, see “auto Type Keyword Support” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Improved Inlining in C++

The heuristics used by the compiler to determine whether or not to inline functions in C++ have been updated. Depending upon the circumstances, this may result in slight improvements in both size and speed in C++ programs.

Additionally, when using GNU C++ compatibility mode, function templates are now considered for inlining under a broader range of compiler options. Previously, function templates were only considered for inlining in GNU C++ compatibility mode when the function qualified for automatic inlining or when either intermodule inlining or `--max_inlining` was enabled for the compilation.

For more information, see:

- “Using Inlining Optimizations” in Chapter 4, “Optimizing Your Programs” in *MULTI: Building Applications for Embedded ARM v2023.5*,
- “Intermodule Inlining” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*, and
- “C++ Inlining Level” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Ability to Check for Use of Floating Point

The compiler is now able to give a warning or error for the use of floating point. Furthermore, this diagnostic can be given for the use of double precision and long double precision floating type, or for all floating point types. Previously, the only checking the compiler offered was to give a fatal error for all floating point types. For more information, see “Do not allow types double or long double” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Intermodule Variable Allocation

This release allows global and static variables to be allocated during interprocedural analysis, which can improve code quality when functions reference globals or are inlined into callers that are defined in other modules. For more information, see “IP Allocate Variables” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Calls to `pow(x, y)` Now Optimized in Certain Cases

If the optimization strategy **-Ospace**, **-Ogeneral**, or **-Ospeed** is enabled, and **-ffunctions** is enabled, calls to `pow(x, y)` will be optimized if either `x` or `y` has one of the following constant integer values:

- If `x` has constant value 2 or 10, `pow` is converted to an equivalent call to `exp2` or `exp`.
- If `y` has constant value 2, 3, ... 8, `pow` is converted to a series of multiplies.

If the optimization strategy **-Ospace**, **-Ogeneral**, or **-Ospeed** is enabled, and the optimization is not disabled by the new option **-Ono_inline_constant_math**, calls to `exp2(x)` (where `x` is a constant integer) and calls to `sqrt(x)` (where `x` is a finite, positive constant) will be replaced with an equivalent constant value.

Generating a Target-Walkable Stack Now Supported in Thumb Mode

Generating a target-walkable stack (**-gtws**) is now supported in both ARM and Thumb mode. It was previously supported only in ARM mode. For more information,

see “Generate Target-Walkable Stack” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Ability to Control Generation of Hardware Divide Instructions

The compiler now allows you to control the generation of hardware divide instructions regardless of CPU type. Previously, the compiler generated these instructions only for certain CPU types. For more information, see “Hardware Divide Instruction” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

VFP v4 Now Supported

This release adds support for the v4 version of the VFP instruction set. For more information, see “Floating-Point Coprocessor Version” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Inlining Constant Math Functions

Calls to certain math functions with constant parameters are now optimized into a constant expression. For more information, see “Inline Constant Math Functions” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Changes in Behavior

Changes in the Behavior of Some #pragma Directives

The following #pragma directives now have no effect when used inside a function. Additionally, using any of these #pragma directives inside a function will trigger a compiler warning:

- #pragma ghs extra_stack
- #pragma ghs max_stack
- #pragma ghs max_instances

- `#pragma alignfunc`

Previously, when used inside a function, these `#pragma` directives silently affected the next function declared (not the enclosing function).

index() and rindex() Prototypes Now Hidden

The prototypes for functions `index()` and `rindex()`, which are non-standard, are no longer visible unless:

- The `__BSD_INDEX_PROTOTYPE` symbol is defined, or
- The operating system is `INTEGRITY` and `_POSIX_SOURCE` is not defined.

Change in Default Behavior for Virtual Function Table Size

The `--large_vtbl_offsets` option is now enabled by default unless you are building for an incompatible version of `INTEGRITY` (version 11 or earlier). In addition, `--large_vtbl_offsets` is incompatible with `MULTI` versions earlier than 7.x, with code built using Compiler versions 2012.5 or earlier, and with code built using `--no_large_vtbl_offsets`. If you have an existing code base built using exceptions with `EC++` or `EEC++`, you will now need to also pass `--no_large_vtbl_offsets`.

The `--no_large_vtbl_offsets` option is now deprecated for all versions of `INTEGRITY` that follow `INTEGRITY 11`.

For more information about Virtual Function Table Offset Size, see “Virtual Function Table Offset Size” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New C++ Restrictions Regarding --large_vtbl_offsets

When using `--large_vtbl_offsets` (either explicitly or by default), you must use a C++ library with the same exception handling support as the compiled code. In general, the driver will select an appropriate library for you, but if you explicitly select a library (for example, with `--stdl` or `--stdle`), you must ensure that it matches your selected level of exception handling.

If you use an invalid combination of options according to the above rules (for example, by specifying **--stdl** and **--exceptions**), the driver will issue a diagnostic similar to the one below:

```
Error: Option "--stdl" requires option "--no_exceptions"
```

See “Specifying C++ Libraries” in Chapter 12, “Green Hills C++” in *MULTI: Building Applications for Embedded ARM v2023.5*, and see “C++ Exception Handling” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Furthermore, the **--large_vtbl_offsets** option now requires that the **--link_once_templates** option is also used. Note that while **--link_once_templates** is also enabled by default, manually disabling it with the **--no_link_once_templates** option will imply **--no_large_vtbl_offsets**. Explicitly specifying both **--no_link_once_templates** and **--large_vtbl_offsets** is an invalid combination.

See “Virtual Function Table Offset Size” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*, and see “Link-Once Template Instantiation” in Chapter 12, “Green Hills C++” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Link-Once Template Instantiation Now Enabled by Default

The **--link_once_templates** option is now enabled by default.



Note

For targets that support large virtual function table offsets, disabling link-once template instantiation (via the **--no_link_once_templates** option) will imply **--no_large_vtbl_offsets**.

Limited Support for C-Like Structure Packing in C++

The compiler now has more explicit support for C-like structure packing in C++. In general, only C-like usage of packed structures (or classes) is supported. Most C++ specific features (for example, references, pointers to members, etc.) are not supported in combination with structure packing, nor were they before.

Packing of non-POD (Plain Old Data) classes is not supported unless the **-misalign_pack** option is enabled. (For the purposes of this restriction, the C++98/03 definition for POD is used.)

In conjunction with the above restriction, a warning will be issued if you explicitly attempt to pack a non-POD. Additionally, a warning will be issued if the current structure packing level, which is determined by the **-pack=*n*** command line option and any active `#pragma pack` directives, would have otherwise had an effect on a non-POD (that is, only if one of the fields of a non-POD would have had its alignment changed by the current structure packing level).



Note

A “Plain Old Data” class is a class that has no constructors, no destructors, no private non-static data members, no protected non-static data members, no base classes, no virtual functions, and no non-POD members.

New Default for Generation of Hardware Divide Instructions

--hardware_divide is now enabled by default in ARM mode for the Cortex-R5.

-gs No Longer Implies -gtws

The option **-gs** no longer implies **-gtws**. This improves code size and speed.

Instantiate Extern Inline Now Enabled by Default

The **--instantiate_extern_inline** option is now enabled by default.

While **--instantiate_extern_inline** usually results in smaller program sizes, the results depend upon the code being compiled. In general, **--instantiate_extern_inline** leads to better program size when the majority of non-static inline functions have more than one call site (across the entire program). However, when the majority of non-static inline functions have exactly one call site, this option may lead to worsened program size. This can be mitigated (without disabling **--instantiate_extern_inline**) by marking inline functions as static when they are only used in one module.

sqrt() and sqrtf() Now Return NAN for Negative Input

Library functions `sqrt()` and `sqrtf()` now return a quiet NAN instead of 0.0 when the input is less than zero.

Removed and Deprecated Features

The following features are no longer available:

- The **regnum** debug server command
- The **rg** debug server command

The following feature has been deprecated:

- Support for Exported Templates (**--export**).

Chapter 16

Changes in Version 2013.5

Contents

Product Compatibility	118
New Features and Enhancements	118
Changes in Behavior	120

Product Compatibility

Green Hills Compiler 2013.5 is officially supported with INTEGRITY versions 5.x, 10.0.2, 11.0.2, 11.0.4, and 11.2.2. For information about compatibility with newer versions of INTEGRITY, see the INTEGRITY release notes.

Green Hills Compiler 2013.5 is officially supported with ThreadX 5.5.x and newer.



Warning

Using Compiler 2013.5 with MULTI 6 or later and a Green Hills Debug Probe requires Green Hills Debug Probe software version 4.4.4 or newer. Do not install Green Hills Debug Probe software version 4.2.x or earlier over your Compiler 2015.1 installation if you intend to use it with MULTI 6 or later.



Note

If your MULTI IDE is newer than the compiler, see the IDE release notes for up-to-date product compatibility.

New Features and Enhancements

Static Assertions

This release adds a C11/C++11 style `static_assert` in permissive C and C++ language dialects, that allow you to statically check conditions at compile time with a standard mechanism. For more information, see “Static Assertions” in Chapter 14, “Macros, Pragma Directives, and Ininsics” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Specify a Compiler Inside a Project File

You can now specify the location of the Compiler installation directory that should be used to build your project from the Project Manager using the **gbuildDir** option inside a project (**.gpj**) file as a Group 1 option. For example:

```
#!gbuild
gbuildDir=c:\ghs\comp_201354
```

```
primaryTarget=ppc_standalone.tgt  
[Project]  
...
```

This option only affects projects built from the Project Manager; it is ignored by the **gbuild** utility program.

With MULTI 6 and earlier, this option silently defaults to the Compiler installation directory that the MULTI IDE is linked with in the event that a bad path is supplied.

For more information, see “Group 1 Directives” in Appendix C, “Green Hills Project File Format” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Ability to Specify the Size of `wchar_t`

You can now select the size of the `wchar_t` type using the **-wchar_u16** and **-wchar_s32** options to conserve memory when necessary. For more information, see “WChar Size” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Stack Smashing Protection

The new **-stack_protector** option provides protection against stack smashing attacks by using a *stack canary* — a random number on the stack — that is compared to an expected value in function epilogues. For more information, see “Check for Stack Smashing Attacks” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Memory Clearing Optimization Control

An optimization that changes `memset()` filled with zeros to optimized calls to `memclr()` can now be controlled using the **-Omemclr** and **-Onomemclr** options. For more information, see “Convert C Memset to Memclr” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Updated gstack Utility Program for Better C++ Analysis

The **gstack** utility program has been updated to provide better support for analyzing function calls involving C++ virtual functions, startup code, and libraries. For more information about **gstack**, see “The gstack Utility Program” in Chapter 10, “Utility Programs” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Attribute and Intrinsic to Remove Specific Functions from Profiling Reports

The new `ghs_noprofile` attribute and `__ghs_noprofile_func()` intrinsic allow you to mark functions so that any code paths they appear in are not considered when MULTI builds reports from coverage data gathered with **-coverage=**.

If you are using MULTI 6.1.4, you must enable the **-Xprofileignorefatalblocks** switch to use this feature. With this version of MULTI, the following caveats apply:

- The **Coverage** tab will always display 0 for **% Covered**, but the **Block Detailed** tab will continue to display count information for the block.
- Coverage line coloring in the Debugger is reversed for functions with the `ghs_noprofile` attribute.
- The `ghs_noprofile` attribute is ignored by **TimeMachine** profiling.

Changes in Behavior

GNU Compatibility Updated for Better Compatibility with Version 4.3

The GNU C and C++ dialects have been updated to provide closer compatibility with the 4.3 version of the GNU compiler. To revert to a version of these dialects that is more closely compatible with version 3.3, use **--gnu_version=30300** in conjunction with the **-gcc** or **-g++** language dialect option. Make sure that you have also selected a GNU dialect before specifying this option.

Additionally, dependent name processing is now on by default when using the GNU C++ dialect, even when emulating version 3.3 of the GNU compiler. To disable dependent name processing, use **--no_dep_name**.

Furthermore, the parsing of function template definitions is done lazily in the GNU C++ dialect. Even though the parsing of a template in its generic form (without argument substitution) is typically done to resolve non-dependent names, this parse is only done for templates that are actually instantiated, and the parse itself is deferred until the entire input file has been processed. This may result in the apparent suppression of errors in unused function templates. This may be disabled with the **--no_defer_parse_function_templates** option.

For more information, see:

- “GNU C/C++ Extensions” in Chapter 11, “Green Hills C” in *MULTI: Building Applications for Embedded ARM v2023.5*
- “Dependent Name Lookup” in Chapter 12, “Green Hills C++” in *MULTI: Building Applications for Embedded ARM v2023.5*
- “Deferral of Function Template Parsing” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*

Chapter 17

Changes in Version 2013.1

Contents

Product Compatibility	124
New Features and Enhancements	124
Removed and Deprecated Features	125

Product Compatibility

Green Hills Compiler 2013.1 is officially supported with INTEGRITY versions 5.x, 10.0.2, and 11.0.4. For information about compatibility with newer versions of INTEGRITY, see the INTEGRITY release notes.

Green Hills Compiler 2013.1 is officially supported with ThreadX 5.5.x and newer.



Warning

Do not install Green Hills Debug Probe software version 4.2 or earlier over your compiler installation.



Note

If your MULTI IDE is newer than the compiler, see the IDE release notes for up-to-date product compatibility.

New Features and Enhancements

New CPU Support

The following CPUs are now supported in version 2013.1:

- Cortex-M0+ (`cortexm0plus`)
- Cortex-M4F (`cortexm4f`)
- Cortex-R4F (`cortexr4f`)

Relaxed Size Restrictions

The compiler no longer restricts arrays and structures to 256 megabytes in size. It now allows uninitialized arrays and structures declared as up to 2 gigabytes on 32-bit targets, and over 2 gigabytes on 64-bit targets.

The `--large_vtbl_offsets` option eliminates the 64 kilobyte limit on classes with multiple virtual inheritance. For more information, see “Virtual Function Table Offset Size” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

There may be some issues with debugging information for large arrays and structures, such as warnings issued from **dblink**, problems viewing these objects in the MULTI Debugger, or problems when these objects are involved in command line function calls. In addition, large programs may not load into the simulator if they consume too much RAM on the host.

protrans Supports Converting Section Dumps to .pro Files

If you use one of the **-coverage=** options to collect profiling information from your program and dump that program into a file on your host computer, you can now use **protrans** to convert that file into a **.pro** file using the **-cc**, **-cd**, or **-cf** option. For more information, see “The protrans Utility” in Chapter 10, “Utility Programs” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Removed and Deprecated Features

The following features have been deprecated:

- Call count and call graph profiling (**-p** and **-pg**).
- Run-mode debug over a freeze-mode debug connection with hardware targets running INTEGRITY 5. Run-mode connections are still supported over Ethernet and serial ports.
- Socket emulation over freeze-mode host I/O with hardware targets running INTEGRITY 5. Socket emulation is still supported over run-mode debug, or you can use the GHnet v2 TCP/IP stack on the target.

Chapter 18

Changes in Version 2012.5

Contents

Product Compatibility	128
New Features and Enhancements	128
Changes in Behavior	130
Removed and Deprecated Features	131

In the Green Hills Compiler 2012.5 release, we focused on improving the **gstack** utility program, improving profiling data collection, and increasing C++ performance.

Product Compatibility

Green Hills Compiler 2012.5 is officially supported with INTEGRITY versions 5.x, 10.0.2, and 11.0.4. For information about compatibility with newer versions of INTEGRITY, see the INTEGRITY release notes.

Green Hills Compiler 2012.5 is officially supported with ThreadX 5.5.x and newer.



Warning

Do not install Green Hills Debug Probe software version 4.2 or earlier over your compiler installation.



Note

If your MULTI IDE is newer than the compiler, see the IDE release notes for up-to-date product compatibility.

New Features and Enhancements

Updated gstack Utility Program

The **gstack** stack analysis program has been greatly improved with the ability to analyze recursive functions, function pointers, and interrupts. Its output has also been improved to provide detailed information to guide you in annotating your program for the most accurate results. For more information, see “The gstack Utility Program” in Chapter 10, “Utility Programs” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New CPU Support

The following CPUs are now supported in version 2012.5:

- Cortex-R5 (`cortexr5`)

- Cortex-R7 (`cortexr7`)
- Cortex-A7 (`cortexa7`)

New Block Profiling System

The **-coverage=** option provides a faster method for measuring both performance and code coverage than the legacy **-a** method. If you only want to test for coverage and not execution counts, **-coverage=flag** provides a significantly more efficient method than **-a**. For more information, see “Enabling Instrumented Coverage or Performance Profiling” in Chapter 2, “Developing for ARM” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New ax Modifier

The **ax** librarian now includes the **u** modifier for compatibility with the GNU archiver.

New Linker Options to Control Linker p_align Behavior

The new **-max_p_align**, **-any_p_align**, and **-no_p_align** options allow you to specify how the linker determines the maximum value of the `p_align` field in ELF program headers. For more information, see “Linker-Specific Options” in Chapter 8, “The elxr Linker” in *MULTI: Building Applications for Embedded ARM v2023.5*.

-fsingle Supported on More Targets

The **-fsingle** option is now supported on all targets that support **-fhard**. For more information about these options, see “Floating-Point Mode” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

New Stack Check Attribute

The new C/C++ `stackcheck` attribute allows you to enable stack checking on a per-function basis. For more information, see “`stackcheck`” in “Attributes” in

Chapter 11, “Green Hills C” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Changes in Behavior

Clean Builds Recommended

To mitigate any problems that may occur due to small changes in behavior from previous versions, we recommend that you clean and rebuild your programs from scratch when you build them for the first time with the upgraded compiler.

Error Compiling `ind_initmem.c`

When porting a customized **libstartup** from an earlier version to Green Hills Compiler 2012.5, one of the following messages may be given when compiling **`ind_initmem.c`**:

```
#error This file should only be included from another file in the same directory
#error Please include <stddef.h> instead of ghs_null.h
```

To fix this problem, edit **`ind_initmem.c`** to replace:

```
#include <ghs_null.h>
```

with:

```
#include <stddef.h>
```

For more information, see “C and C++ Header Files” in Chapter 15, “Libraries and Header Files” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Vector's Clear Method No Longer Deallocates

Prior to Green Hills Compiler 2012.5, vector's `clear()` method deallocated memory, resetting the capacity to zero, even if the `reserve()` method had been called. In Green Hills Compiler 2012.5, `clear()` no longer deallocates memory.

To deallocate a vector's memory, call the `swap()` method instead. This technique is portable across versions of the Green Hills Compiler and other STL implementations:

```
std::vector<T> v;  
...  
std::vector<T>().swap(v);  
  
std::vector<T> v;  
...  
std::vector<T>(v).swap(v);
```

This change to `clear()` can be reverted by defining the `__GHS_VECTOR_CLEAR_DEALLOCATES` macro to 1. This macro might be removed in a future version of the compiler.

Implicit Inclusion is No Longer Enabled By Default

Implicit inclusion is no longer enabled by default. If you require this feature, pass the `--implicit_include` option to the driver.

Eclipse Plug-in Option Type Changes

In the Green Hills Eclipse plug-in, the types of a few options have changed. If you have existing Green Hills Eclipse Managed Make projects that you created using an older version of the Green Hills Eclipse plug-in and you encounter options that do not work well (for example, they display erroneous or garbled text values in the C/C++ Build Settings dialog) we recommend creating a new project using the 2012.5 version of the plug-in and re-importing your sources into the new project, rather than reusing the old project with the new plug-in.

ghide No Longer Hides COMMON Symbols

The **ghide** utility no longer hides `COMMON` symbols. For more information about **ghide**, see “The ghide Utility Program” in Chapter 10, “Utility Programs” in *MULTI: Building Applications for Embedded ARM v2023.5*.

Removed and Deprecated Features

The following features have been deprecated:

- The **Legacy Coverage Profiling (-a)** driver option.

Chapter 19

Changes in Version 2012.1

Contents

Product Compatibility	134
New Features and Enhancements	134
Changes in Behavior	137
Removed and Deprecated Features	138

In the Green Hills Compiler 2012.1 release, we focused on improving build performance, enhancing existing optimizations, and making architectural changes that allow you to upgrade the MULTI IDE and Compiler separately. This version of the compiler also adds GHS Standard Mode and coding standard profiles, two new features that help you improve code quality.

Product Compatibility

Green Hills Compiler 2012.1 is officially supported with INTEGRITY versions 5.x, 10.0.2, and 11.0.4. For information about compatibility with newer versions of INTEGRITY, see the INTEGRITY release notes.

Green Hills Compiler 2012.1 is officially supported with ThreadX 5.5.x and newer.



Warning

Do not install Green Hills Debug Probe software version 4.2 or earlier over your compiler installation.



Note

If your MULTI IDE is newer than the compiler, see the IDE release notes for up-to-date product compatibility.

New Features and Enhancements

Separate Releases Allow Greater Flexibility When Upgrading

The Compiler and IDE products now release independently of each other, allowing you to upgrade them separately. This makes it possible to freeze on one version of the Compiler while upgrading the version of the IDE, or vice versa. This release structure also allows Green Hills Software to release compilers with new CPU support on a faster schedule.

Faster Builds

Build speeds have increased significantly in this release of the MULTI Compiler. Build speed on Windows hosts has increased, and parallel build speed has improved

up to 50%. In addition, **gbuild** now builds in parallel by default, so users that did not explicitly pass **-parallel=** in previous versions of MULTI may see much larger improvements. In one real-world case, a customer reported a 4.5x increase in build speed.

New CPU Support

The following CPUs are now supported in version 2012.0:

- `cortexa15`
- `cortexa5`
- `cortexm0`
- `cortexm1`
- `cortexm4`

New Flash Support

Flash support has been added for the following chips in version 2012.0:

- Kinetis K40
- Kinetis K60

Define Macros in Any Project File

You can now define macros in any project file, not just your Top Project.

New Layout for Better Debugging When Running Out ROM

The Green Hills Compiler now ships with a layout that allows you to execute your program out of ROM on a target while retaining certain Debugger features that were not previously available when running out of ROM, such as:

- System Calls (`printf`, `exit`, file I/O, etc.)
- Command Line Function Calls
- Run-Time Error Checking

- Memory Checking and Leak Detection

To use this layout, specify `-layout=romdebug` to the compiler driver or select **Link to and execute out of ROM, with enhanced debugging** from the Project Manager.

New Coding Standard Features

The Green Hills Compiler now ships with GHS Standard Mode, an internally-designed collection of diagnostics that help you avoid common pitfalls while developing programs. We use this standard internally because its rules are pragmatic to apply to existing projects and helpful for improving code quality.

The compiler also provides coding standard profiles, a feature that allows you to create your own coding standards by packaging diagnostic severity levels into a single file to distribute among all developers in your organization. For more information, see Chapter 13, “Coding Standards” in *MULTI: Building Applications for Embedded ARM v2023.5*.

UTF-8 Support

The Green Hills Compiler now supports multi-byte characters encoded in UTF-8. For more information, see “Host and Target Character Encoding” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM v2023.5*.

ARM NEON Support

The Green Hills Compiler now supports ARM NEON extensions. ARM NEON is a Single Instruction stream Multiple Data stream (SIMD) engine for enhancing performance in a variety of applications.

Unused Virtual Function Deletion Now Supported for ARM Targets

You can now delete unused virtual functions automatically with the `-uvfd` and `-force_uvfd` options.

Changes in Behavior

New Licenses Required

The MULTI licensing system was updated in this release. If you are upgrading from an older release, you must obtain new licenses. For more information, please contact Green Hills Support.

Clean Builds Recommended

To mitigate any problems that may occur due to small changes in behavior from previous versions, we recommend that you clean and rebuild your programs from scratch when you build them for the first time with the upgraded compiler.

gbuild Defaults to Parallel Builds

gbuild now builds programs in parallel by default, and automatically determines the number of builds to run at once based on the number of CPUs on your host machine. You do not need to specify **parallel=*n*** to **gbuild** unless you want to limit the number of CPUs used for building programs.

Different malloc() Implementation Used By Default in Stand-Alone Projects

The compiler uses a new `malloc()` implementation by default for all Stand-Alone projects. This implementation generally provides better performance, but has a slightly larger library footprint. To use the old implementation instead, pass the **-no_fast_malloc** option.

__MULTI_DIR__ Customization File Macro Changes

If you use the predefined `__MULTI_DIR__` macro in your customization files, replace each instance with `__TOOLS_DIR__`, which points to your MULTI Compiler installation directory. There is no longer a macro defined for the MULTI IDE installation directory.

Pointers Are Now Unsigned by Default

Pointers are now unsigned by default for all architectures, and shipping libraries are built with **--unsigned_pointer**. If you require signed pointers, pass the **--signed_pointer** option, but keep in mind that operations that rely on memory spanning 0 may not work consistently between user code and the libraries.

memmove Moved to libind.a

The `memmove()` function has been moved from **libansi.a** to **libind.a**. If your build fails because `memmove()` cannot be located, include **libind.a**.

Removed and Deprecated Features

The following features are no longer available:

- Windows 2000 support
- Elan licenses
- The **gbldconvert** utility that converts **.bld** files to **.gpj** files. If you are migrating from MULTI 4 or earlier and need to convert your **.bld** files, contact Green Hills Support for more information.

The following features have been deprecated:

- **gproxy**
- **grun**
- P&E Lightning card support in **peserv**
- The `truncate()` function in **libsys.a**.
- **--cxx_include_directory** (use the **-I** driver option instead).

Chapter 20

Changes in Version 5.4

Contents

Product Compatibility	140
New Features and Enhancements	140
Removed Features	140

Product Compatibility

The 5.4 compiler has been tested for compatibility and is officially supported with:

- INTEGRITY version 10.0.x
- Green Hills Probe version 3.8.x

For information about the compatibility of the 5.4 compiler with older versions of these products, contact your Green Hills sales representative. For information about compatibility with newer versions of these products, see the release notes for the appropriate product.

New Features and Enhancements

UTF-8 Support

The compiler now supports UTF-8 encoding for extended characters, in addition to EUC-JP and Shift-JS. For more information, see “Extended Character Support” in the *MULTI: Building Applications* book.

Updated MULTI Eclipse Plug-In

The MULTI Eclipse plug-in now supports Eclipse 3.7.x and CDT 8.0.x. For more information, see the documentation about the MULTI Eclipse plug-in in the *MULTI: Building Applications* book.

Removed Features

Driver No Longer Accepts .bld Files

The driver and Project Manager no longer build projects in the legacy **.bld** format, and the **gbldconvert** utility that converts files from **.bld** to the newer **.gpj** format has been deprecated. Support for converting **.bld** files will be removed in the next release.

Chapter 21

Changes in Version 5.2

Contents

Product Compatibility	142
New Features and Enhancements	142
Changes in Behavior	143
Removed Features	147
Deprecated Features and Discontinued Support	148

Product Compatibility

The MULTI 5.2 compiler has been tested for compatibility and is officially supported with:

- INTEGRITY version 10.0.x
- Green Hills Probe version 3.8.x
- ThreadX version 5.5.x and newer.

For information about the compatibility of the MULTI 5.2 compiler with older versions of these products, contact your Green Hills sales representative. For information about compatibility with newer versions of these products, see the release notes for the appropriate product.

New Features and Enhancements

New -Omoredebug Optimization Strategy

The new **-Omoredebug** optimization strategy has been added in this release and is enabled by default for new projects. This strategy enables only optimizations that do not affect debugging ability. Additionally, the compiler intentionally generates code so that:

- Variable lifetimes are extended to let you view variable values at any point in the function, and
- Every statement has a breakpoint.

Note that in some cases, **-Omoredebug** may cause your program's stack size to increase.

Support for Mixed Endianness

The compiler now supports generation of big endian data with little endian code. For more information, see the documentation for the **-bigendian** option in MULTI: Building Applications for Embedded ARM.

Updated Compiler Front-End

MULTI Compiler 5.2 is based on a newer version of the EDG C++ front-end. The new front-end includes many bug fixes, improvements in standard compliance, and improvements in GNU compatibility. The default severity levels of some diagnostics may have changed.

Updated Fast Flash Programmer

The fast flash programmer includes several enhancements:

- faster SREC handling
- delta flashing capabilities (only re-flash changed sectors)
- improved logging
- support for new flash parts

For more information about which flash parts are supported, contact your Green Hills sales representative.

General Improvements

The following features have seen major improvements:

- Wholeprogram optimizations
- Interprocedural optimizations

Changes in Behavior

New Warnings Enabled by Default

The **--prototype_warnings** option is now set by default. This option may generate warnings that indicate bugs in your code.

Changes to Line Continuations in GNU C Mode

In versions of the MULTI compiler earlier than 5.2, a backslash (\) followed by a space was never parsed as a line continuation. As of 5.2, in the GNU C or C++ dialect, the compiler parses backslash as a line continuation character whether or not it is followed by a space.

If you compile your code using the GNU C or C++ dialect and it has a line that contains a backslash followed by a whitespace, it might create an unintentional line continuation. The following example shows one of these cases (the space is indicated by an underscore, because it does not display in print):

```
#include <stdio.h>
int main(void) {
    int i = 42; // use space after backslash, as in C:\_
    i = 0; // ignored in GNU
    if (i != 0) {
        printf("i = %d, should be 0\n", i);
    }
}
```

When the compiler encounters one of these cases, it issues diagnostic 1400. If you need to hide this diagnostic, pass the option **--diag_suppress=1400** to the driver.

Changes in std::string

Due to changes in `std::string`, the C++ libraries included with this release are incompatible with the C++ libraries shipped with previous versions of MULTI.

Options Enabled by Default

The following options are now enabled by default in MULTI 5:

- **--alternative_tokens**
- **--no_implicit_extern_c_type_conversion**
- **--no_guiding_decls**
- **--readonly_typeinfo**
- **--readonly_virtual_tables**

The following options are enabled by default unless you pass **--guiding_decls**:

- **--no_old_specializations**
- **--no_implicit_typename**

Map File Format Changes

The format of map files generated by the **elxr** linker has changed in version 5.2. The map file now lists the size of each symbol in addition to its address (in the format *address+size*).

To force the linker to produce map files in the old format, pass the following option to the driver:

```
-lnk=-old_mapfile
```

This option is not supported in conjunction with the **-Mu** or **-MI** linker options.

Changes to the Connection Editor for simarm

The **Connection Editor** for **simarm** now offers three options for **Simulate FPU** in a drop-down menu. The options are:

- **<default>** — Set this option based on the selected CPU.
- **Software** — Explicitly pass **-nofpu**
- **Hardware** — Explicitly pass **-fpu**

Loops with an Iteration Variable Shorter than int May Execute Slower than in Previous Releases

If you have enabled speed optimizations and your program has a loop that uses an iteration variable of a type smaller than **int**, that loop may execute slower than it did with previous versions of MULTI. To work around this issue, use an iteration variable of type **int** or larger, or **int_fastN_t** or **uint_fastN_t**, defined in **stdint.h**.

Mixed Endian Access in ARMv6

When compiling with the **-bigendian** option, the toolchain now generates BE8 code for ARMv6 and later architectures. In previous versions, BE32 was the only supported big-endian format.

Big Endian VFP Doubles

The order of VFP single-precision registers overlapping with double-precision registers is not architecturally defined. When running in big endian mode, the simulator now uses the same register representation as in little endian mode, in order to match known hardware implementations. The Debugger uses this register order when viewing VFP double-precision registers.

Thumb-2 Relocation Number Changes

Relocation numbers for some Thumb-2 instructions have been changed to match the ARM ABI. The 5.2 linker can link objects compiled using previous versions of MULTI, but previous versions of the linker cannot link objects that were compiled with the MULTI 5.2 tools and contain Thumb-2 code.

__alt_init Hook Removed From Stand-Alone Run-time Library

The call to `__alt_init` from the stand-alone run-time routine `__ghs_ind_crt1` has been removed as of the 5.2 release. We recommend renaming `__alt_init` to `__ghs_board_devices_init`, which is called at approximately the same point in the startup sequence. A second approach is to add the following C code to your project:

```
#if defined(__GHS_VERSION_NUMBER) && (__GHS_VERSION_NUMBER >= 520)
void __alt_init(void);
void __ghs_board_devices_init(void) {
    __alt_init();
}
#endif
```

If neither of these approaches work for you, you can customize **libsys.a** and modify `__ghs_ind_crt1` in **ind_crt1.c** to call `__alt_init` prior to calling `main()`. For more information, see “Creating a Project with a Customizable Run-Time

Environment” in “Libraries and Header Files” in the *MULTI: Building Applications* book.

Removed Features

The following features are no longer included in MULTI, but may still be available for purchase separately. For more information, contact your Green Hills sales representative.

- Elan licensing tools
- HP-UX host support
- The following driver options:
 - **--array_new_and_delete**
 - **--common_implicit_initialization/--no_common_implicit_initialization**
 - **--distinct_template_signatures**
 - **--early_tiebreaker/--late_tiebreaker**
 - **--enum_overloading/--no_enum_overloading**
 - **--explicit**
 - **--extern_inline/--no_extern_inline**
 - **--friend_injection/--no_friend_injection**
 - **--include_once/--include_never**
 - **--initpipointers**
 - **--memory_description_file**
 - **--new_for_init/-old_for_init**
 - **-Owholeprogram_libs**
 - **--relative_xof_path/--no_relative_xof_path**
 - **--short_lifetime_temps/--long_lifetime_temps**
 - **--std_qualifier_deduction**
 - **--typename**
 - **-use_vp_as_dbg_source_root/-no_use_vp_as_dbg_source_root**

Deprecated Features and Discontinued Support

Deprecated K&R C Support

Support for the K&R C dialect (**-k+r**) is deprecated and will be removed in the next MULTI release. Certain old-style features, such as K&R-style parameter declarations, will continue to be supported outside of the K&R C dialect.

Deprecated **-gnu** Option

The **-gnu** driver option is deprecated. To get the same behavior as **-gnu**, pass both the **-gcc** and **--g++** options instead.

Chapter 22

Changes in Version 5.0.6

Contents

New Features and Enhancements	150
Changes in Behavior	151
Unbundled Features	155
Deprecated Features and Discontinued Support	156

New Features and Enhancements

Improved Code Generation

MULTI 5 comes complete with Green Hills Software's world class compilers, including new interprocedural optimization and link-time optimization. MULTI 5 boasts compilers which generate some of the industry's smallest and fastest code. Additionally, you can use MULTI 5 to enforce programming guidelines such as the MISRA 2004 coding standards.

Improved Dependency Analysis

Building programs with complicated dependencies (such as an INTEGRITY kernel) has historically required that the user identify libraries that might have changed since the last build in order to avoid link-time errors. MULTI's Improved Dependency Analysis solves this problem by automatically building out of date libraries if the current program is dependent on those libraries. This capability extends to prebuilt libraries or other libraries linked in with an option such as **-l** or **-kernel**.

DoubleCheck Static Source Code Analyzer

The DoubleCheck static source code analyzer reviews code, searching for problems which may lead to bugs in the final product. DoubleCheck is fully integrated with the MULTI IDE. It executes automatically when you build your application. Any problems discovered by DoubleCheck are reported to you alongside syntax errors reported by the compiler. This creates an opportunity for you to immediately fix bugs, saving many hours of debugging time later in your development process.

New Fast Simulator Available

MULTI 5 provides a new fast simulator mode (**-fastsim**) if you do not need legacy feature support. This feature requires additional licenses. To verify the installation of these licenses, see the instructions in the “Fast Mode and Compatibility Mode” section of the *MULTI: Configuring Connections* book.

Improved Simulator Speeds

MULTI 5 comes with backwards-compatible simulators that are up to 20 times faster than the old simulators. You can leverage this speed while prototyping your application.

Fast Flash Programmer

The MULTI IDE's Fast Flash Programmer solves several problems which occur in firmware development. It significantly cuts down on the time and effort required to repair corrupted boot code, fix bugs in firmware and program production boards. The Fast Flash Programmer uses little or no RAM and runs with greater speed than its predecessor. Additionally, the Fast Flash programmer provides effective feedback when hardware causes a programming failure. Operation of the Fast Flash Programmer can be scripted using the MULTI command interface for greater flexibility and power.

Changes in Behavior

Changes to Driver Option Defaults

Builder Option	MULTI 4.2.4 Default	MULTI 5.0.6 Default
Accept GNU <code>__asm__</code> statements	Option does not exist	Off (<code>-no_gnu_asm</code>)
Alternative C++ Tokens	Off (<code>--no_alternative_tokens</code>)	On (<code>--alternative_tokens</code>)
Consistent Code Without Debug Information	Option does not exist	Off (<code>-noconsistentcode</code>)
Constant Propagation	Off (<code>-Onoconstprop</code>)	Depends on optimization strategy
Data Alignment	Option does not exist	4-Byte Boundary (<code>-align4</code>)
DoubleCheck Level	Option does not exist	None (<code>-double_check.level=none</code>)
DoubleCheck Output that Stops Builds	Option does not exist	Off (<code>-double_check.stop_build=off</code>)
Guiding Declarations of Template Functions	On (<code>--guiding_decls</code>)	Off (<code>--no_guiding_decls</code>)
Link-Once Template Instantiation	Option does not exist	Off (<code>--no_link_once_templates</code>)

Builder Option	MULTI 4.2.4 Default	MULTI 5.0.6 Default
-Olimit= Without Debug Information	Option does not exist	Off (-noglimits)
Simplify C Print Functions	Option does not exist	On (-Oprintfuncs)
Stack Limit Checking	Option does not exist	Off (-no_stack_check)
Section Overlap Checking	Warnings (-overlap_warn)	Errors on Non-Zero Overlap (-nooverlap)
Support for C Type Information In Assembly	Option does not exist	Off (-noasm3g)
Support for Implicit Extern C Type Conversion	On (-implicit_extern_c_type_conversion)	Off (-no_implicit_extern_c_type_conversion)
Support for Old-Style Specializations	On (--old_specializations)	Off (--no_old_specializations)
Support for Implicit Typenames	On (--implicit_typename)	Off (--no_implicit_typename)
Treat Doubles as Singles	Option does not exist	Off (-nofloatsingle)

Options Enabled by Standard C++ Dialects

When you select either of the Green Hills Standard C++ dialects (`--std` or `--STD`), MULTI enables various options in order to conform more closely to the C++ standard as defined in International Standard ISO/IEC 14882:1998. MULTI 5 enables a slightly different set of options from MULTI 4:

Options enabled by `--std` and `--STD`:

- `--alternative_tokens`
- `--no_implicit_extern_c_type_conversion`
- `--no_old_specializations`
- `--no_implicit_typename`
- `--no_guiding_decls`

Additional options enabled by `--STD`:

- `--dep_name`

Libind.a Splitup

Some of the contents of the libind.a library from previous releases have been split out into a new library. The libmath.a library is new in MULTI 5, and contains math routines.

Building System Libraries from 4.x Distributions

The MULTI 4 System Libraries and headers make use of **stdio.h** field names that are not preceded by an underscore. In MULTI 5, the names are all defined with a leading underscore.

If you are using MULTI 5 to build a project with system libraries (libsys.a) from a MULTI 4 distribution, you must pass **-D__GHS_OLD_STDIO_FIELD_NAMES** so that MULTI 5's **stdio.h** defines the old names as well.

Global Const Variables Placed in Read-Only Data

In prior versions of MULTI, global `const` variables without an initializer, such as:

```
const int i;
```

were output as common variables (see “Allocation of Uninitialized Global Variables” in Chapter 3, “Builder and Driver Options” in the *MULTI: Building Applications* book). This could cause inconsistencies when using special data areas or when referencing the variable with position-independent code (PIC) or position-independent data (PID).

In MULTI 5, these variables are output as if they have been initialized to zero. The example above would be output in the same way as:

```
const int i = 0;
```

As a result, these variables are typically placed in read-only data.

Debugging Code Optimized for Speed

When debugging code optimized for speed (e.g. **-g -Ospeed**), new optimizations may inhibit debugging. The debugger may not be able to single step correctly over

certain optimized source lines, causing the program to continue running beyond the next line.

Workaround: If you want to debug optimized code, compile with the `-Odebug` optimization strategy for maximum debuggability.

Libraries for PIC Programs

Beginning in MULTI 5, programs compiled for PIC must now use the PID libraries (i.e. those library directories ending in **p** or **pb**). In order to allow for this, the PID register, `r9`, must be initialized in the setup code and maintained unmodified throughout the program for PIC.

Hand-coded assembly routines for PIC programs must be inspected to make sure they avoid modifying `r9`. C and C++ programs compiled for PIC (but without PID) with prior versions of the Green Hills compiler must be recompiled with MULTI 5 to be linked against MULTI 5 libraries.

Assembler Evaluation of Local Thumb Labels

When the assembler takes the address of labels with the `.thumb` attribute, in sections with the code attribute (such as `.text`), with no explicit `.type`, that address will have the 1-bit set to indicate thumb execution. Previously this was done only for labels with the `.type $function`.

Workaround: For labels that do not represent executable thumb code, the labels may be declared `.nothumb`, given a `.type` of `$object`, or placed in sections without the `CODE` attribute.

GNU Assembler Support in MULTI 5

The behavior of the compiler when outputting code for the GNU assembler has changed. Previously, the compiler would automatically insert a `#` character before most numeric literals. In this release, the compiler no longer inserts this character when expanding an asm macro. Users who use constant (“%con”) arguments for asm macros may need to add this character manually to the macro bodies.

Support for XScale Extensions to the ARM Architecture

The XScale instructions provided by the new DSP coprocessor are supported by MULTI built-in functions and the assembler. The coprocessor must be manually enabled before use. For example:

```
#include "arm_ghs.h"
void enableCP0() {
    __MCR(15, 0, 0x3fff, 15, 1, 0);
}
```

Allocation of Memory Pages at Run Time

If you are using the new fast simulator in ROM mode (e.g., with **simarm -rom** or **simppc -rom** or with **simarm -rom_use_entry** or **simppc -rom_use_entry**), the **-allocmem** option is not passed by default, meaning that memory pages are not allocated at run time when simulating accesses to memory outside the known extent of RAM. This behavior differs from that of the old simulator. For more information, see “ROM Mode” in the *MULTI: Configuring Connections* book.

For other differences between the compatibility mode simulator and the new fast simulator, see “Simulator for ARM (simarm) Connections” or “Simulator for PowerPC (simppc) Connections” in the *MULTI: Configuring Connections* book.

Unbundled Features

Unbundled features are no longer included in MULTI, but are still available for purchase separately.

Unbundled in 5.0.6

The following features have been unbundled from MULTI:

- FORTRAN compiler
- CodeBalance
- Distributed Build System
- Remote serial connections

- TraceEdge and In-Memory TimeMachine
- Pre-MULTI-4 debug server protocol compatibility
- EST ICE integration
- ARM Remote Debug Interface (RDI) integration, including ARMulator
- PMON ROM monitor integration
- IDT ROM monitor integration
- Green Hills ROM monitor integration
- HP-UX host support

Deprecated Features and Discontinued Support

Deprecated Options

The following options have been deprecated in this release:

Builder Option	Driver Options
ANSI C and Standard C++ Extensions	<code>--discretionary_errors</code> , <code>--discretionary_warnings</code> , <code>-pedantic-errors</code> , <code>-pedantic</code>
Auto-Instantiation of Templates	<code>--auto_instantiation</code> , <code>-template=auto</code> , <code>--no_auto_instantiation</code> , <code>-template=noauto</code>
Ignore #include Directives	<code>--no_include_never</code> , <code>--include_never</code> , <code>--include_once</code> , <code>--no_include_once</code>
Link Map Method	<code>-default_lnk</code> , <code>-no_default_lnk</code> , <code>-default_i</code> , <code>-no_default_i</code>
Minimum Structure Alignment	<code>--struct_min_alignment</code>
New-Style 'for-init' Code	<code>--no_for_init_diff_warning</code> , <code>--for_init_diff_warning</code>
PCH Messages	<code>--pch_messages</code> , <code>--no_pch_messages</code> , <code>--no_pch_silent</code> , <code>--pch_silent</code>
Precompiled Header File	<code>--no_pch</code> , <code>--use_pch</code> , <code>--pch</code> , <code>--create_pch</code>
Signedness of Enum Type	<code>--unsigned_enum_fields</code> , <code>--signed_enum_fields</code>
Unroll Loops Up to 8 Times	<code>-Ounroll8</code> , <code>-Onounroll8</code>
Use ThreadX Demo Library	<code>-no_use_demo_library</code> , <code>-use_demo_library</code>

Builder Option	Driver Options
Use Before 'Set'	<code>--no_use_before_set_warnings,</code> <code>--use_before_set_warnings</code>
Use Debug Library List (.dli) Files	<code>-no_use_dli_files, -use_dli_files</code>

The ghexfile Utility is No Longer Available

The **ghexfile** utility is no longer available. If you used **ghexfile** in a previous version of MULTI, reconfigure your projects to use **gsrec** instead. **gsrec** includes functionality that was formerly included in the **ghexfile** utility. For more information, see “The gsrec Utility Program” in “Utility Programs” in the *MULTI: Building Applications* book.

Manual Template Instantiation Mode Options No Longer Supported

The following **Manual Template Instantiation Mode** options are no longer supported:

- **-tall**
- **-tlocal**
- **-tused**
- **-tnone**

Programming flash from grun

Programming flash memory from grun is no longer supported in MULTI 5. Similar functionality can be obtained by scripting the flash burn command in the debugger.

Legacy Assembler Syntax Disabled by Default in MULTI 5

Ambiguously named legacy assembler operations (UGT, UGE, ULT, ULE, USHR, ROTL, and ROTR) have been disabled by default. They are instead available in the format `:UGT:`, `:UGE:`, `:ULT:`, `:ULE:`, `:USHR:`, `:ROTL:`, and `:ROTR:`. The deprecated syntax may be reenabled with the assembler option **-named_operators**.

Chapter 23

Known Issues

Contents

Code Generation Defects	160
Installation and Licensing	160
Host Support	161
Toolchain Issues	161
Target Connection Issues	164
Debugging Issues	164
File System Issues	168
Miscellaneous	168

Code Generation Defects

This section contains a summary of known code generation defects for compiler releases. These defects cover problems that could silently result in an incorrect output binary program from otherwise correct source input. For an up-to-date listing, please contact Green Hills Support.

Installation and Licensing

Name of Install Directory Cannot Contain Spaces

The MULTI Compiler cannot be run from a directory whose name contains spaces.

Compiler and Probe Missing From ginstall

If **ginstall** does not list the Compiler and Probe as installation options, you may be trying to install a 64-bit version of the Compiler on a 32-bit host, an invalid setup choice. On Linux, you may additionally see the following errors:

```
ginstall_probe: cannot execute binary file
ginstall_comp: cannot execute binary file
```

To remedy this issue, check your installation tools and note whether they are for a 64-bit host. If so, check if your host supports 64-bits. On Linux hosts, use the **arch** command. On Windows hosts, open **Control Panel** → **System** and look next to "System type". If your host only supports 32-bits, replace it with a 64-bit host. If you need additional assistance, contact Green Hills Support.

Empty Parent Directory Deletion During Uninstall on Windows Hosts

If the MULTI IDE, compiler, or licensing utility are uninstalled on Windows, the uninstaller will also delete the product's parent directory if said directory is now empty. This behavior functions recursively. That is, the uninstaller will delete all successive empty parent directories as it moves up their hierarchy.

Host Support

UNC Paths Are Not Supported in the MULTI Debugger

On Windows hosts with MULTI 6.1.4 or older, creating new projects or debugging projects over a UNC path is not supported. In general, we recommend mapping UNC paths to network drives before trying to access them in the MULTI IDE.

UNC Paths to Virtual Machines Are Not Supported

When remotely accessing a MULTI or Green Hills Compiler installation on a Windows virtual machine using a UNC path, you may get errors that the application could not be found or that it could not obtain a license. To work around this issue, map the path to a network drive and use the drive letter instead of the UNC path.

Toolchain Issues

INTEGRITY Fails to Build

If you install a version of the toolchain and attempt to build INTEGRITY, you may receive an error message like the following:

```
Error: Program 'C \ghs\intl144\multi\bin\win64\intex'
does not exist in this distribution
```

or

```
Error: Program '/bin/sh'
does not exist in this distribution
```

For information about building INTEGRITY, contact Green Hills Support.

NXP MAC71xx Processors and ARM/Thumb Mixed Code

Early revisions of the NXP MAC71xx processors require that Thumb entry points be 4-byte aligned when ARM/Thumb mixed code is used and the code is linked to be run from the external bus interface. Contact NXP or GHS for more information

if you need to run ARM/Thumb mixed code from the external bus. Code run from the internal flash and SRAM is not affected.

gstack Provides Warnings for Green Hills Library Functions

Some functions in the Green Hills libraries are not annotated for **gstack**. If you link your program with these libraries, you may receive warnings about these Green Hills library functions when using **gstack**.

gstack Provides Warnings for INTEGRITY

INTEGRITY has not been modified to take advantage of new **gstack** features. If you use **gstack** with an INTEGRITY program, you may receive warnings about INTEGRITY functions.

Use of -a, -p, or -pg in INTEGRITY KernelSpace Requires -Onomemfuncs

If **-a**, **-p**, or **-pg** are used with INTEGRITY KernelSpace code, **-Onomemfuncs** should be used at the same time.

Cryptographic Extension Not Supported

The optional Cryptography instruction set extension is not supported.

Building Projects with Absolute Path Output Locations

Building a project that specifies output locations with absolute paths will embed these paths in any resultant files. This may cause unexpected behavior if the output files are copied or moved. For example, performing a clean build on a copy of the output files may remove these files from their original directory rather than their current directory. This behavior applies to paths specified with the `%expand_path` macro function. (Note that project files generated by the **New Project** wizard may use the `%expand_path` macro function.)

To work around this issue, perform a clean build prior to copying or moving. This will ensure that no output files reference the old location.

Building a Project with Multiple Instances of a File

If you are using Compiler 2016.5 or earlier and your project contains a source, object, or executable file more than once—and the options associated with the files are not identical—**gbuild** will reuse the output file as long as it was newer than its inputs, even if the options were not identical. To correct this behavior, indicate unique output directories for each instance of the file using **-object_dir** or **:outputDir**.

Error Associated with Calling `sqrt`

In C++03 dialects, calling the `sqrt` function with an integral type argument may result in the following error:

```
more than one instance of overloaded function "sqrt" matches
the argument list
```

The error occurs because the full set of `sqrt` overloads are now indirectly pulled in by more headers (such as `<algorithm>`). A workaround is to cast the argument to `sqrt` to the desired type (for example, `double`).

Toolchain Components Failing to Launch

Components of the Green Hills toolchain may fail to launch on Linux systems using a 32-bit **libc.so** that is built assuming a 16-byte stack alignment. This problem has been observed with Manjaro Linux and may affect other distributions. If you experience any of these issues, we recommend that you switch to a supported Linux version. For more information, see “System Requirements” on page 2.

Increased Compile Time and Host Memory Usage with Complex Templates

The compiler may require substantial host memory and additional time to compile files containing deeply recursive templates. This is particularly true for files containing recursive instances of class templates.

Toolchain File Size Limitation

The toolchain is not guaranteed to work correctly with files that are 2 gigabytes or larger.

Target Connection Issues

Flash Driver Library `libflash.a` Limited to Four CFI Drivers

The flash driver library distributed with the MULTI IDE offers support for Common Flash Interface (CFI)-compatible flash devices. Drivers for these devices are configured at run time. Previous versions of **libflash.a** supported only targets with a single CFI flash device. MULTI 5 now includes support for up to four CFI flash devices. Calls to `FLASH_loadDriver()` on a fifth CFI flash device will return an error code. Multiple calls to `FLASH_loadDriver()` on a single flash base address will return the same CFI driver, even if the hardware is changed or remapped in memory. These limitations do not affect non-CFI drivers. You can force a non-CFI driver by setting the `disableCFI` flag in `FLASH_loadDriver_2()`.

Debugging Issues

Auto Trace Collection Reduces Performance

Automatically enabled trace collection reduces the performance of simulated targets by a significant amount.

To work around this issue, disable automatic trace collection. For instructions, see the documentation about enabling and disabling trace collection in the *MULTI: Debugging* book. Alternatively, you can counteract the loss of performance by using the fast simulator instead of the compatibility mode simulator.

Target Agent May Cause Failures When Programming Flash

The MULTI **Fast Flash Programmer** dialog contains an input field for the RAM base address. During a program operation, the flash programmer downloads target agents to this address. If the memory used by the target agent conflicts with reserved

memory sections, programming may fail. Error messages printed by the flash programmer may indicate that either a hardware exception was hit or the target agent stalled. Check the link map of your executable or the memory map of the board to confirm that the RAM base address entered in the flash dialog is located after the end of any reserved sections.

References Not Available for Inlined Functions

Issuing the **browseref** or **xref** command or right-clicking a symbol (a function, type, member, variable, etc.) and selecting **Browse References** allows you to view all references of a symbol. However, references are not stored, and consequently not displayed, for inlined functions.

Workaround: Disable inlining by compiling your program with **--no_inlining**. Note that functions defined in header files may be implicitly inlined by the compiler if this option is not used.

Cannot Debug Executables in Eclipse Projects with a Space in the Name

You cannot debug executables in Eclipse projects with a space in the name. Do not put a space in the name of an Eclipse project.

TimeMachine is Not Compatible with NEON Instructions

Values displayed in the TimeMachine debugger may be incorrect if your application uses NEON instructions.

TimeMachine Does Not Track VFP Registers

TimeMachine does not attempt to reconstruct the values of VFP registers and displays them as `Unknown`.

Jazelle Mode Trace is Not Supported

Trace collection is not supported for ARM targets that enter Jazelle Mode. If a target enters Jazelle Mode while trace is being captured, further trace collected may be incorrect.

Raw Trace is Not Displayed When Collecting Data Only Trace

When an ARM processor is being traced with data only trace collection enabled, and the **Display raw trace packets** check box is enabled in trace options, the raw trace packets are not displayed in the trace list. Instead, blank lines are displayed in the trace list.

Trace on Cogent CSB637/937

If trace is enabled on the CSB637/937, you may encounter run control stability problems (cannot halt after extended run times).

Workaround: Try configuring the trace interface to an 8-bit port width to improve signal integrity on the run control interface.

Tracing INTEGRITY Applications on AT91RM9200 Targets

The AT91RM9200 processor suffers from an issue which occasionally results in incorrect trace data. The problem sometimes occurs when the kernel is entered from a virtual AddressSpace and when the kernel returns to a virtual AddressSpace. In both cases the problem results in some instructions being traced in the wrong AddressSpace. This problem may affect other ARM7 and ARM9 processors which do not implement the ContextID register, but has only been observed by Green Hills on AT91RM9200 targets.

Workaround: To minimize the impact of this problem, disable the trace options **Abort processing on opcode failure** and **Read unknown opcodes from target**. Both options can be found on the **Analysis** tab of the **Trace Options** window.

C11 Debugging Issues

Some C11 features are not currently well supported by the MULTI Debugger. Below is a non-comprehensive list of issues:

- The C11 `alignof` and `alignas` operators are not supported in the Debugger. If you highlight an `alignof` or `alignas` expression, the Debugger will issue an error: "Unknown name 'alignof' in expression" or "Unknown name 'alignas' in expression".

C++11 Debugging Issues

Some C++11 features are not currently well supported by the MULTI Debugger. Below is a non-comprehensive list of issues:

- The C++11 `alignof` operator is not supported in the Debugger. If you highlight an `alignof` expression, the Debugger will issue an error: "Unknown name 'alignof' in expression."

Memory Leak Detection May Result in False Positives/Negatives

Memory leak detection may fail to identify some memory leaks while incorrectly reporting others. In the later case, this is because the detection algorithms do not traverse through pointers that are represented in an encoded form or stored in a misaligned memory location. Memory that is only reachable through these means will be susceptible to false positives.

Stack Trace Causes Incorrect Disassembly

Source-level debugging information (`-G` or `-g`) is required in order for the debugger to correctly disassemble code that contains literal pools. (Note that the Green Hills C and C++ run-time is built with `-gs`.) Additionally, you should avoid setting breakpoints in literal pools, as doing so may change the underlying data on the target and lead to errant program execution.

File System Issues

Two Dots (..) Not Supported After Symbolic Links in Paths

MULTI and other Green Hills tools do not support paths that use two dots (..) to indicate the parent directory of a symbolic link. For example, if `symlink` is a symbolic link, the following paths are not supported:

```
/projects/symlink/..  
/projects/symlink/foo/../../bar
```

NFS May Cause Poor Performance

Linux only

MULTI tools may hang if they are accessing files over an NFS server that is slow or is not responding. In certain cases, it may not be possible to terminate them, even when using `kill -9`. These problems are caused by a fundamental limitation in the design and implementation of NFS.

Workaround: To help prevent these problems, eliminate references to broken file system mount points in your user configuration directory (`~/.ghs/*`) — in particular in the **integrity.dist** and/or **uvelocity.dist** files located there.

Miscellaneous

Links to Connecting Book

The MULTI documentation set contains links to the *MULTI: Configuring Connections* book. These links are disabled for MULTI and compiler versions designated for native targets.