ARM® Compiler toolchain

Version 5.02

Migration and Compatibility

ARM DUI 0530H (ID070912)

ARM Compiler toolchain
Migration and Compatibility

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
28 May 2010	A	Non-Confidential	ARM Compiler toolchain v4.1 Release
30 September 2010	B	Non-Confidential	Update 1 for ARM Compiler toolchain v4.1
28 January 2011	C	Non-Confidential	Update 2 for ARM Compiler toolchain v4.1 Patch 3
30 April 2011	D	Non-Confidential	ARM Compiler toolchain v5.0 Release
29 July 2011	E	Non-Confidential	Update 1 for ARM Compiler toolchain v5.0
30 September 2011	F	Non-Confidential	ARM Compiler toolchain v5.01 Release
29 February 2012	G	Non-Confidential	Document update 1 for ARM Compiler toolchain v5.01 Release
27 July 2012	H	Non-Confidential	ARM Compiler toolchain v5.02 Release

Proprietary Notice

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

The information in this document is final, that is for a developed product.

Web Address

`http://www.arm.com`

Contents
ARM Compiler toolchain Migration and Compatibility

Chapter 1
Conventions and feedback

The following describes the typographical conventions and how to give feedback:

Typographical conventions

The following typographical conventions are used:

`monospace` Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.

`mono`space Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.

`monospace italic`

Denotes arguments to commands and functions where the argument is to be replaced by a specific value.

`monospace bold`

Denotes language keywords when used outside example code.

italic Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.

bold Highlights interface elements, such as menu names. Also used for emphasis in descriptive lists, where appropriate, and for ARM® processor signal names.

Feedback on this product

If you have any comments and suggestions about this product, contact your supplier and give:

- your name and company

- the serial number of the product
- details of the release you are using
- details of the platform you are using, such as the hardware platform, operating system type and version
- a small standalone sample of code that reproduces the problem
- a clear explanation of what you expected to happen, and what actually happened
- the commands you used, including any command-line options
- sample output illustrating the problem
- the version string of the tools, including the version number and build numbers.

Feedback on content

If you have comments on content then send an e-mail to `errata@arm.com`. Give:

- the title
- the number, ARM DUI 0530H
- if viewing online, the topic names to which your comments apply
- if viewing a PDF version of a document, the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

ARM periodically provides updates and corrections to its documentation on the ARM Information Center, together with knowledge articles and *Frequently Asked Questions* (FAQs).

Other information

- ARM Information Center, `http://infocenter.arm.com/help/index.jsp`
- ARM Technical Support Knowledge Articles, `http://infocenter.arm.com/help/topic/com.arm.doc.faqs/index.html`
- ARM Support and Maintenance, `http://www.arm.com/support/services/support-maintenance.php`
- ARM Glossary, `http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html`.

Chapter 2
Configuration information for different versions of the ARM compilation tools

The following topics summarize the FLEX*net*, GCC, and Cygwin versions supported by the different versions of the ARM compilation tools:

2.1 FLEX*net* versions used the in the compilation tools

The FLEX*net* versions in the compilation tools are:

Table 2-1 FLEX*net* versions

Compilation tools version	Windows	Linux
ARM Compiler toolchain 5.02	10.8.10.0	10.8.10.0
ARM Compiler toolchain 5.01	10.8.10.0	10.8.10.0
ARM Compiler toolchain 5.0	10.8.7.0	10.8.7.0
ARM Compiler toolchain 4.1	10.8.7.0	10.8.7.0
RVCT 4.0 build 471	10.8.7.0	10.8.7.0
RVCT 4.0	10.8.5.0	9.2
RVCT 3.1 build 836	10.8.7.0	10.8.7.0
RVCT 3.1 build 739	10.8.5.0	10.8.5.0
RVCT 3.1	10.8.5.0	9.2
RVCT 3.0	10.8.5.0	10.8.0
RVCT 2.2	9.0.0	9.0.0
RVCT 2.1	9.0.0	9.0.0
RVCT 2.0	8.1b	8.1b
ADS 1.2	7.2i	7.2i

2.1.1 See also

Other information

- *ARM® DS-5™ License Management Guide*, http://infocenter.arm.com/help/topic/com.arm.doc.dui0577-/index.html.

2.2 GCC versions emulated

The GCC versions emulated in the compilation tools are:

Table 2-2 GCC versions

Compilation tools version	GCC version
ARM Compiler toolchain 5.02	4.2.0
ARM Compiler toolchain 5.01	4.2.0
ARM Compiler toolchain 5.0	4.2.0
ARM Compiler toolchain 4.1	4.2.0
RVCT 4.0	4.2.0
RVCT 3.1	3.3.0

2.2.1 See also

Reference

- Chapter 7 *Migrating from RVCT v4.0 to ARM Compiler v4.1*
- Chapter 8 *Migrating from RVCT v3.1 to RVCT v4.0*.

Compiler Reference:

- *--gnu_version=version* on page 3-110.

2.3 Cygwin versions supported

ARM Compiler toolchain v5.0 and later supports Cygwin 1.7.7-1.

2.3.1 See also

Concepts

Introducing the ARM Compiler toolchain:

- *Specifying Cygwin paths in compilation tools on Windows* on page 2-29.

Chapter 3
Migrating from ARM Compiler v5.0 to v5.01 and later

The following topic describes the changes that affect migration and compatibility between ARM Compiler v5.0 and v5.01 and later:

3.1 General changes between ARM Compiler v5.0 and v5.01 and later

The following changes have been made in ARM Compiler toolcahin v5.01:

- The version-specific environment variables have changed to use a single digit version, for example, `ARMCC5INC`.
- The version number reported by the tools using `--version_number` has changed:
  - In version 5.0 and earlier, the format is `VVbbbb`.
  - In version 5.01 and later, the format is `VVVbbbb`.

  For example, version 5.01 build 2345 is reported as `5012345`.

3.1.1 See also

Reference

Introducing the ARM® Compiler toolchain:

- *Toolchain environment variables* on page 2-14

Assembler Reference:

- *--version_number* on page 2-84

Compiler Reference:

- *--version_number* on page 3-214

Linker Reference:

- *--version_number* on page 2-185

Using the fromelf Image Converter:

- *--version_number* on page 4-75

Creating Static Software Libraries with armar:

- *--version_number* on page 6-34.

3.2 Documentation changes between ARM Compiler v5.0 and v5.01 and later

For technical changes that have been made to the ARM Compiler toolchain documentation, see the following revision summaries:

- Appendix A *Revisions for Migration and Compatibility* (this document)
- Appendix A *Revisions for Introducing the ARM Compiler toolchain*
- Appendix A *Revisions for Developing Software for ARM Processors*
- Appendix A *Revisions for Using the Assembler*
- Appendix A *Revisions for Using the Compiler*
- Appendix A *Revisions for Using the Linker*
- Appendix A *Revisions for Using ARM C and C++ Libraries and Floating-Point Support*
- Appendix A *Revisions for Assembler Reference*
- Appendix H *Revisions for the Compiler Reference*
- Appendix A *Revisions for the Linker Reference*
- Appendix A *Revisions for the ARM C and C++ Libraries and Floating-Point Support Reference*
- Appendix A *Revisions for Using the fromelf Image Converter*
- Appendix A *Revisions for Building Linux Applications with the ARM Compiler toolchain and GNU Libraries*
- Appendix A *Revisions for the Errors and Warnings Reference*.

Chapter 4
Migrating from ARM Compiler v4.1 Patch 3 to v5.0

The following topic describes the changes that affect migration and compatibility between ARM Compiler v4.1 Patch 3 and v5.0:

4.1 General changes between ARM Compiler v4.1 Patch 3 and v5.0

The following changes have been made in ARM Compiler toolcahin v5.0:

- The tools no longer require any environment variables to be set.
- An additional convention for finding the default header and library directories has been added to the v5.0 tools. When no environment variables or related command-line options are present:
  - the compiler looks in `../include`.
  - the linker looks in `../lib`.

  These locations match the relative paths to the include and library directories from the DS-5 `bin` directory.

4.1.1 See also

Reference

Introducing the ARM Compiler toolchain:

- *Toolchain environment variables* on page 2-14

4.2 Compiler changes between ARM Compiler v4.1 Patch 3 and v5.0

The following changes have been made to the compiler:

- The *Edison Design Group* (EDG) front-end used by the compiler has been updated to version 4.1. However, this does not create any compatibility issues.
- If `ARMCC50INC` is not set and `-J` is not present on the command line, the compiler searches for the default includes in `../include`, relative to the location of `armcc.exe`.
- Improved GCC compatibility, and supports a GCC fallback mode.
- The *Link-time code generation* (LTCG) feature is deprecated. As an alternative ARM recommends you use the `--multifile` compiler option.
- Profiler-guided optimization with `--profile` is deprecated, and is not currently compatible with ARM Streamline.

4.2.1 See also

Concepts

Using the Compiler:

- *Using GCC fallback when building applications on page 3-24*.

Reference

Compiler Reference:

- *-Jdir[,dir,...]* on page 3-125
- *--multifile, --no_multifile* on page 3-150
- *-Warmcc,--gcc_fallback* on page 3-222
- *Predefined macros* on page 5-183.

Introducing the ARM Compiler toolchain:

- *Toolchain environment variables* on page 2-14

4.3 Linker changes between ARM Compiler v4.1 Patch 3 and v5.0

The following changes have been made to the linker:

- If `ARMCC50LIB` is not set and `--libpath` is not present on the command line, the linker searches for the default libraries in `../lib`, relative to the location of `armlink.exe`.
- The *Link-time code generation* (LTCG) feature is deprecated. As an alternative ARM recommends you use the `--multifile` compiler option.
- Profiler-guided optimization with `--profile` is deprecated, and is not currently compatible with ARM Streamline.

4.3.1 See also

Reference

Linker Reference:

- *--libpath=pathlist* on page 2-96

Compiler Reference:

- *--multifile, --no_multifile* on page 3-150.

Introducing the ARM Compiler toolchain:

- *Toolchain environment variables* on page 2-14.

4.4 Documentation changes between ARM Compiler v4.1 Patch 3 and v5.0

For technical changes that have been made to the ARM Compiler toolchain documentation, see the following revision summaries:

- Appendix A *Revisions for Migration and Compatibility* (this document)
- Appendix A *Revisions for Introducing the ARM Compiler toolchain*
- Appendix A *Revisions for Developing Software for ARM Processors*
- Appendix A *Revisions for Using the Assembler*
- Appendix A *Revisions for Using the Compiler*
- Appendix A *Revisions for Using the Linker*
- Appendix A *Revisions for Using ARM C and C++ Libraries and Floating-Point Support*
- Appendix A *Revisions for Assembler Reference*
- Appendix H *Revisions for the Compiler Reference*
- Appendix A *Revisions for the Linker Reference*
- Appendix A *Revisions for the ARM C and C++ Libraries and Floating-Point Support Reference*
- Appendix A *Revisions for Using the fromelf Image Converter*
- Appendix A *Revisions for Building Linux Applications with the ARM Compiler toolchain and GNU Libraries*
- Appendix A *Revisions for the Errors and Warnings Reference*.

Chapter 5
Migrating from ARM Compiler v4.1 SP1 to v4.1 Patch 3

The following topic describes the changes that affect migration and compatibility between ARM Compiler v4.1 SP1and v4.1 Patch 3:

- *C and C++ library changes between ARM Compiler v4.1 SP1 and v4.1 Patch 3* on page 5-2.

5.1 C and C++ library changes between ARM Compiler v4.1 SP1 and v4.1 Patch 3

The new implementation of `alloca()` allocates memory on the stack and not on the heap. This does not cause compatibility problems with software that assumes the old heap-based implementation of `alloca()`. However, such softwre might contain operations that are no longer required, such as implementing `__user_perthread_libspace` for the `alloca` state that is no longer used.

5.1.1 See also

Concepts

Using ARM C and C++ Libraries and Floating-Point Support:

- *Library heap usage requirements of the ARM C and C++ libraries on page 2-8*
- *Use of the __user_libspace static data area by the C libraries on page 2-21*
- *Building an application without the C library on page 2-41*.

Chapter 6
Migrating from ARM Compiler v4.1 to v4.1 SP1

The following topics describe the changes that affect migration and compatibility between ARM Compiler v4.1and v4.1 SP1:

6.1 Compiler changes between ARM Compiler v4.1 and v4.1 SP1

The compiler faults use of the `at` attribute when it is used on declarations with incomplete non-array types. For example, if `foo` is not declared, the following causes an error:

```
struct foo a __attribute__((at(0x16000)));
```

6.1.1 See also

Concepts

- Chapter 6 *Migrating from ARM Compiler v4.1 to v4.1 SP1*.

Reference

Compiler Reference:

- *__attribute__((at(address))) variable attribute* on page 5-72.

6.2 Linker changes between ARM Compiler v4.1 and v4.1 SP1

There are no technical changes to `armlink` that affect migration between ARM Compiler v4.1 and v4.1 SP1.

6.2.1 See also

Concepts

- Chapter 6 *Migrating from ARM Compiler v4.1 to v4.1 SP1*.

6.3 Assembler changes between ARM Compiler v4.1 and v4.1 SP1

There are no technical changes to `armasm` that affect migration between ARM Compiler v4.1 and v4.1 SP1.

6.3.1 See also

Concepts

- Chapter 6 *Migrating from ARM Compiler v4.1 to v4.1 SP1*.

6.4 C and C++ library changes between ARM Compiler v4.1 and v4.1 SP1

The symbol `__use_accurate_range_reduction` is retained for backward compatibility, but no longer has any effect.

The C99 complex number functions in the previous hardware floating point version of the library only had the *hardfp* linkage functions and not the *softfp* linkage functions. The new library has both the *hardfp* linkage and *softfp* linkage functions. This means that existing object code that was built to use hardware floating point might not function correctly when calling complex functions from the library. The linker issues a warning in this case. You must recompile all the code that might use the affected functions and that was built to use hardware floating point. You must relink them with the new library.

6.4.1 See also

Concepts

Using ARM® C and C++ Libraries and Floating-Point Support:

- Chapter 2 *The ARM C and C++ libraries*.

Reference

ARM® C and C++ Libraries and Floating-Point Support Reference:

- Chapter 2 *The C and C++ libraries*.

Compiler Reference:

- *--apcs=qualifer...qualifier* on page 3-11
- *--fpu=name* on page 3-100.

6.5 fromelf changes between ARM Compiler v4.1 and v4.1 SP1

`fromelf` can now process all files, or a subset of files, in an archive.

6.5.1 See also

Reference

Using the fromelf Image Converter:

- *input_file* on page 4-48.

6.6 Documentation changes between ARM Compiler v4.1 and v4.1 SP1

For technical changes that have been made to the ARM Compiler toolchain documentation, see the following revision summaries:

- Appendix A *Revisions for Migration and Compatibility* (this document)
- Appendix A *Revisions for Introducing the ARM Compiler toolchain*
- Appendix A *Revisions for Developing Software for ARM Processors*
- Appendix A *Revisions for Using the Assembler*
- Appendix A *Revisions for Using the Linker*
- Appendix A *Revisions for Using ARM C and C++ Libraries and Floating-Point Support*
- Appendix A *Revisions for Assembler Reference*
- Appendix H *Revisions for the Compiler Reference*
- Appendix A *Revisions for the Linker Reference*
- Appendix A *Revisions for the ARM C and C++ Libraries and Floating-Point Support Reference*
- Appendix A *Revisions for Using the fromelf Image Converter*
- Appendix A *Revisions for the Errors and Warnings Reference*.

Chapter 7
Migrating from RVCT v4.0 to ARM Compiler v4.1

The following topics describe the changes that affect migration and compatibility between RVCT v4.0 and ARM Compiler v4.1:

7.1 General changes between RVCT v4.0 and ARM Compiler v4.1

The convention for naming environment variables, such as those for setting default header and library directories, has changed. These are now prefixed with `ARMCC` rather than `RVCT`. For example, `ARMCC41INC` rather than `RVCT40INC`.

7.1.1 Compatibility of ARM Compiler v4.1 with legacy objects and libraries

Backwards compatibility of RVCT v2.x, v3.x, and v4.0 to the objects and libaries is supported provided you have not built them with `--apcs /adsabi`.

Given these restrictions, ARM strongly recommends that you rebuild your entire project, including any user, or third-party supplied libraries, with ARM Compiler v4.1. This is to avoid any potential incompatibilities, and to take full advantage of the improved optimization, enhancements, and new features provided by ARM Compiler v4.1.

7.1.2 See also

Reference

Introducing the ARM® Compiler toolchain:

- *Toolchain environment variables* on page 2-14.

7.2 Compiler changes between RVCT v4.0 and ARM Compiler v4.1

Sign rules on enumerators has changed in line with convention. Enumerator container is now unsigned unless a negative constant is defined. The RVCT v4.0 10Q1 patch made this change in GCC mode only.

`-O3` no longer implies `--multifile`. The `--multifile` option has always been available as a separate option and it is recommended you put this into your builds.

7.2.1 See also

Concepts

- *General changes between RVCT v4.0 and ARM Compiler v4.1* on page 7-2.

Reference

Compiler Reference:

- *--multifile, --no_multifile* on page 3-150
- *-Onum* on page 3-156.

7.3 Linker changes between RVCT v4.0 and ARM Compiler v4.1

The following changes to the linker have been made:

GNU ld script support

`armlink` v4.1 supports a subset of GNU linker control scripts. To more closely match the behavior of GNU ld, `armlink` uses an internal linker control script when you specify the `--sysv` command-line option. In previous versions of `armlink` an internal scatter file was used.

The use of a control script produces a logically equivalent, but physically different layout to RVCT v4.0. To revert back to the default scatter file layout use the command-line option `--no_use_sysv_default_script`.

You can replace the internal control script with a user-defined control script using the `-T` option.

ARM/Thumb synonyms removed

Support for the deprecated feature ARM/Thumb Synonyms has been removed in 4.1. The ARM/Thumb synonyms feature permits an ARM Global Symbol Definition of symbol `S` and a Thumb Global Symbol Definition of `S` to coexist. All branches from ARM state are directed at the ARM Definition, all branches from Thumb state are directed at the Thumb Definition.

In 4.0 armlink gives a deprecated feature warning `L6455E` when it detects ARM/Thumb Synonyms.

In 4.1 armlink gives an error message `L6822E` when it detects ARM/Thumb Synonyms.

To recreate the behavior of synonyms rename both the ARM and Thumb definitions and relink. For each undefined symbol you have to point it at the ARM or the Thumb synonym.

7.3.1 See also

Concepts

- *General changes between RVCT v4.0 and ARM Compiler v4.1* on page 7-2.

Reference

Linker Reference:

- *--use_sysv_default_script, --no_use_sysv_default_script* on page 2-178
- *--sysv* on page 2-170.

7.4 Assembler changes between RVCT v4.0 and ARM Compiler v4.1

The following changes to the assembler have been made:

Change to the way the assembler reads and processes files

Older assemblers sometimes permitted the source file being assembled to vary between the two passes of the assembler. In the following example, the symbol `num` is defined in the second pass because the symbol `foo` is not defined in the first pass.

```
        AREA x,CODE
        [ :DEF: foo
num     EQU 42
        ]
foo     DCD num
        END
```

The way the assembler reads and processes the file has now changed, and is stricter. You must rewrite code such as this to ensure that the path through the file is the same in both passes.

The following code shows another example where the assembler faults:

```
        AREA FOO, CODE
      IF :DEF: VAR
      ELSE
VAR   EQU 0
      EDNIF
        END
```

The resulting error is:

```
Error: A1903E : Line not seen in first pass; cannot be assembled.
```

To avoid this error, you must rewrite this code as:

```
        AREA FOO, CODE
      IF :LNOT: :DEF: VARVAR    EQU 0
      EDNIF
        END
```

Change to messages output by the assembler

Generally, any messages referring to a position on the source line now has a caret character pointing to the offending part of the source line, for example:

```
"foo.s", line 3 (column 19): Warning: A1865W: '#' not seen before constant
expression
    3 00000000         ADD r0,r1,1
                                 ^
```

Changes to diagnostic messages

Various instructions in ARM (using SP) were deprecated when 32-bit Thumb instructions were introduced. These instructions are no longer diagnosed as deprecated unless assembling for a CPU that has 32-bit Thumb instructions. To enable the warnings on earlier CPUs, you can use the option `--diag_warning=1745,1786,1788,1789,1892`. This change was introduced in the RVCT v4.0 09Q4 patch.

Obsolete command-line option

The `-O` command-line option is obsolete. Use `-o` instead.

7.4.1 See also

Concepts

- *General changes between RVCT v4.0 and ARM Compiler v4.1* on page 7-2.

Reference

Assembler Reference:

- *--diag_warning=tag{, tag}* on page 2-29
- *-o filename* on page 2-67.

Using the Assembler:

- *How the assembler works* on page 2-4
- *2 pass assembler diagnostics* on page 7-21.

7.5 C and C++ library changes between RVCT v4.0 and ARM Compiler v4.1

The libraries now use more 32-bit Thumb code on targets that support 32-bit Thumb. This is expected to result in reduced code size without affecting performance. The linker option `--no_thumb2_library` falls back to the old-style libraries if necessary.

Math function returns in some corner cases now conform to POSIX/C99 requirements. You can enable older behavior with:

```
#pragma import __use_rvct_matherr
```

From RVCT v4.0 09Q4 patch onwards, you can enable the newer behavior with:

```
#pragma import __use_c99_matherr.
```

7.5.1 See also

Concepts

- *General changes between RVCT v4.0 and ARM Compiler v4.1* on page 7-2.

Concepts

Using ARM® C and C++ Libraries and Floating-Point Support:

- *How the ARM C library fulfills ISO C specification requirements on page 2-107*.

Reference

Linker Reference:

- *--thumb2_library, --no_thumb2_library* on page 2-172.

Chapter 8
Migrating from RVCT v3.1 to RVCT v4.0

The following topics describe the changes that affect migration and compatibility between RVCT v3.1 and RVCT v4.0:

8.1 Default `--gnu_version` changed from `303000` (GCC 3.3) to `402000` (GCC 4.2)

This affects which GNU extensions are accepted, such as `__attribute__((visibility(...)))` and lvalue casts, even in non-GNU modes.

8.1.1 See also

Reference

Compiler Reference:

- *--gnu_version=version* on page 3-110.

8.2 General changes between RVCT v3.1 and RVCT v4.0

The following changes affect multiple tools:

8.2.1 Restrictions on --fpu

`--fpu=VFPv2` or `--fpu=VFPv3` are only accepted if CPU architecture is greater than or equal to ARMv5TE. This affects all tools that accept `--fpu`.

Note

The assembler assembles VFP instructions when you use the `--unsafe` option, so do not use `--fpu` when using `--unsafe`. If you use `--fpu` with `--unsafe`, the assembler downgrades the reported architecture error to a warning.

8.2.2 Remove support for v5TExP and derivatives, and all ARMv5 architectures without T

The following `--cpu` choices are obsolete and have been removed:

- 5
- 5E
- 5ExP
- 5EJ
- 5EWMMX2
- 5EWMMX
- 5TEx
- ARM9E-S-rev0
- ARM946E-S-rev0
- ARM966E-S-rev0.

8.2.3 Compatibility of RVCT v4.0 with legacy objects and libraries

Backwards compatibility of RVCT v2.x, v3.x and v4.0 object and library code is supported provided you have not built them with `--apcs /adsabi` and use the RVCT v4.0 linker and C/C++ libraries. Forward compatibility is not guaranteed.

Given these restrictions, ARM strongly recommends that you rebuild your entire project, including any user, or third-party supplied libraries, with RVCT v4.0 and later. This is to avoid any potential incompatibilities, and to take full advantage of the improved optimization, enhancements, and new features provided by RVCT v4.0 and later.

8.2.4 See also

Reference

Compiler Reference:

- *--apcs=qualifer...qualifier* on page 3-11
- *--cpu=name* on page 3-49
- *--fpu=name* on page 3-100.

Linker Reference:

- *--cpu=name* on page 2-38
- *--fpu=name* on page 2-76.

Assembler Reference:

- *--apcs=qualifier...qualifier* on page 2-7

- *--cpu=name* on page 2-19
- *--fpu=name* on page 2-39
- *--unsafe* on page 2-82.

8.3 Changes to symbol visibility between RVCT v3.1 and RVCT v4.0

The following changes to symbol visibility have been made:

8.3.1 Change to ELF visibility used to represent `__declspec(dllexport)`

When using the `--hide_all` compiler command-line option, which is the default, the ELF visibility used to represent `__declspec(dllexport)` in RVCT v3.1 and earlier was STV_DEFAULT. In RVCT v4.0 it is STV_PROTECTED. Symbols that are STV_PROTECTED can be referred to by other DLLs but cannot be preempted at load-time.

When using the `--no_hide_all` command-line option, the visibility of imported and exported symbols is still STV_DEFAULT as it was in RVCT v3.1.

8.3.2 `__attribute(visibility(...))`

The GNU-style `__attribute(visibility(...))` has been added and is available even without specifying the `--gnu` compiler command-line option. Using it overrides any implicit visibility. For example, the following results in STV_DEFAULT visibility instead of STV_HIDDEN:

```
__declspec(visibilty("default")) int x = 42;
```

8.3.3 RVCT v3.1 symbol visibility summary

The following tables summarize the visibility rules in RVCT v3.1:

Table 8-1 RVCT v3.1 symbol visibility summary

Code	`--hide_all` **(default)**	`--no_hide_all`	`--dllexport_all`
`extern int x;` `extern int g(void);`	STV_HIDDEN	STV_DEFAULT	STV_HIDDEN
`extern int y = 42;` `extern int f() { return g() + x; }`	STV_HIDDEN	STV_DEFAULT	STV_DEFAULT
`__declspec(dllimport) extern int imx;` `__declspec(dllimport) extern int img(void);`	STV_DEFAULT	STV_DEFAULT	STV_DEFAULT

`__declspec(dllexport) extern int exy = 42;`
`__declspec(dllexport) extern int exf() {`
`return img() + imx; }`
	STV_DEFAULT	STV_DEFAULT	STV_DEFAULT

`/* exporting undefs (unusual?) */`
`__declspec(dllexport) extern int exz;`
`__declspec(dllexport) extern int exh(void);`
	STV_HIDDEN	STV_HIDDEN	STV_HIDDEN

Table 8-2 RVCT v3.1 symbol visibility summary for references to run-time functions

Code	`--no_dllimport_runtime` `--hide_all` **(default)**	`--no_hide_all`	`--dllexport_all`

`/* references to runtime functions, for`
`example __aeabi_fmul */`
`float fn(float a, float b) { return a*b; }`
	STV_HIDDEN	STV_DEFAULT	STV_DEFAULT

8.3.4 RVCT v4.0 symbol visibility summary

The following tables summarize the visibility rules in RVCT v4.0:

Table 8-3 RVCT v4.0 symbol visibility summary

Code	`--hide_all` **(default)**	`--no_hide_all`	`--dllexport_all`
`extern int x;` `extern int g(void);`	STV_HIDDEN	STV_DEFAULT	STV_HIDDEN
`extern int y = 42;` `extern int f() { return g() + x; }`	STV_HIDDEN	STV_DEFAULT	STV_PROTECTED
`__declspec(dllimport) extern int imx;` `__declspec(dllimport) extern int img(void);`	STV_DEFAULT	STV_DEFAULT	STV_DEFAULT

`__declspec(dllexport) extern int exy = 42;`
`__declspec(dllexport) extern int exf() {`
`return img() + imx; }`
	STV_PROTECTED	STV_PROTECTED	STV_PROTECTED

`/* exporting undefs (unusual?) */`
`__declspec(dllexport) extern int exz;`
`__declspec(dllexport) extern int exh(void);`
	STV_PROTECTED	STV_PROTECTED	STV_PROTECTED

Table 8-4 RVCT v4.0 symbol visibility summary for references to run-time functions

Code	`--no_dllimport_runtime` `--hide_all` **(default)**	`--no_hide_all`	`--dllexport_all`
`/* references to runtime functions, for example __aeabi_fmul */` `float fn(float a, float b) { return a*b; }`	STV_HIDDEN	STV_DEFAULT	STV_DEFAULT

8.3.5 See also

Concepts

- *General changes between RVCT v3.1 and RVCT v4.0* on page 8-3.

Reference

Compiler Reference:

- *--default_definition_visibility=visibility* on page 3-58
- *--dllexport_all, --no_dllexport_all* on page 3-77
- *--dllimport_runtime, --no_dllimport_runtime* on page 3-78
- *--gnu* on page 3-107
- *--hide_all, --no_hide_all* on page 3-113.

8.4 Compiler changes between RVCT v3.1 and RVCT v4.0

The following compiler changes have been made:

8.4.1 Single compiler executable

The executables `tcc`, `armcpp` and `tcpp` are no longer delivered.

To compile for Thumb, use the `--thumb` command-line option.

To compile for C++, use the `--cpp` command-line option.

Note

The compiler automatically selects C++ for files with the `.cpp` extension, as before.

8.4.2 Vectorizing compiler

The NEON vectorizing compiler is provided as standard functionality, and is no longer provided as a separate add-on. A license to use the NEON vectorizing compiler is provided with the Professional Edition of the ARM development tools.

8.4.3 VAST Changes

VAST has been upgraded through two versions (VAST 11 for 4.0 Alpha and 4.0 Alpha2 and later). Apart from the following issue, you do not have to make any changes to your v3.1 builds to use the new VAST.

RVCT v3.1 reassociated saturating ALU operations. This meant programs like the following could produce different results with `--vectorize` and `--no_vectorize`:

```
int g_448464(short *a, short *b, int n)
{
    int i; short s = 0;
    for (i = 0; i < n; i++)  s = L_mac(s, a[i], b[i]);
    return s;
}
```

In RVCT v4.0, you might see a performance degradation because of this issue.

The `--reassociate_saturation` and `--no_reassociate_saturation` command-line options have been added to permit reassociation to occur.

8.4.4 See also

Concepts

- *General changes between RVCT v3.1 and RVCT v4.0* on page 8-3.

Reference

Compiler Reference:

- *--cpp* on page 3-47
- *--reassociate_saturation, --no_reassociate_saturation* on page 3-177
- *--thumb* on page 3-197
- *--vectorize, --no_vectorize* on page 3-213.

8.5 Linker changes between RVCT v3.1 and RVCT v4.0

The following linker changes have been made:

8.5.1 Helper functions

Before RVCT v4.0, helper functions were implemented in helper library files such as `h_tf.l`, provided with the ARM compiler. All helper library filenames start with `h_`. RVCT v4.0 no longer requires helper libraries. Instead, all helper functions are generated by the compiler and become part of an object file.

Placing ARM library helper functions with scatter files

In RVCT v3.1 and earlier, the helper functions reside in libraries provided with the ARM vompiler. Therefore, it was possible to use `armlib` and `cpplib` in a scatter file to inform the linker where to place these helper functions in memory.

Because Iin RVCT v4.0 and later, the helper functions are generated by the compiler in the resulting object files. They are no longer in the standard C libraries, so it is no longer possible to use `armlib` and `cpplib` in a scatter file. Instead, place the helper functions using `*.* (i.__ARM_*)` in your scatter file.

Warning L6932W when linking with helper libraries

When using RVCT v4.0 and later, you might see the following linker warning:

```
Warning: L6932W: Library reports warning: use of helper library h_xx.l is deprecated
```

Before RVCT v4.0, one reason for linking with a helper library is if an object file references the helper function `__ARM_switch8`. You can find this out by looking at verbose output from the linker (using the armlink `--verbose` option), for example:

```
Loading member object1.o from lib1.a.
    reference :  strncmp
    reference :  __ARM_switch8
```

In RVCT v4.0 and later, because helper libraries are no longer required, the verbose output from the linker might show, for example:

```
Loading member object2.o from lib2.a.
    ...
    definition:  __ARM_common_switch8_thumb
```

In this case, the helper function `__ARM_common_switch8_thumb` is in the object file `object2.o`, rather than in a helper library.

So, if you are using RVCT v4.0 and the linker is generating warning message L6932W, it is likely that you are linking with objects or libraries that were built with RVCT v3.1, not RVCT v4.0.

8.5.2 Linker steering files and symbol visibility

In RVCT v3.1 the visibility of a symbol was overridden by the steering file or `.directive` commands `IMPORT` and `EXPORT`. When this occurred the linker issued a warning message, for example:

```
Warning: L6780W: STV_HIDDEN visibility removed from symbol hidden_symbol through EXPORT.
```

In RVCT v4.0 the steering file mechanism respects the visibility of the symbol, so an IMPORT or EXPORT of a STV_HIDDEN symbol is ignored. You can restore the v3.1 behavior with the --override_visibility command-line option.

8.5.3 Linker-defined symbols

In the majority of cases region related symbols behave identically to v3.1.

Section-relative symbols

The execution region Base and Limit symbols are now section-relative. There is no sensible section for a $$Length symbol so this remains absolute.

This means that the linker-defined symbol is assigned to the most appropriate section in the execution region. The following example shows this:

```
ExecRegion ER

RO Section 1 ; Image$$ER$$Base and Image$$ER$$RO$$Base, val 0
RO Section 2 ; Image$$ER$$RO$$Limit, val Limit(RO Section 2)

RW Section 1 ; Image$$ER$$RW$$Base, val 0
RW Section 2 ; Image$$ER$$Limit and Image$$ER$$RW$$Limit,
               val Limit(RW Section 2)

ZI Section 1 ; Image$$ER$$ZI$$Base, val 0
ZI Section 2 ; Image$$ER$$ZI$$Limit, val Limit(ZI Section 2)
```

In each case the value of the ...$$Length symbol is the value of the ...$$Limit symbol minus the ...$$Base symbol.

If there is no appropriate section that exists in the execution region then the linker defines a zero-sized section of the appropriate type to hold the symbols.

Impact of the change

The change to section-relative symbols removes several special cases from the linker implementation, that might improve reliability. It also means that dynamic relocations work naturally on SysV and BPABI links.

Alignment

The ...$$Limit symbols are no longer guaranteed to be four-byte aligned because the limit of the section it is defined in might not be aligned to a four-byte boundary.

This might affect you if you have code that accidentally relies on the symbol values being aligned. If you require an aligned $$Limit or $$Length then you must align the symbol value yourself.

For example, the following legacy initialization code might fail if Image$$<Region_Name>$$Length is not word aligned:

```
    LDR R1,|Load$$region_name$$Base|
    LDR R0,|Image$$region_name$$Base|
    LDR R4,|Image$$region_name$$Length|

    ADD R4, R4, R0
copy_rw_data
    LDRNE  R3,[R1],#4
    STRNE  R3,[R0],#4
    CMP  R0,R4
    BNE  copy_rw_data
```

Writing your own initialization code is not recommended, because system initialization is more complex than in earlier toolchain releases. ARM recommends that you use the `__main` code provided with the ARM Compiler toolchain.

Delayed Relocations

The linker has introduced an extra address assignment and relocation pass after RW compression. This permits more information about load addresses to be used in linker-defined symbols.

Be aware that:

- `Load$$`*`region_name`*`$$Base` is the address of *`region_name`* prior to C-library initialization
- `Load$$`*`region_name`*`$$Limit` is the limit of *`region_name`* prior to C-library initialization
- `Image$$`*`region_name`*`$$Base` is the address of *`region_name`* after C-library initialization
- `Image$$`*`region_name`*`$$Limit` is the limit of *`region_name`* after C-library initialization.

Load Region Symbols have the following properties:

- They are ABSOLUTE because section-relative symbols can only have Execution addresses.
- They take into account RW compression
- They do not include ZI because it does not exist prior to C-library initialization.

In addition to `Load$$$$Base`, the linker now supports the following linker-defined symbols:

`Load$$`*`region_name`*`$$BaseLoad$$`*`region_name`*`$$LimitLoad$$`*`region_name`*`$$LengthLoad$$`*`region_name`*`$$RO$$BaseLoad$$`*`region_name`*`$$RO$$LimitLoad$$`*`region_name`*`$$RO$$LengthLoad$$`*`region_name`*`$$RW$$BaseLoad$$`*`region_name`*`$$RW$$LimitLoad$$`*`region_name`*`$$RW$$Length`

Limits of Delayed Relocation

All relocations from RW compressed execution regions must be performed prior to compression because the linker cannot resolve a delayed relocation on compressed data.

If the linker detects a relocation from a RW-compressed region `REGION` to a linker-defined symbol that depends on RW compression then the linker disables compression for `REGION`.

Load Region Symbols

RVCT v4.0 now permits linker-defined symbols for load regions. They follow the same principle as the `Load$$` symbols for execution regions. Because a load region might contain many execution regions it is not always possible to define the `$$RO` and `$$RW` components. Therefore, load region symbols only describe the region as a whole.

```
Load$$LR$$Load_Region_Name$$Base ; Base address of <Load Region Name>
Load$$LR$$Load_Region_Name$$Limit ; Load Address of last byte of content
                                  ; in Load
region.Load$$LR$$Load_Region_Name$$Length ; Limit - Base
```

Image-related symbols

The RVCT v4.0 linker implements these in the same way as the execution region-related symbols.

They are defined only when scatter files are not used. This means that they are available for the --sysv and --bpabi linking models.

```
Image$$RO$$Base  ; Equivalent to Image$$ER_RO$$BaseImage$$RO$$Limit ;
Equivalent to Image$$ER_RO$$LimitImage$$RW$$Base  ; Equivalent to
Image$$ER_RW$$BaseImage$$RW$$Limit ; Equivalent to
Image$$ER_RW$$LimitImage$$ZI$$Base  ; Equivalent to
Image$$ER_ZI$$BaseImage$$ZI$$Limit ; Equivalent to Image$$ER_ZI$$Limit
```

Interaction with ZEROPAD

An execution region with the ZEROPAD keyword writes all ZI data into the file:

- Image$$ symbols define execution addresses post initialization.

  In this case, it does not matter that the zero bytes are in the file or generated. So for Image$$ symbols, ZEROPAD does not affect the values of the linker-defined symbols.

- Load$$ symbols define load addresses pre initialization.

  In this case, any zero bytes written to the file are visible, Therefore, the Limit and Length take into account the zero bytes written into the file.

8.5.4 Build attributes

The RVCT v4.0 linker fully supports reading and writing of the ABI Build Attributes section. The linker can now check more properties such as wchar_t and enum size. This might result in the linker diagnosing errors in old objects that might have inconsistencies in the Build Attributes. Most of the Build Attributes messages can be downgraded to permit armlink to continue.

The --cpu option now checks the FPU attributes if the CPU chosen has a built-in FPU. For example, --cpu=cortex-a8 implies --fpu=vfpv3. In RVCT v3.1 the --cpu option only checked the build attributes of the chosen CPU.

The error message L6463E: Input Objects contain *archtype* instructions but could not find valid target for *archtype* architecture based on object attributes. Suggest using --cpu option to select a specific cpu. is given in one of two situations:

- the ELF file contains instructions from architecture *archtype* yet the Build Attributes claim that *archtype* is not supported
- the Build Attributes are inconsistent enough that the linker cannot map them to an existing CPU.

If setting the --cpu option still fails, the option --force_explicit_attr causes the linker to retry the CPU mapping using Build Attributes constructed from --cpu=*archtype*. This might help if the Error is being given solely because of inconsistent Build Attributes.

8.5.5 C library initialization

A change to the linker when dealing with C library initialization code causes specially named sections in the linker map file created with the --map command-line option. You can ignore these specially named sections.

8.5.6 ARM Linux

When building a shared object the linker automatically imports any reference with STV_DEFAULT visibility that is undefined. This matches the behavior of GCC. This might result in some failed links now being successful.

Prelink support reserves some extra space, this results in slightly larger images and shared objects. The prelink support can be turned off with `--no_prelink_support`.

There have been numerous small changes with regards to symbol visibility, these are described in the Symbol visibility changes.

8.5.7 RW compression

Some error handling code is run later so that information from RW compression can be used. In almost all cases, this means more customer programs are able to link. There is one case where RVCT v4.0 has removed a special case so that it could diagnose more RW compression errors.

Multiple in-place execution regions with RW compression are no longer a special case. It used to be possible to write:

```
LR1 0x0
{
    ER1 +0 { file1.o(+RW) }
    ER2 +0 { file2.o(+RW) }
}
```

This is no longer possible under v4.0 and the linker gives an error message that ER1 decompresses over ER2. This change has been made to permit the linker to diagnose:

```
LR1 0x0
{
    ER1 +0 { file1.o(+RW) }
    ER2 +0 { file2.o(+RO) }   ; NOTE RO not RW
}
```

This fails at runtime on RVCT v3.1.

8.5.8 See also

Concepts

- *General changes between RVCT v3.1 and RVCT v4.0* on page 8-3.

Using the Linker:

- *Optimization with RW data compression* on page 5-13
- *Accessing linker-defined symbols* on page 7-4
- *Region-related symbols* on page 7-5
- *Image$$ execution region symbols* on page 7-6
- *Load$$ execution region symbols* on page 7-7
- *Importing linker-defined symbols in C and C++* on page 7-12
- *Section-related symbols* on page 7-14
- *Example of placing ARM library helper functions* on page 8-53.

Using ARM C and C++ Libraries and Floating-Point Support:

- *Initialization of the execution environment and execution of the application on page 2-55*.

Reference

Linker Reference:

- *--bpabi* on page 2-24
- *--cpu=name* on page 2-38
- *--force_explicit_attr* on page 2-72
- *--fpu=name* on page 2-76
- *--map, --no_map* on page 2-108

- *--override_visibility* on page 2-115
- *--prelink_support, --no_prelink_support* on page 2-125
- *--sysv* on page 2-170
- *--verbose* on page 2-184
- *EXPORT* on page 3-2
- *IMPORT* on page 3-4
- *Execution region attributes* on page 4-12.

8.6 Assembler changes between RVCT v3.1 and RVCT v4.0

The following changes to the assembler have been made:

- The `-O` command-line option is deprecated. `-O` is a synonym for `-o` to output to a named file. This has been deprecated to avoid user confusion with the `armcc` option with the same name.
- The `-D` command-line option is obsolete. Use `--depend` instead.
- `LDM r0!,{r0-r4}` no longer ignores writeback. Previously in Thumb, `LDM r0!,{r0-r4}` assembled with a warning to the 16-bit `LDM` instruction and no writeback was performed. Because the syntax requests writeback, and this encoding is only available in 32-bit Thumb, it produces an error. To get the 16-bit Thumb instruction you must remove the writeback.
- The logical operator `|` is deprecated because it can cause problems with substitution of variables in source. The assembler warns on the first use of `|` as a logical operator. Use the `:OR:` operator instead.

8.6.1 See also

Concepts

- *General changes between RVCT v3.1 and RVCT v4.0* on page 8-3.

Reference

Using the Assembler:

- *Addition, subtraction, and logical operators* on page 8-26.

Assembler Reference:

- *--depend=dependfile* on page 2-21
- *-o filename* on page 2-67.

8.7 fromelf changes between RVCT v3.1 and RVCT v4.0

Use of single letters as parameters to the `--text` option, either with a `/` or `=` as a separator is obsolete. The syntax `--text/cd` or `--text=cd` are no longer accepted. You have to specify `-cd`.

8.7.1 See also

Concepts

- *General changes between RVCT v3.1 and RVCT v4.0* on page 8-3.

Reference

- *--text* on page 4-73.

8.8 C and C++ library changes between RVCT v3.1 and RVCT v4.0

The following changes to the libraries have been made:

8.8.1 Support for non-standard C library math functions

Non-standard C library math functions are no longer supplied in `math.h`. They are still provided in the library itself. You can still request the header file from ARM if needed. Contact your supplier.

8.8.2 Remove `__ENABLE_LEGACY_MATHLIB`

In RVCT v2.2 changes were made to the behavior of some mathlib functions to bring them in-line with C99. If you relied on the old non-C99 behavior, you could revert the behavior by defining the following at compile time:

```
#define __ENABLE_LEGACY_MATHLIB
```

This has been removed in RVCT v4.0.

8.8.3 See also

Concepts

- *General changes between RVCT v3.1 and RVCT v4.0* on page 8-3.

Chapter 9
Migrating from RVCT v3.0 to RVCT v3.1

The following topic describes the changes that affect migration and compatibility between RVCT v3.0 and RVCT v3.1:

9.1 General changes between RVCT v3.0 and RVCT v3.1

The following changes affect multiple tools:

- support for the old ABI (`--apcs=/adsabi`) is no longer available
- `-O3` no longer implies `--fpmode=fast`.

9.1.1 Compatibility of RVCT v3.1 with legacy objects and libraries

RVCT v3.1 removes the option `--apcs /adsabi`. Compilation of ADS-compatible objects, and linking of legacy ADS objects and libraries is no longer possible.

Backwards compatibility of RVCT 2.x object and library code is supported provided you use the RVCT v3.1 linker and C/C++ libraries. Forward compatibility is not guaranteed.

Given these restrictions, ARM strongly recommends that you rebuild your entire project, including any user, or third-party supplied libraries, with RVCT v3.1. This is to avoid any potential incompatibilities, and to take full advantage of the improved optimization, enhancements, and new features provided by RVCT v3.1.

9.1.2 See also

Reference

Compiler Reference:

- *--apcs=qualifer...qualifier* on page 3-11
- *--dwarf2* on page 3-80
- *--dwarf3* on page 3-81
- *--fpmode=model* on page 3-97
- *-g* on page 3-105
- *-Onum* on page 3-156.

9.2 Assembler changes between RVCT v3.0 and RVCT v3.1

Disassembly output now conforms to the new *Unified Assembly Language* (UAL) format. VFP mnemonics have changed, so that `FMULS` is now `VMUL.F32`.

9.2.1 See also

Concepts

- *General changes between RVCT v3.0 and RVCT v3.1* on page 9-2.

Using the Assembler:

- *Unified Assembler Language* on page 5-3
- *Assembly language changes after RVCTv2.1* on page 5-39
- *VFP directives and vector notation* on page 9-39.

9.3 Linker changes between RVCT v3.0 and RVCT v3.1

In a scatter file, special case handling of the load address of root ZI has been removed. Consider:

```
LR1 0x8000
{
    ER_RO +0
    {
        *(+RO)
    }
    ER_RW +0
    {
        *(+RW)
    }
    ER_ZI +0
    {
        *(+ZI)
    }
}
LR2 +0
{
    ...
}
```

In versions of RVCT up to v3.0 the linker includes the size of `ER_ZI` when calculating the base address of `LR2`. The justification being that at image initialization `ER_ZI` is written over the top of `LR2`.

In version 3.1 and later this special case has been removed because you can write an equivalent scatter file using `ImageLimit()` built-in function, for example:

```
LR1 0x8000
{
    ER_RO +0
    {
         *(+RO)
    }
    ER_RW +0
    {
        *(+RW)
    }
    ER_ZI +0
    {
        *(+ZI)
    }
}
LR2 ImageLimit(LR1)
{
    ...
}
```

9.3.1 See also

Concepts

- *General changes between RVCT v3.0 and RVCT v3.1* on page 9-2.

Reference

Linker Reference:

- *Execution address built-in functions for use in scatter files* on page 4-35.

Chapter 10
Migrating from RVCT v2.2 to RVCT v3.0

The following topics describe the changes that affect migration and compatibility between RVCT v2.2 and RVCT v3.0:

10.1 General changes between RVCT v2.2 and RVCT v3.0

The following changes affect multiple tools:

- DWARF3 is the default.
- Since RVCT v2.1, `-g` no longer implies `-O0`. If you specify `-g` without an optimization level, the following warning is produced:

  ```
  Warning: C2083W: -g defaults to -O2 if no optimisation level is specified
  ```

10.1.1 Compatibility with legacy RVCT v2.x objects and libraries

Backwards compatibility of RVCT v2.x object and library code is supported provided you use the RVCT v3.0 linker and C/C++ libraries. Forward compatibility is not guaranteed.

You must link using the RVCT v3.0 linker, not the linker of older ARM tools. This is because older linkers cannot process objects produced by the RVCT v3.0 compiler.

Given these restrictions, ARM strongly recommends that you rebuild your entire project, including any user-supplied libraries, with RVCT v3.0 to avoid any potential incompatibilities, and to take full advantage of the improved optimization, enhancements, and new features provided by RVCT v3.0.

10.1.2 See also

Reference

Compiler Reference:

- *--apcs=qualifer...qualifier* on page 3-11
- *--dwarf2* on page 3-80
- *--dwarf3* on page 3-81
- *--fpmode=model* on page 3-97
- *-g* on page 3-105
- *-Onum* on page 3-156.

10.2 Compiler changes between RVCT v2.2 and RVCT v3.0

With the resolution of C++ core issue #446, temporaries are now created in some cases of conditional class rvalue expressions where they were not before.

Starting with RVCT v4.0 10Q1 (build 771) you can use the `--diag_warning=2817` command-line option to get a warning if this situation exists.

10.2.1 See also

Concepts

- *General changes between RVCT v2.2 and RVCT v3.0* on page 10-2.

Reference

Compiler Reference:

- *--diag_warning=optimizations* on page 3-76.

10.3 Linker changes between RVCT v2.2 and RVCT v3.0

The following changes have been made to the linker:

- If you specify the compiler option `--fpu=softvfp` together with a CPU with implicit VFP hardware, the linker no longer chooses a library that implements the software floating-point calls using VFP instructions. If you require this legacy behavior, specify the compiler option `--fpu=softvfp+vfp`.
- `armlink` writes the contents of load regions into the output ELF file in the order that load regions are written in the scatter file. Each load region is represented by one ELF program segment. In RVCT v2.2 the Program Header Table entries describing the program segments are given the same order as the program segments in the ELF file. To be more compliant with the ELF specification, in RVCT v3.0 and later the Program Header Table entries are sorted in ascending virtual address order.
- The `--no_strict_ph` command-line option has been added to switch off the sorting of the Program Header Table entries.

10.3.1 See also

Concepts

- *General changes between RVCT v2.2 and RVCT v3.0* on page 10-2.

Reference

- *--strict_ph, --no_strict_ph* on page 2-159.

Compiler Reference:

- *--cpu=name* on page 3-49
- *--fpu=name* on page 3-100.

10.4 C and C++ library changes between RVCT v2.2 and RVCT v3.0

The function `__user_initial_stackheap()` sets up and returns the locations of the initial stack and heap. In RVCT v3.x and later, ARM recommends you modify your source code to use `__user_setup_stackheap()` instead. `__user_initial_stackheap()` is still supported for backwards compatibility with earlier versions of the ARM C and C++ libraries.

If you continue to use `__user_initial_stackheap()`, be aware of the following changes in RVCT v3.0.

- In RVCT v2.x and earlier, the default implementation of `__user_initial_stackheap()` uses the value of the symbol `Image$$ZI$$Limit`. This symbol is not defined if you specify a scatter file using the `--scatter` linker command line option. Therefore, you must re-implement `__user_initial_stackheap()` to set the heap and stack boundaries if you are using a scatter file, otherwise your link step fails.

  In RVCT v3.x and later, multiple implementations of `__user_initial_stackheap()` are provided by the ARM C library. RVCT automatically selects the correct implementation using information given in the scatter file. This means that you do not have to re-implement this function if you are using scatter files.

- When migrating your application from RVCT v2.2 to RVCT v3.0, you might see the following linker error:

  ```
  Error L6218E: Undefined symbol main (referred from kernel.o).
  ```

  The linker is reporting that your application does not include a `main()` function. The error is generated because of the way RVCT v3.0 selects from the multiple implementations of `__user_initial_stackheap()`. These implementations refer to the `__rt_exit()` function. This is contained in `kernel.o`, that also contains the `__rt_lib_init()` function. Because the `__rt_lib_init()` function calls `main()`, this results in an undefined symbol error when `main()` is not present.

  If you do not provide a `main()` function as the entry point of your C code, the simplest solution is to implement either:

  — an empty, dummy `main()` function

  — a dummy implementation of `__rt_exit()`.

  If one of these stubs is contained in a separate source file, it is usually removed by the unused section elimination feature of the linker so there is no overhead in the final linked image.

10.4.1 See also

Concepts

- *General changes between RVCT v2.2 and RVCT v3.0* on page 10-2.

Using the Linker:

- *Elimination of unused sections* on page 5-4
- *Image symbols* on page 7-15.

Reference

ARM® C and C++ Libraries and Floating-Point Support Reference:

- *__user_setup_stackheap() on page 2-60*
- *Legacy function __user_initial_stackheap() on page 2-71*.

Linker Reference:

- *--scatter=filename* on page 2-142.

Appendix A
Revisions for Migration and Compatibility

The following technical changes have been made to *Migration and Compatibility*:

Table A-1 Differences between Issue G and Issue H

Change	Topics affected
Updated the FLEX*net* version used by ARM Compiler toolchain.	*FLEXnet versions used the in the compilation tools* on page 2-2
Corrected the GCC version emulated for ARM Compiler v5.01.	*GCC versions emulated* on page 2-3
Removed the changes that do not have an impact on migration or compatibility.	*Linker changes between ARM Compiler v4.1 and v4.1 SP1* on page 6-3

Table A-2 Differences between Issue F and Issue G

Change	Topics affected
Updated the table of FLEX*net* versions.	*FLEXnet versions used the in the compilation tools* on page 2-2

Table A-3 Differences between Issue D and Issue F

Change	Topics affected
Added new chapter for changes between v5.0.1 and v5.01.	Chapter 3 *Migrating from ARM Compiler v5.0 to v5.01 and later*
Updated the version of FLEXnet supported.	*FLEXnet versions used the in the compilation tools* on page 2-2
Updated the version of GCC supported.	*GCC versions emulated* on page 2-3
Added details about backwards compatibility with legacy objects and library code.	• *General changes between RVCT v4.0 and ARM Compiler v4.1* on page 7-2 • *General changes between RVCT v3.1 and RVCT v4.0* on page 8-3 • *General changes between RVCT v3.0 and RVCT v3.1* on page 9-2 • *General changes between RVCT v2.2 and RVCT v3.0* on page 10-2.
Where appropriate: • changed Thumb-2 to 32-bit Thumb.	• *C and C++ library changes between RVCT v4.0 and ARM Compiler v4.1* on page 7-7 • *Assembler changes between RVCT v3.1 and RVCT v4.0* on page 8-14
Modified the description of the C and C++ library changes between RVCT v2.2 and RVCT v3.0.	*C and C++ library changes between RVCT v2.2 and RVCT v3.0* on page 10-5
Added a description of linker warning L6932W when linking with helper libraries.	*Linker changes between RVCT v3.1 and RVCT v4.0* on page 8-8

Table A-4 Differences between Issue C and Issue D

Change	Topics affected
Added topic for Cygwin supported versions.	*Cygwin versions supported* on page 2-4
Added migration chapter for v4.1 Patch 3 to v5.0.	Chapter 4 *Migrating from ARM Compiler v4.1 Patch 3 to v5.0*
Added details about how `armlink` chooses libraries when specifying the compiler option `--fpu=softvfp` together with a CPU with implicit VFP hardware.	*Linker changes between RVCT v2.2 and RVCT v3.0* on page 10-4

Table A-5 Differences between Issue B and Issue C

Change	Topics affected
Added migration topic for v4.1 SP1 to v4.1 Patch 3.	*C and C++ library changes between ARM Compiler v4.1 SP1 and v4.1 Patch 3* on page 5-2
Added details about placing ARM library helper functions with scatter files.	*Linker changes between RVCT v3.1 and RVCT v4.0* on page 8-8
Added more examples to demonstrate the change to the 2 pass assembler.	*Assembler changes between RVCT v4.0 and ARM Compiler v4.1* on page 7-5

Table A-6 Differences between Issue A and Issue B

Change	Topics affected
Added details for migrating from ARM Compiler v4.1 to v4.1 SP1.	Chapter 6 *Migrating from ARM Compiler v4.1 to v4.1 SP1*
Added details for `__user_initial_stackheap()` and `__user_setup_stackheap()` for migrating from v2.2 to v3.0, and later.	*C and C++ library changes between RVCT v2.2 and RVCT v3.0* on page 10-5
Added details for the addition of *softfp* linkage functions in the hardware floating point version of the library.	*C and C++ library changes between ARM Compiler v4.1 and v4.1 SP1* on page 6-5
Added details for the deprecation of the `|` logical operator.	*Assembler changes between RVCT v3.1 and RVCT v4.0* on page 8-14