ARM® Compiler toolchain v5.02 for µVision

Migration and Compatibility

ARM®

ARM DUI 0593B (ID062912)

ARM Compiler toolchain v5.02 for µVision
Migration and Compatibility

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
June 2011	A	Non-Confidential	Release for ARM Compiler toolchain v4.1 for µVision
July 2012	B	Non-Confidential	Release for ARM Compiler toolchain v5.02 for µVision

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 v5.02 for µVision Migration and Compatibility

Chapter 1 **Conventions and feedback**

Chapter 2 **Migrating from ARM Compiler v4.1 for µVision to ARM Compiler v5.02 for µVision**

2.1 General changes between ARM Compiler v4.1 for µVision and ARM Compiler v5.02 for µVision 2-2
2.2 Compiler changes between ARM Compiler v4.1 for µVision and ARM Compiler v5.02 for µVision 2-3
2.3 Linker changes between ARM Compiler v4.1 for µVision and ARM Compiler v5.02 for µVision 2-4
2.4 Assembler changes between ARM Compiler v4.1 for µVision and ARM Compiler v5.02 for µVision 2-5

Chapter 3 **Migrating from RVCT v4.0 for µVision to ARM Compiler v4.1 for µVision**

3.1 General changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision 3-2
3.2 Compiler changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision 3-3
3.3 Linker changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision ... 3-4
3.4 Assembler changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision 3-5
3.5 C and C++ library changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision 3-7
3.6 fromelf changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision .. 3-8

Chapter 4 **Migrating from RVCT v3.1 for µVision to RVCT v4.0 for µVision**
4.1 General changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision 4-2
4.2 Changes to symbol visibility between RVCT v3.1 for µVision and RVCT v4.0 for µVision 4-3
4.3 Compiler changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision 4-5
4.4 Linker changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision 4-6
4.5 Assembler changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision 4-11
4.6 fromelf changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision 4-12
4.7 C and C++ library changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision 4-13

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 0593B
- 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 Product Manuals, `http://www.keil.com/support/man_arm.htm`
- Keil Support Knowledgebase, `http://www.keil.com/support/knowledgebase.asp`
- Keil Product Support, `http://www.keil.com/support/`
- ARM Glossary, `http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html`.

Chapter 2
Migrating from ARM Compiler v4.1 for µVision to ARM Compiler v5.02 for µVision

The following topics describe the changes that affect migration and compatibility between ARM Compiler v4.1 for µVision and ARM Compiler v5.02 for µVision:

- *General changes between ARM Compiler v4.1 for µVision and ARM Compiler v5.02 for µVision* on page 2-2
- *Compiler changes between ARM Compiler v4.1 for µVision and ARM Compiler v5.02 for µVision* on page 2-3
- *Linker changes between ARM Compiler v4.1 for µVision and ARM Compiler v5.02 for µVision* on page 2-4
- *Assembler changes between ARM Compiler v4.1 for µVision and ARM Compiler v5.02 for µVision* on page 2-5.

2.1 General changes between ARM Compiler v4.1 for µVision and ARM Compiler v5.02 for µVision

The following changes have been made in ARM Compiler v5.02 for µVision:

- 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`.
- 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 4.1 and earlier, the format is `VVbbbb`.
  - In version 5.02 and later, the format is `VVVbbbb`.

  For example, version 5.02 build 2345 is reported as `5022345`.

2.1.1 See also

Reference

Introducing the ARM® Compiler toolchain:

- *Toolchain environment variables* on page 2-12

Assembler Reference:

- *--version_number* on page 2-79

Compiler Reference:

- *--version_number on page 3-169*

Linker Reference:

- *--version_number* on page 2-142

Using the fromelf Image Converter:

- *--version_number* on page 4-55

Creating Static Software Libraries with armar:

- *--version_number* on page 6-32.

2.2 Compiler changes between ARM Compiler v4.1 for µVision and ARM Compiler v5.02 for µVision

The following changes to armcc have been made:

- 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 `ARMCC5INC` 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`.
- The *Link-time code generation* (LTCG) feature is deprecated. As an alternative ARM recommends you use the `--multifile` compiler option.

2.2.1 See also

Reference

Compiler Reference:

- *-Jdir[,dir,...] on page 3-93*
- *--multifile, --no_multifile on page 3-117*
- __attribute__((at(address))) variable attribute on page 5-72
- Predefined macros on page 5-183.

Introducing the ARM Compiler toolchain:

- *Toolchain environment variables* on page 2-12

2.3 Linker changes between ARM Compiler v4.1 for µVision and ARM Compiler v5.02 for µVision

The following changes to `armlink` have been made:

Library searching

If `ARMCC5LIB` 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`.

Link-time code generation

The *Link-time code generation* (LTCG) feature is deprecated. As an alternative ARM recommends you use the `--multifile` compiler option.

2.3.1 See also

Concepts

Introducing the ARM Compiler toolchain:

- *About the ARM Compiler toolchain* on page 2-3.

Reference

Linker Reference:

- *--libpath=pathlist* on page 2-74.

Compiler Reference:

- *--multifile, --no_multifile on page 3-117*.

Introducing the ARM Compiler toolchain:

- *Toolchain environment variables* on page 2-12.

2.4 Assembler changes between ARM Compiler v4.1 for µVision and ARM Compiler v5.02 for µVision

There are no technical changes to `armasm` that affect migration between ARM Compiler v4.1 and v5.02.

Chapter 3
Migrating from RVCT v4.0 for µVision to ARM Compiler v4.1 for µVision

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

- *General changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision* on page 3-2
- *Compiler changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision* on page 3-3
- *Linker changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision* on page 3-4
- *Assembler changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision* on page 3-5
- *C and C++ library changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision* on page 3-7.

3.1 General changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision

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`.

3.1.1 See also

Reference

Introducing the ARM® Compiler toolchain:

- *Toolchain environment variables* on page 2-12.

3.2 Compiler changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision

The following changes to the compiler have been made:

Sign rules on enumerators have changed in line with convention. The enumerator container is now unsigned unless a negative constant is defined.

`-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.

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)));
```

3.2.1 See also

Concepts

- *General changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision* on page 3-2.

Reference

Compiler Reference:

- *--multifile, --no_multifile on page 3-117*
- *-Onum on page 3-122*.
- *__attribute__((at(address))) variable attribute* on page 5-70.

3.3 Linker changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision

The following changes to the linker have been made:

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.

3.3.1 See also

Concepts

- *General changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision* on page 3-2.

Introducing the ARM Compiler toolchain:

- *About the ARM Compiler toolchain* on page 2-3.

3.4 Assembler changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision

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`.

Obsolete command-line option

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

3.4.1 See also

Concepts

- *General changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision* on page 3-2.

Reference

Assembler Reference:

- *--diag_warning=tag{, tag}* on page 2-27
- *-o filename* on page 2-64.

Using the Assembler:

- *How the assembler works* on page 2-4
- *2 pass assembler diagnostics* on page 7-20.

3.5 C and C++ library changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision

The libraries now use more Thumb2 code on targets that support Thumb2. 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
```

You can enable the newer behavior with:

```
#pragma import __use_c99_matherr.
```

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.

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 software might contain operations that are no longer required, such as implementing `__user_perthread_libspace` for the `alloca` state that is no longer used.

3.5.1 See also

Concepts

- *General changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision* on page 3-2.

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

- Chapter 2 *The ARM C and C++ libraries*.
- *How the ARM C library fulfills ISO C specification requirements* on page 2-107.
- *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.

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 3-10*
- *--fpu=name on page 3-75*.

3.6 fromelf changes between RVCT v4.0 for µVision and ARM Compiler v4.1 for µVision

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

3.6.1 See also

Reference

Using the fromelf Image Converter:

- *input_file* on page 4-38.

Chapter 4
Migrating from RVCT v3.1 for µVision to RVCT v4.0 for µVision

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

- *General changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision* on page 4-2
- *Changes to symbol visibility between RVCT v3.1 for µVision and RVCT v4.0 for µVision* on page 4-3
- *Compiler changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision* on page 4-5
- *Linker changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision* on page 4-6
- *Assembler changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision* on page 4-11
- *fromelf changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision* on page 4-12
- *C and C++ library changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision* on page 4-13.

4.1 General changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision

The following changes affect multiple tools:

4.1.1 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.

4.1.2 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.

4.1.3 See also

Reference

Compiler Reference:

- *--cpu=name on page 3-33*
- *--fpu=name on page 3-75*.

Linker Reference:

- *--cpu=name* on page 2-28
- *--fpu=name* on page 2-56.

Assembler Reference:

- *--cpu=name* on page 2-19
- *--fpu=name* on page 2-37
- *--unsafe* on page 2-77.

4.2 Changes to symbol visibility between RVCT v3.1 for µVision and RVCT v4.0 for µVision

The following changes to symbol visibility have been made:

4.2.1 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 for µVision and earlier was STV_DEFAULT. In RVCT v4.0 for µVision 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 v3.1for µVision.

4.2.2 `__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;
```

4.2.3 RVCT v3.1 for µVision symbol visibility summary

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

Table 4-1 RVCT v3.1 for µVision 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 4-2 RVCT v3.1 for µVision 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

4.2.4 RVCT v4.0 for µVision symbol visibility summary

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

Table 4-3 RVCT v4.0 for µVision 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 4-4 RVCT v4.0 for µVision 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

4.2.5 See also

Concepts

- *General changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision* on page 4-2.

4.3 Compiler changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision

The following compiler changes have been made:

4.3.1 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.

4.3.2 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 for µVision 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 for µVision, 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.

4.3.3 See also

Concepts

- *General changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision* on page 4-2.

Reference

Compiler Reference:

- *--cpp on page 3-31*
- *--reassociate_saturation, --no_reassociate_saturation on page 3-143*
- *--thumb on page 3-160*

4.4 Linker changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision

The following linker changes have been made:

4.4.1 Placing ARM library helper functions with scatter files

In RVCT v3.1 for µVision and earlier, the helper functions reside in libraries provided with the ARM compiler. 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.

In RVCT v4.0 for µVision 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.

4.4.2 Linker steering files and symbol visibility

In RVCT v3.1 for µVision 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 for µVision 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.

4.4.3 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.

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 for µVision 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 for µVision linker implements these in the same way as the execution region-related symbols.

They are defined only when scatter files are not used.

```
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.

4.4.4 Build attributes

The RVCT v4.0 for µVision 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-R4F implies --fpu=vfpv3_d16. In RVCT v3.1 for µVision 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.

4.4.5 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.

4.4.6 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 for µVision 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 for µVision.

4.4.7 See also

Concepts

- *General changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision* on page 4-2.

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 8-51.

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:

- *--cpu=name* on page 2-28
- *--force_explicit_attr* on page 2-54
- *--fpu=name* on page 2-56
- *--map, --no_map* on page 2-83
- *--override_visibility* on page 2-90
- *EXPORT* on page 3-2
- *IMPORT* on page 3-4
- *Execution region attributes* on page 4-12.

4.5 Assembler changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision

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 Thumb-2, 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.

4.5.1 See also

Concepts

- *General changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision* on page 4-2.

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 2-64.

4.6 fromelf changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision

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`.

4.6.1 See also

Concepts

- *General changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision* on page 4-2.

Reference

- *--text* on page 4-53.

4.7 C and C++ library changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision

The following changes to the libraries have been made:

4.7.1 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.

4.7.2 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 for µVision.

4.7.3 See also

Concepts

- *General changes between RVCT v3.1 for µVision and RVCT v4.0 for µVision* on page 4-2.