ARM® Compiler toolchain

Version 4.1

Migration and Compatibility

ARM DUI 0530A (ID052210)

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 v4.1 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 0530A
- 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`
- Keil Distributors, `http://www.keil.com/distis`.

Chapter 2
Configuration information for different versions of the compilation tools

The following topics summarize the FLEX*net* and GCC versions used in the different versions of the 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 v4.1	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

- *FLEXnet for ARM Tools License Management Guide*, `http://infocenter.arm.com/help/topic/com.arm.doc.dui0209-/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
RVCT 4.0	4.2.0
RVCT 3.1	3.3.0

2.2.1 See also

Reference

- Chapter 3 *Migrating from RVCT v4.0 to ARM Compiler v4.1*
- Chapter 4 *Migrating from RVCT v3.1 to RVCT v4.0.*

Compiler Reference:

- *--gnu_version=version* on page 3-82.

Chapter 3
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:

3.1 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`.

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 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.

3.2.1 See also

Reference

Compiler Reference:

- *--multifile, --no_multifile* on page 3-109
- *-Onum* on page 3-114.

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

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.

3.3.1 See also

Reference

Linker Reference:

- --*use_sysv_default_script, --no_use_sysv_default_script* on page 2-159
- --*sysv* on page 2-153.

3.4 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 allowed 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.

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.

3.4.1 See also

Reference

Assembler Reference:

- *--diag_warning=tag{, tag}* on page 2-14
- *-o filename* on page 2-24.

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

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
```

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

```
#pragma import __use_c99_matherr.
```

3.5.1 See also

Concepts

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

- *How the ARM C library fulfills ISO C specification requirements* on page 2-127.

Chapter 4
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:

4.1 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.

4.1.1 See also

Reference

Compiler Reference:

- *--gnu_version=version* on page 3-82.

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

The following changes affect multiple tools:

4.2.1 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.

4.2.2 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.

4.2.3 See also

Reference

Compiler Reference:

- *--cpu=name* on page 3-41
- *--fpu=name* on page 3-75.

Linker Reference:

- *--cpu=name* on page 2-30
- *--fpu=name* on page 2-66.

Assembler Reference:

- *--cpu=name* on page 2-10
- *--fpu=name* on page 2-17
- *--unsafe* on page 2-29.

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

The following changes to symbol visibility have been made:

4.3.1 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.

4.3.2 `__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;
```

4.3.3 RVCT 3.1 symbol visibility summary

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

Table 4-1 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 4-2 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

4.3.4 RVCT v4.0 symbol visibility summary

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

Table 4-3 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 4-4 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

4.3.5 See also

Reference

Compiler Reference:

- *--default_definition_visibility=visibility* on page 3-48
- *--dllexport_all, --no_dllexport_all* on page 3-61
- *--dllimport_runtime, --no_dllimport_runtime* on page 3-61
- *--gnu* on page 3-80
- *--hide_all, --no_hide_all* on page 3-84.

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

The following compiler changes have been made:

4.4.1 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.

4.4.2 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.

4.4.3 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 3.1 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 4.0, you might see a performance degradation because of this issue.

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

4.4.4 See also

Reference

Compiler Reference:

- *--cpp* on page 3-40
- *--reassociate_saturation, --no_reassociate_saturation* on page 3-128
- *--thumb* on page 3-142
- *--vectorize, --no_vectorize* on page 3-153.

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

The following linker changes have been made:

4.5.1 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.

4.5.2 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 allows 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$$region_name$$Base
Load$$region_name$$Limit
Load$$region_name$$Length

Load$$region_name$$RO$$Base
Load$$region_name$$RO$$Limit
Load$$region_name$$RO$$Length

Load$$region_name$$RW$$Base
Load$$region_name$$RW$$Limit
Load$$region_name$$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 allows 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$$ER_RO$$Base
Image$$RO$$Limit ; Equivalent to Image$$ER_RO$$Limit

Image$$RW$$Base  ; Equivalent to Image$$ER_RW$$Base
Image$$RW$$Limit ; Equivalent to Image$$ER_RW$$Limit

Image$$ZI$$Base  ; Equivalent to Image$$ER_ZI$$Base
Image$$ZI$$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.

4.5.3 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 allow `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.

4.5.4 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.

4.5.5 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.

4.5.6 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 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 allow 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.

4.5.7 See also

Concepts

Using the Linker:

- *Optimization with RW data compression* on page 5-13
- *Accessing linker-defined symbols* on page 7-3
- *Region-related symbols* on page 7-4
- *Image$$ execution region symbols* on page 7-5
- *Load$$ execution region symbols* on page 7-6
- *Importing linker-defined symbols* on page 7-11

- *Section-related symbols* on page 7-12.

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

- *Initialization of the execution environment and execution of the application* on page 2-65.

Reference

Linker Reference:

- *--bpabi* on page 2-16
- *--cpu=name* on page 2-30
- *--force_explicit_attr* on page 2-62
- *--fpu=name* on page 2-66
- *--map, --no_map* on page 2-96
- *--override_visibility* on page 2-103
- *--prelink_support, --no_prelink_support* on page 2-113
- *--sysv* on page 2-153
- *EXPORT* on page 3-2
- *IMPORT* on page 3-4
- *Execution region attributes* on page 4-11.

4.6 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 Thumb-2, it produces an error. To get the 16-bit Thumb instruction you must remove the writeback.

4.6.1 See also

Reference

Assembler Reference:

- *--depend=dependfile* on page 2-11
- *-o filename* on page 2-24.

4.7 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`.

4.7.1 See also

Reference

- --*text* on page 4-72.

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

The following changes to the libraries have been made:

4.8.1 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.

4.8.2 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.

Chapter 5
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:

5.1 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`.

5.1.1 See also

Reference

Compiler Reference:

- *--apcs=qualifer...qualifier* on page 3-9
- *--dwarf2* on page 3-62
- *--dwarf3* on page 3-63
- *--fpmode=model* on page 3-72
- *-g* on page 3-79
- *-Onum* on page 3-114.

5.2 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`.

5.2.1 See also

Concepts

Using the Assembler:

- *Unified Assembler Language* on page 5-3
- *Assembly language changes after RVCTv2.1* on page 5-42
- *VFP directives and vector notation* on page 9-43.

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

In a scatter-loading 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)
{
    ...
}
```

5.3.1 See also

Reference

Linker Reference:

- *Execution address built-in functions for use in scatter files* on page 4-30.

Chapter 6
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:

- *General changes between RVCT v2.2 and RVCT v3.0* on page 6-2
- *Compiler changes between RVCT v2.2 and RVCT v3.0* on page 6-3
- *Linker changes between RVCT v2.2 and RVCT v3.0* on page 6-4.

6.1 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
```

6.1.1 See also

Reference

Compiler Reference:

- *--apcs=qualifer...qualifier* on page 3-9
- *--dwarf2* on page 3-62
- *--dwarf3* on page 3-63
- *--fpmode=model* on page 3-72
- *-g* on page 3-79
- *-Onum* on page 3-114.

6.2 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 4.0 10Q1 (build 771) you can use the `--diag_warning=2817` command-line option to get a warning if this situation exists.

6.2.1 See also

Reference

Compiler Reference:

- *--diag_warning=optimizations* on page 3-60.

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

`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.

6.3.1 See also

Reference

- *--strict_ph, --no_strict_ph* on page 2-144.