RealView® Compilation Tools

Version 3.0

Essentials Guide

ARM®

Copyright © 2002-2006 ARM Limited. All rights reserved.
ARM DUI 0202G

RealView Compilation Tools
Essentials Guide

Copyright © 2002-2006 ARM Limited. All rights reserved.

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
August 2002	A	Non-Confidential	Release 1.2
January 2003	B	Non-Confidential	Release 2.0
September 2003	C	Non-Confidential	Release 2.0.1 for RVDS v2.0
January 2004	D	Non-Confidential	Release 2.1 for RVDS v2.1
December 2004	E	Non-Confidential	Release 2.2 for RVDS v2.2
May 2005	F	Non-Confidential	Release 2.2 for RVDS v2.2 SP1
March 2006	G	Non-Confidential	Release 3.0 for RVDS v3.0

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks owned by ARM Limited. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM Limited shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

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

Copyright © 2002-2006 ARM Limited. All rights reserved.

Copyright © 2002-2006 ARM Limited. All rights reserved.

Contents
RealView Compilation Tools Essentials Guide

Copyright © 2002-2006 ARM Limited. All rights reserved.

 Copyright © 2002-2006 ARM Limited. All rights reserved.

Preface

This preface introduces the *RealView Compilation Tools Essentials Guide*. It contains the following sections:

Copyright © 2002-2006 ARM Limited. All rights reserved.

About this book

This book provides an overview of the *RealView® Compilation Tools* (RVCT) tools and documentation.

Intended audience

This book is written for all developers who are producing applications using RVCT. It assumes that you are an experienced software developer.

Using this book

This book is organized into the following chapters and appendixes:

Chapter 1 ***Introduction***

Read this chapter for an introduction to RVCT. The components of RVCT and the online documentation are described.

Chapter 2 ***Differences***

Read this chapter for details of the differences between the new release of RVCT and the previous release.

Chapter 3 ***Creating an Application***

Read this chapter for a brief overview of how to create an application using RVCT.

Appendix A ***About Previous Releases***

Read this appendix for details of new features and changes in previous releases of RVCT.

This book assumes that you have installed your ARM® software in the default location, for example, on Windows this might be *volume*:\Program Files\ARM. This is assumed to be the location of *install_directory* when referring to path names, for example, *install_directory*\Documentation\.... You might have to change this if you have installed your ARM software in a different location.

Copyright © 2002-2006 ARM Limited. All rights reserved.

Typographical conventions

The following typographical conventions are used in this book:

italic	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes ARM processor signal names. Also used for terms in descriptive lists, where appropriate.
`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.

Further reading

This section lists publications from ARM Limited that provide additional information on developing code for the ARM family of processors.

ARM Limited periodically provides updates and corrections to its documentation. See `http://www.arm.com` for current errata sheets and addenda, and the ARM Frequently Asked Questions.

ARM publications

This book contains general information about RVCT. Other publications included in the suite are:

- *RealView Compilation Tools v3.0 Compiler and Libraries Guide* (ARM DUI 0205). This book provides reference information for RVCT. It describes the command-line options to the compiler and gives reference material on the ARM implementation of the C and C++ compiler and the C libraries.
- *RealView Compilation Tools v3.0 Assembler Guide* (ARM DUI 0204). This book provides reference and tutorial information on the ARM assembler.

Copyright © 2002-2006 ARM Limited. All rights reserved.

- *RealView Compilation Tools v3.0 Linker and Utilities Guide* (ARM DUI 0206). This book provides reference information on the command-line options to the ARM linker and the fromelf utility.
- *RealView Compilation Tools v3.0 Developer Guide* (ARM DUI 0203). This book provides tutorial information on writing code targeted at the ARM family of processors.
- *RealView Development Suite Glossary* (ARM DUI 0324). An alphabetically arranged glossary defines the special terms used in the documentation suite.

For full information about the base standard, software interfaces, and standards supported by ARM, see *install_directory*\Documentation\Specifications\....

In addition, refer to the following documentation for specific information relating to ARM products:

- *ARM Architecture Reference Manual* (ARM DDI 0100)
- *ARM Reference Peripheral Specification* (ARM DDI 0062)
- the ARM datasheet or technical reference manual for your hardware device.

Other publications

For an introduction to ARM architecture, see Andrew N. Sloss, Dominic Symes and Chris Wright, *ARM System Developer's Guide: Designing and Optimizing System Software* (2004). Morgan Kaufmann, ISBN 1-558-60874-5.

Other publications are also listed elsewhere.

Copyright © 2002-2006 ARM Limited. All rights reserved.

Feedback

ARM Limited welcomes feedback on both RealView Compilation Tools and its documentation.

Feedback on the RealView Compilation Tools

If you have any problems with RVCT, contact your supplier. To help them provide a rapid and useful response, give:

- your name and company
- the serial number of the product
- details of the release you are using
- details of the platform you are running on, 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 tool, including the version number and date.

Feedback on this book

If you notice any errors or omissions in this book, send email to `errata@arm.com` giving:

- the document title
- the document number
- the page number(s) to which your comments apply
- a concise explanation of the problem.

General suggestions for additions and improvements are also welcome.

Copyright © 2002-2006 ARM Limited. All rights reserved.

 Copyright © 2002-2006 ARM Limited. All rights reserved.

Chapter 1
Introduction

This chapter introduces *RealView® Compilation Tools* (RVCT) and describes its software components and documentation. It contains the following sections:

- *About RealView Compilation Tools* on page 1-2
- *Getting more information online* on page 1-7.

Copyright © 2002-2006 ARM Limited. All rights reserved.

1.1 About RealView Compilation Tools

RVCT consists of a suite of tools, together with supporting documentation and examples, that enable you to write and build applications for the ARM® family of *Reduced Instruction Set Computing* (RISC) processors.

You can use RVCT to build software programs in C, C++, or ARM assembly language.

1.1.1 Components of RVCT

For information about RVCT see:

- *Development tools*
- *Standards compliance* on page 1-4
- *Compliance with the ABI for the ARM Architecture (base standard)* on page 1-4
- *Supporting software* on page 1-6
- *Code examples* on page 1-6.

Development tools

The following development tools are provided when you install RVCT:

armcc The ARM and Thumb® C and C++ compiler. It compiles either:

- ISO C:1990 source
- ISO C++:1998 source

into:

- 32-bit ARM code
- 16-bit Thumb code
- 16/32-bit Thumb-2 code.

The Thumb-2 instruction set includes older 16-bit Thumb instructions as a subset. Thumb-2 introduces many new 32-bit instructions and some new 16-bit instructions. On those processors that support it, Thumb-2 gives near ARM performance together with code size that is similar to older versions of Thumb.

armasm The ARM and Thumb assembler. This assembles ARM assembly language and Thumb/Thumb-2 assembly language source.

armlink The ARM linker. This combines the contents of one or more object files with selected parts of one or more object libraries to produce an executable program. The ARM linker creates ELF executable images.

 Copyright © 2002-2006 ARM Limited. All rights reserved.

Rogue Wave C++ library

The Rogue Wave library provides an implementation of the standard C++ library as defined in the *ISO/IEC 14822:1998 International Standard for C++*. For more information on the Rogue Wave library, see the HTML documentation on the CD ROM.

C++ runtime libraries

The ARM C++ runtime libraries enable support for core C++ language features, as provided in the C++ header files `new`, `typeinfo`, and `exception` in *`install_directory`*`\RVCT\Data\...\include\...`.

C library The ARM C library provides an implementation of the library features as defined in the *ISO/IEC 9899:1990, C Standard* and the *Normative Addendum 1* from 1994.

fromelf The ARM image conversion utility. This accepts ELF format input files and converts them to a variety of output formats, including:

- plain binary
- Motorola 32-bit S-record format
- Intel Hex 32 format
- Verilog-like hex format.

`fromelf` can also generate text information about the input image, such as disassembly and its code and data size.

armar The ARM librarian enables sets of ELF format object files to be collected together and maintained in libraries. You can pass such a library to the linker in place of several ELF files.

Note

RVCT tools do not currently provide 64-bit support. All file I/O routines use standard system calls with file sizes that fit in a 32-bit (**`signed`**) **`int`**. This means that image size is currently limited to 2GB.

Sun Solaris machines are all 64-bit but they will hit memory problems when building large images that extend over the 2GB boundary. This generates an `armlink` warning message (L6000U) to indicate that there is not enough memory. This might cause confusion since sufficient memory is available but the application cannot access it.

Copyright © 2002-2006 ARM Limited. All rights reserved.

Standards compliance

RVCT conforms to the following standards. In each case, the level of compliance is noted:

ar `armar` produces, and `armlink` consumes, UNIX-style object code archives. `armar` can list and extract most `ar`-format object code archives, and `armlink` can use an `ar`-format archive created by another archive utility provided that it contains a symbol table member.

DWARF 3 DWARF 3 debug tables (Draft Standard 9.7) are supported by all the tools in the RVCT suite, and by DWARF 3 compatible debuggers from ARM, for example, RealView Debugger.

DWARF 3 debug tables are not supported by *ARM eXtended Debugger* (AXD) or *ARM Symbolic Debugger* (`armsd`).

DWARF 2 DWARF 2 debug tables are supported by all the tools in the RVCT suite, and by ELF/DWARF 2 compatible debuggers from ARM, for example, RealView Debugger, AXD, and `armsd`.

Note

The DWARF 2 standard is ambiguous in some areas (such as debug frame data) so there is no guarantee that third-party debuggers can consume the DWARF 2 produced by ARM code generation tools or that ARM debuggers can consume the DWARF 2 produced by third-party tools.

ISO C The ARM compiler accepts *ISO/IEC 9899:1990 C*, including the *Normative Addendum 1* from 1994 (except file I/O), as input. The option `--strict` can be used to enforce strict ISO compliance.

ISO C++ The ARM compiler supports the full *ISO/IEC 14822:1998 C++* language, with the exception of the **`export`** keyword.

ELF The ARM tools produce relocatable and executable files in ELF format. The fromelf utility can translate ELF files into other formats.

Compliance with the ABI for the ARM Architecture (base standard)

The *Application Binary Interface* (ABI) for the ARM Architecture is a collection of standards, some open and some specific to the ARM architecture, that regulate the inter-operation of binary code and development tools in ARM-based execution environments ranging from bare metal to major operating systems such as ARM Linux.

By conforming to this standard, ARM and Thumb objects and libraries from different producers can work together.

 Copyright © 2002-2006 ARM Limited. All rights reserved.

The *ABI for the ARM Architecture (base standard)* [BSABI] consists of a family of specifications including:

AAPCS *Procedure Call Standard for the ARM Architecture*. Governs the exchange of control and data between functions at runtime. There is a variant of the AAPCS for each of the major execution environment types supported by RVCT.

CPPABI *C++ ABI for the ARM Architecture*. Builds on the generic C++ ABI (originally developed for IA-64) to govern interworking between independent C++ compilers.

EHABI *Exception Handling ABI for the ARM Architecture.* Defines both the language-independent and C++-specific aspects of how exceptions are thrown and handled.

AAELF *ELF for the ARM Architecture*. Builds on the generic ELF standard to govern the exchange of linkable and executable files between producers and consumers.

AADWARF *DWARF for the ARM Architecture.* This ABI uses DWARF 3 to govern the exchange of debugging data between object producers and debuggers.

RTABI *Runtime ABI for the ARM Architecture*. Governs what independently produced objects can assume of their execution environments by way of floating-point and compiler helper function support.

CLIBABI *C Library ABI for the ARM Architecture*. Defines an ABI to the ISO C:1990 library.

BPABI *Base Platform ABI for the ARM Architecture*. Governs the format and content of executable and shared object files generated by static linkers. Supports platform-specific executable files using post linking. Provides a base standard that is used to derive a platform ABI.

For more information on the base standard, software interfaces, and standards supported by ARM, see *install_directory*`\Documentation\Specifications\...`.

For details of the latest published versions, see `http://www.arm.com`.

If you are upgrading to the latest release of RVCT from a previous release, ensure that you are using the most recent versions of the ARM specifications.

Copyright © 2002-2006 ARM Limited. All rights reserved.

Supporting software

To debug your programs under simulation, or on hardware based on an ARM core, use a suitable debugger, for example:

- ARM RealView Debugger v3.0. This is compatible with ELF, DWARF 2, or DWARF 3 as produced by GCC 3.4 or RVCT v2.2 or above.
- AXD v1.3. This is compatible with ELF and DWARF 2.

To debug your programs under simulation, use the *RealView ARMulator® ISS* (RVISS) or the *Instruction Set System Model* (ISSM) supporting software. RVISS is an *Instruction Set Simulator* (ISS) that is supplied with RealView Development Suite. It communicates with a debugger, for example, RealView Debugger, and can run on the same host computer or on a system remote from that running the debugger. For more details, see *RealView ARMulator ISS User Guide*.

Simulator models for Cortex™ processors (Cortex-A8 and Cortex-M3) are now available with RealView Development Suite v3.0. These models are accessible through the ISSM target access in RealView Debugger v3.0.

Code examples

This book references examples provided with RealView Development Suite in the main examples directory *install_directory*\RVDS\Examples. See *RealView Development Suite Getting Started Guide* for a summary of the examples provided.

 Copyright © 2002-2006 ARM Limited. All rights reserved.

1.2 Getting more information online

Depending on your installation, the full documentation suite is available online as PDF and DynaText for Windows and Sun Solaris, and as PDF for Red Hat Linux. The PDF and DynaText files contain the same information.

RealView Development Suite v3.0 includes a new *PDF Documentation Suite* that can be accessed from a single PDF, `Collection.pdf`. If you install the full documentation suite, a text search of all the PDF files is possible from this collection.

In addition, documentation for the Rogue Wave C++ library is available in HTML format on all supported platforms. This is installed by default for a Typical installation.

For more details, see:

- *RVCT on Windows*
- *RVCT on Sun Solaris and Red Hat Linux*
- *Rogue Wave documentation* on page 1-8.

1.2.1 RVCT on Windows

Select **Programs → ARM** from the Windows **Start** menu. From here select either:

- **RealView Development Suite v3.0 → RVDS v3.0 Documentation Suite** to view the PDF files
- **DynaText Documentation** to view the DynaText files.

Unless you change the defaults, the PDF and DynaText files are installed in *`install_directory`*`\Documentation\RVCT\...\windows`.

1.2.2 RVCT on Sun Solaris and Red Hat Linux

The full documentation suite is available as PDF and DynaText files for Sun Solaris, and as PDF on Red Hat Linux. If you have set up desktop links, use these to access the required documentation.

Unless you change the defaults, the PDF and DynaText files are installed in *`install_directory`*`/Documentation/RVCT/.../unix`.

Note

DynaText files are not installed on Red Hat Linux.

Copyright © 2002-2006 ARM Limited. All rights reserved.

1.2.3 Rogue Wave documentation

The manuals for the Rogue Wave Standard C++ Library for RVCT are provided on the product CD ROM as HTML files. Use a web browser, such as Netscape or Internet Explorer, to view these files. For example, select the file *install_directory*\Documentation\RogueWave\1.0\release\stdref\index.htm to display the HTML documentation for Rogue Wave (see Figure 1-1 where the *install_directory* is D:\ARM).

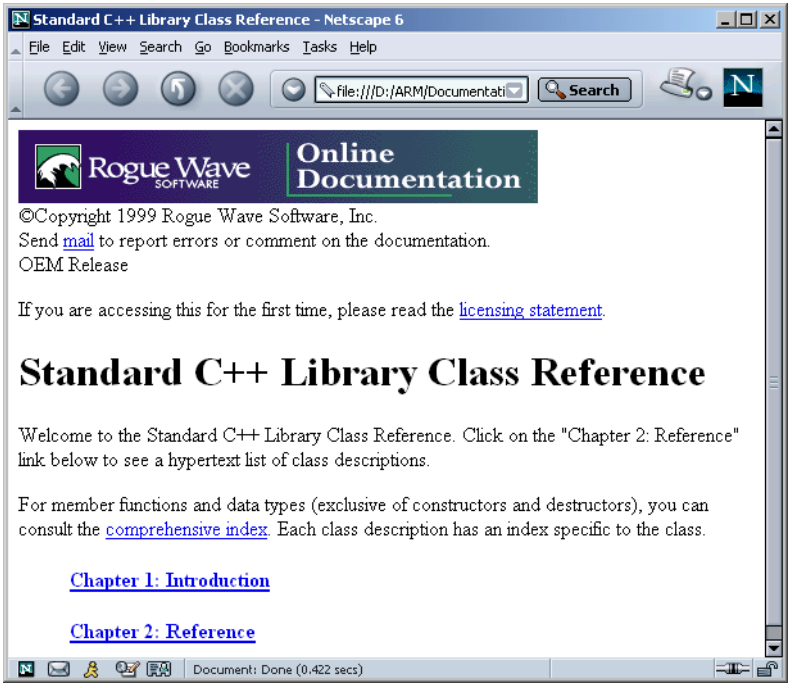

Figure 1-1 HTML browser

Copyright © 2002-2006 ARM Limited. All rights reserved.

Chapter 2
Differences

This chapter describes the major differences between the latest release of *RealView® Compilation Tools* and the previous release.

It contains the following sections:

- *RVCT v3.0 overview* on page 2-2
- *Differences between RVCT v3.0 and RVCT v2.2* on page 2-6
- *Compatibility of RVCT v3.0 with legacy objects and libraries* on page 2-12.

For information about previous releases of RVCT, see:

- Appendix A *About Previous Releases*
- Appendix E in *RealView Compilation Tools v3.0 Compiler and Libraries Guide*.

Copyright © 2002-2006 ARM Limited. All rights reserved.

2.1 RVCT v3.0 overview

The most important differences between RVCT v3.0 and RVCT v2.2 are:

- RVCT v3.0 includes support for two ARMv7 architecture profiles:

  ARMv7-A

  The application profile for systems supporting the ARM and Thumb-2 instruction sets, with Thumb®-2EE, for virtual (MMU-based) memory systems.

  ARMv7-M

  The microcontroller profile for Thumb-2 only.

  RVCT v3.0 includes support for all ARM architectures from ARMv4 onwards. All architecture names prior to ARMv4 are now obsolete and no longer supported.

- RVCT v3.0 supports two of the ARM Cortex™ processor family:
  — Cortex-A8
  — Cortex-M3.

  To see a full list of supported architectures and cores use:

  ```
  armcc --cpu list
  ```

- RVCT v3.0 contains preliminary support for two components of ARMv7 architecture, that is, the Advanced SIMD Extension (also called NEON™ Technology) and VFPv3. This means that a substantial number of SIMD (*Single Instruction, Multiple Data*) instructions and some VFPv3 instructions are now available in both ARM and Thumb-2 instruction sets.

  NEON is a 64/128 bit hybrid SIMD technology targeted at signal processing applications and embedded processors. NEON is implemented as part of the processor, but has its own execution pipelines and a register bank that is distinct from the ARM register bank. NEON instructions are available in both ARM and Thumb-2.

  To see a full list of supported floating-point architectures use:

  ```
  armcc --fpu list
  ```

- RVCT v3.0 supports the Thumb-2 Execution Environment (Thumb-2EE) for ARMv7. The Thumb-2EE instruction set is based on Thumb-2, with some changes and additions to make it a better target for dynamically generated code.

  The Thumb-2EE extension introduces a new instruction set state, ThumbEE state. In this state, the instruction set is almost identical to the Thumb-2 instruction set. However, some instructions have modified behavior, and some new instructions are available.

 Copyright © 2002-2006 ARM Limited. All rights reserved.

- The ARM assembler can be used to assemble Intel® Wireless MMX™ Technology instructions to develop code, for example, for the PXA270 processor.
- RVCT v3.0 provides full support for DWARF 3 (Draft Standard 9.7) debug tables, as required by the latest release of the *ABI for the ARM Architecture (base standard)* [BSABI].

  DWARF 3 is the default format and includes:
  - full support for debugging C++ namespaces
  - an increase in the size of debug information.

  In this release of the compiler and assembler:
  - if you do not specify a format, DWARF 3 is assumed
  - if you specify `--debug` (`-g`), DWARF 3 is generated by default.

  The command-line option, `--dwarf2`, is still supported for backwards compatibility.
- The ARM compiler and linker support *Thread Local Storage* (TLS) to enable programs to use multiple threads. Two new keywords are available to support TLS on MPCore™:
  - `__declspec(thread)`
  - `__thread.`
- The ARM compiler now includes high-level scalar optimizations, including loop unrolling. These enhancements are automatically invoked when compiling with `-O3` and can give significant performance benefits at a small code size cost.
- In RVCT v3.0, dividing an integer (either **`int`** or **`long long`**) by zero returns zero by default. This is a change to the previous behavior where the result was to terminate the program with an error message.
- In RVCT v3.0, you can change the language used for error and warning messages. For example, to display messages in Japanese on an English-based workstation, use:

  ```
  --message_locale ja_JP
  ```
- Previously available in the ARM compiler only, the `--show_cmdline` option is now available in the assembler, linker, `armar`, and `fromelf`. Use this to see how the command line has been processed. The commands are shown normalized, and the contents of any via files are expanded. However, using this option does not trap accidental errors in your command line.
- RVCT v3.0 introduces the environment variables:
  - `RVCT30_CCOPT`
  - `RVCT30_LINKOPT`

Copyright © 2002-2006 ARM Limited. All rights reserved.

— RVCT30_ASMOPT.

The value of this variable is inserted at the front of the command string as appropriate for each of the RVCT tools. Options specified in the environment variables can be overridden by arguments on the command-line.

See *Differences between RVCT v3.0 and RVCT v2.2* on page 2-6 for more details on new features, and changes in behavior, outlined in this section.

2.1.1 Obsolete features

Be aware of the following differences in RVCT v3.0:

- All features and command-line options that were deprecated in RVCT v2.2 are obsolete in RVCT v3.0. See Appendix E in *RealView Compilation Tools v3.0 Compiler and Libraries Guide* for a list.
- The previous release of RVCT supported old compiler options to help you to migrate your message options to the new release. However, these options are now obsolete and no longer supported. See Appendix E in *RealView Compilation Tools v3.0 Compiler and Libraries Guide* for details.
- Multiline strings are not supported in the ARM compiler. This behavior was previously supported in GNU mode for backward compatibility.
- The following compiler-specific pragmas are no longer supported:
  — check_printf_formats, check_scanf_formats, and check_vprintf_formats
  — [no_]debug.
- Support for software stack checking is obsolete. The following compiler and assembler options are not supported in this release:
  — --apcs /swst
  — --apcs /noswst
  — --apcs /swstna.

  The following are no longer available:
  — pragma [no_]check_stack
  — predefined macro __APCS_SWST.

  Software stack checked versions of the libraries are no longer supplied.

 Copyright © 2002-2006 ARM Limited. All rights reserved.

2.1.2 Deprecated features

Be aware of the following differences in RVCT v3.0:

- The ARM compiler supports the option `--apcs /adsabi` to compile code that is compatible with the old *ARM Developer Suite*™ (ADS) *Application Binary Interface* (ABI). This is deprecated and will be removed in a future release.
- The following assembler options are deprecated and will be removed in a future release:
  - `--no_cache`
  - `--no_regs` (use `--regnames=none` instead)
  - `--checkreglist` (use `--diag_warning 1206` instead).
- Support for using section attribute qualifiers (`RO`, `RW`, `ZI`, `DBG`) with the linker option `--remove` is deprecated and will be removed in a future release.
- The C++ configuration option `--dll_vtbl` has been replaced by the new option `--export_all_vtbl`. `--dll_vtbl` is deprecated and will not be supported in the future.
- Support for `--memaccess -UL41` is deprecated and will be removed in a future release. `--memaccess -UL41` is replaced by `--no_unaligned_access`.
- The compiler option `--depend_format=unix` replaces `--unix_depend_format`. This option is deprecated and will be removed in a future release.
- The syntax of `fromelf --text` is different in RVCT v3.0. The following forms are deprecated and will be removed in a future release:
  - `fromelf --text=`*xx*
  - `fromelf --text/`*xx*.
- ARM does not recommend the use of the `__user_stack_slop` function. This is deprecated and will be removed in a future release.

Copyright © 2002-2006 ARM Limited. All rights reserved.

2.2 Differences between RVCT v3.0 and RVCT v2.2

RVCT v3.0 overview on page 2-2 describes the most important differences between RVCT v3.0 and RVCT v2.2. This section describes other differences in more detail, and includes:

- *Changes to the ARM compiler*
- *Changes to library support* on page 2-8
- *Changes to the ARM linker* on page 2-9
- *Changes to the ARM assembler* on page 2-9
- *Changes to the fromelf utility* on page 2-11.

2.2.1 Changes to the ARM compiler

Be aware of the following differences in the compiler in RVCT v3.0:

- As part of the new portability features in RVCT v3.0, the new compiler option `--library_interface` specifies that the compiler output works with the RVCT libraries or with any AEABI-compliant library. For example, use `--library_interface=aeabi_glibc` to specify that the output works with an AEABI-compliant version of the GNU C library.
- The ARM compiler now supports Thread Local Storage (TLS) to enable programs to use multiple threads. With TLS, each thread might modify the global variable, but the changes remain visible to the thread only. Two keywords are available:
  - `__declspec(thread)` asserts that variables are thread-local and have thread storage duration
  - `__thread` is equivalent to `__declspec(thread)`.
- ARM processors prior to ARMv7 use the `SWI` or `SVC` instructions to make semihosting calls. However, if you are compiling for the Cortex-M3 processor, semihosting is implemented using the `BKPT` instruction.

  The new`__semihost()` intrinsic enables you to make semihosting calls from C or C++ that are independent of the target architecture.
- The ARM compiler includes support for NEON Technology by providing intrinsics to generate code for the Cortex-A8 processor in both ARM and Thumb state.
- The ARM compiler supports the GCC `builtin` function `__builtin_expect` to provide branch prediction information in situations where feedback directed optimization is not practical. This is available in GNU mode and ARM mode.

 Copyright © 2002-2006 ARM Limited. All rights reserved.

- The ARM compiler includes the `--info totals` option, to display object sizes. Use this option to see the size of the Code and Data (RO Data, RW Data, ZI Data, and Debug Data) in compiled objects.
- The compiler includes the new `--depend_format=string` options for use on Windows. This changes the format of output dependency files to UNIX-style, for compatibility with some UNIX make programs.
- The ARM compiler now recognizes `.cc` extensions for C++ files. Where an unrecognized extension begins with `.c`, a warning is given, for example `filename.cmd`.
- As part of the ARM support for GNU compiler extensions, `armcc` now accepts conditionals in both C and C++ when using the `--gnu` option. The middle operand in a conditional statement can be omitted, if the result is to be the same as the test, for example:

  ```
  i ? i : j
  ```

  This is most useful if the test modifies the value in some way so that `i` is evaluated only once.
- RVCT v3.0 includes compiler and library support for the POSIX function `wcstombs()` to convert a sequence of multibyte characters from one array and store these characters into another array.
- The ARM compiler supports the option `--unaligned_access` (this replaces the deprecated option `--memaccess`).
- The AAPCS no longer requires bitfields to default to unsigned. As a result, a new option, `--signed_bitfields`, has been added to RVCT v3.0 to specify that bitfields are signed. The default is `--unsigned_bitfields`.
- The ARM compiler used to produce `.directive` sections for any symbol whose destination address was unknown. This requirement has been superseded by ELF-standard mechanisms. This means that there might be a change in behavior for dllimported symbols. For example:

  ```
  __declspec(dllimport)  int f();
  int g() { return f(); }
  ```

  does not generate any `.directive` sections.
- The compiler includes the new `__user_setup_stackheap()` function that returns the locations of the initial stack and heap in the same way as `__user_initial_stackheap()`. `__user_setup_stackheap()` is designed for use with Cortex-M3 and gives improved code size (over `__user_initial_stackheap()`) because there is no requirement for a temporary stack.

Copyright © 2002-2006 ARM Limited. All rights reserved.

- The ARM compiler now supports the command-line option `--diag_style gnu` to display messages matching the format reported by `gcc`. This is in addition to the `arm` and `ide` formats available in the previous release.
- The embedded assembler state at the start of each function is now set by the invocation of the compiler, as modified by `#pragma arm` and `#pragma thumb` pragmas.

  You can also change the state of the embedded assembler within a function, for the current function only, by using explicit `ARM`, `THUMB`, or `CODE16` directives in the embedded assembler function.

2.2.2 Changes to library support

Be aware of the following differences in library support in RVCT v3.0:

- The *C Library ABI for the ARM Architecture* [CLIBABI] describes a test of conformance that is checked by the compiler, `_AEABI_PORTABILITY_LEVEL`. Use this to increase the portability of your code to other implementations of the CLIBABI.
- The latest release of the *ABI for the ARM Architecture (base standard)* [BSABI] corrects a defect in the definition of the `assert()` macro and introduces an obligation to signal conformance. This is now defined as `__aeabi_assert()`. See `assert.h` for more details.
- RVCT v3.0 includes library support for the thread-safe string functions `strlcpy` and `strlcat`.
- RVCT v3.0 includes new library implementations of `__user_initial_stackheap()`. This means that it is not necessary to re-implement this function if you are using scatter-loading files.
- A new library mechanism in RVCT v3.0 means that error messages are more informative if symbol clashes are detected when code references `__use_no_*` symbols such as `__use_no_heap` or `__use_no_semihosting`.
- For full POSIX compliance, `armar` no longer accepts command-line options without a preceding `-`. This means, for example, that a command of the form:

  ```
  armar cru lib.a foo.o
  ```

  is faulted. Where you are using build scripts or make files from previous versions of RVCT, these might require changing.
- `armar` now supports the command-line option `--diag_style gnu` to display messages matching the format reported by `gcc`. This is in addition to the `arm` and `ide` formats available in the previous release.

 Copyright © 2002-2006 ARM Limited. All rights reserved.

2.2.3 Changes to the ARM linker

Be aware of the following differences in the linker in RVCT v3.0:

- In the current release, the linker supports Thread Local Storage (TLS) for SVr4 images and shared libraries only. For full details on the linker implementation, see the Addenda to, and Errata in, the ABI for the ARM Architecture [ABI-addenda].
- RVCT v3.0 includes a new linker option, `--compress_debug`, that forces the compression of `.debug_*` sections so removing some redundancy and improving debug table size. However, using the `--compress_debug` option results in longer link times.
- RVCT v3.0 includes a new linker option, `--info libraries`, that prints the full path name of every library automatically selected for the link stage. You can use this option with a modifier, `--info_lib_prefix`, to display information about a specific library.
- RVCT v3.0 includes a new linker option, `--no_legacyalign`, to force the linker to use natural alignment when placing sections.
- RVCT v3.0 includes a new linker option, `--dynamic_debug`, to force the linker to output dynamic relocations for debug sections.
- RVCT v3.0 includes a new linker option, `--show_cmdline`, to see how the linker has processed the command line.
- The ARM linker now supports the command-line option `--diag_style gnu` to display messages matching the format reported by `gcc`. This is in addition to the `arm` and `ide` formats available in the previous release.

2.2.4 Changes to the ARM assembler

Be aware of the following differences in the assembler in RVCT v3.0:

- RVCT v3.0 supports the Thumb-2 Execution Environment (Thumb-2EE) architecture extension to ARMv7. This is based on Thumb-2. The key differences are:
  - new `ENTERX` and `LEAVEX` state changing instructions in both Thumb state and the new ThumbEE state
  - new `HB`, `HBL`, `HBLP`, and `HBP` instructions to branch to handlers
  - null pointer checking on loads and stores
  - an additional `CHKA` instruction in ThumbEE state to check array bound
  - some other modifications to the load, store, and branch instructions (`BX`, `BLX`, and `BXJ`).

Copyright © 2002-2006 ARM Limited. All rights reserved.

- RVCT v3.0 contains initial support for two optional components of ARMv7 architecture, that is, Advanced SIMD extension and VFPv3.

  Advanced SIMD extension consists of:

  — the Advanced SIMD registers. These are the same as the VFPv3 register bank, but are viewed as either thirty-two 64-bit registers or sixteen 128-bit registers.

  — a substantial number of new instructions that treat the Advanced SIMD registers as vectors of elements.

    These new instructions are available in both ARM and Thumb-2 instruction sets.

  VFPv3 has the following enhancements over VFPv2:

  — A substantial extension to the VFPv2 register bank that doubles the earlier VFP register bank to thirty-two 64-bit registers. These registers are in addition to the main ARM general-purpose registers.

  — Several new instructions that provide efficient loading of a substantial number of commonly used floating-point numbers and efficient conversion between floating-point and fixed-point formats.

    These new instructions are available in both ARM and Thumb-2 instruction sets.

- As part of ARM support for Advanced SIMD instructions, bit 27 of the *Floating-Point Status and Control Register* (FPSCR) is now reserved as a cumulative flag to show when saturation occurs in a saturating integer operation.

- The ARM assembler can be used to assemble Wireless MMX Technology instructions to develop code for the PXA270 processor. Two new directives have been introduced and the syntax of load/store instructions has been changed to provide support for symbols and literals.

- Use the following option to list output to a file:

  `--list` *`file`*

  If no *`file`* is given, use `--list=` to send the output to *`inputfile`*`.lst`.

  Note

  You can use `--list` to send the output to a `.lst` file. However, this syntax is deprecated and the assembler issues a warning.

- Two new directives have been introduced:
  — `RELOC`, to encode an ELF relocation in an object file
  — `QN`, to define a name for a specified NEON Quadword register.

 Copyright © 2002-2006 ARM Limited. All rights reserved.

- The ARM assembler now supports the command-line option `--diag_style gnu` to display messages matching the format reported by `gcc`. This is in addition to the `arm` and `ide` formats available in the previous release.
- Local symbols are no longer preserved with `armasm --debug`. You must specify `--keep` if you want to preserve the local symbols to aid debugging.
- In RVCT v2.2, using the LDR pseudo-instruction with local labels in Thumb assembler code resulted in an address where the Thumb bit was not set. If your code expects this behavior, use `armasm --untyped_local_labels`.

2.2.5 Changes to the fromelf utility

Be aware of the following differences in RVCT v3.0:

- The fromelf utility now supports a `--debugonly` option, to remove code and data sections from ELF output files.
- The syntax of `fromelf --text` is different in RVCT v3.0. The following forms are deprecated and will be removed in a future release:
  - `fromelf --text=`*xx*
  - `fromelf --text/`*xx*.
- RVCT v3.0 includes a new `fromelf` option, `--show_cmdline`, to see how the fromelf utility has processed the command line.
- The fromelf utility now supports the command-line option `--diag_style gnu` to display messages matching the format reported by `gcc`. This is in addition to the `arm` and `ide` formats available in the previous release.

Copyright © 2002-2006 ARM Limited. All rights reserved.

2.3 Compatibility of RVCT v3.0 with legacy objects and libraries

This section describes the level of compatibility with legacy objects provided with RVCT v3.0 and the restrictions:

- *Compatibility with legacy RVCT v2.x objects and libraries*.
- *Compatibility with legacy RVCT v1.2/ADS v1.2 objects and libraries*.

2.3.1 Compatibility with legacy RVCT v2.x objects and libraries

Backward compatibility of object/library code is supported. For example, objects built with RVCT v2.x are compatible with RVCT v3.0 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.

2.3.2 Compatibility with legacy RVCT v1.2/ADS v1.2 objects and libraries

The ABI in RVCT v3.0 is different to that in RVCT v1.2 and ADS v1.2. Therefore, legacy RVCT v1.2/ADS v1.2 objects and libraries are not directly compatible with RVCT v3.0.

In RVCT v2.0 and above, the compiler supports the option `--apcs /adsabi` to compile code that is compatible with the old ADS *Application Binary Interface* (ABI). However, this is deprecated and will be removed in a future release.

When linking with legacy RVCT v1.2/ADS v1.2 objects/libraries, be aware of the following:

- To enable RVCT v3.0 C objects to be used with legacy RVCT v1.2/ADS v1.2 C or C++ objects/libraries, compile the RVCT v3.0 C code with `--apcs /adsabi`.
- To enable RVCT v3.0 C++ objects to be used with legacy C RVCT v1.2/ADS v1.2 objects/libraries, compile the RVCTv3.0 C++ code with `--apcs /adsabi`.

  Note

  This can only work if your C++ code does *not* use any of the Rogue Wave C++ libraries, because these libraries are incompatible with objects compiled with `--apcs /adsabi`.

 Copyright © 2002-2006 ARM Limited. All rights reserved.

- There is no compatibility possible between RVCT v3.0 C++ objects/libraries and legacy RVCT v1.2/ADS v1.2 C++ objects/libraries.
- You must link using the RVCT v3.0 linker, not the ADS v1.2 linker, because the ADS v1.2 linker cannot process objects compiled with `--apcs /adsabi`.
- You must link with the RVCT v3.0 ARM-supplied C and C++ runtime libraries, because these have been built in a special way to be compatible with objects compiled with `--apcs /adsabi`.

Copyright © 2002-2006 ARM Limited. All rights reserved.

 Copyright © 2002-2006 ARM Limited. All rights reserved.

Chapter 3
Creating an Application

This chapter describes how to create an application using *RealView® Compilation Tools* (RVCT). It contains the following sections:

Copyright © 2002-2006 ARM Limited. All rights reserved.

3.1 Using the ARM compiler

The ARM compiler, `armcc`, can compile C and C++ source into 32-bit ARM code, or 16-bit Thumb code, or 16/32-bit Thumb-2 code.

Typically, you invoke the ARM compiler as follows:

`armcc [`*`options`*`] `*`ifile_1 ... ifile_n`*

You can specify one or more input files.

3.1.1 Building an example

Sample C source code for a range of applications is installed in the main examples directory. Each example is accompanied by a `readme.txt` file that describes the example code and how to build it.

For example, source code for a simple Dhrystone program is installed in the main examples directory, in `...\dhrystone`. This can be used to measure integer processing performance of a system.

To build the Dhrystone example:

1. Compile the C files `dhry_1.c` and `dhry_2.c` with the following command:

   ```
   armcc -c -W --debug -O2 -Otime --no_inline --no_multifile -DMSC_CLOCK
   dhry_1.c dhry_2.c
   ```

   where:

`-c`	Tells the compiler to compile only (not to link).
`-W`	Tells the compiler to disable all warnings.
`--debug`	Tells the compiler to add debug tables for source-level debugging (`--debug` is a synonym for `-g`).
`-O2`	Tells the compiler to generate code with high optimizations applied.
`-Otime`	Tells the compiler to optimize the code for speed (not space) for this benchmark.
`--no_inline`	Tells the compiler to disable function inlining (required by Dhrystone).
`--no_multifile`	Tells the compiler to disable the multifile compilation optimization, because Dhrystone demands that the two source files are compiled independently.
`-DMSC_CLOCK`	Tells the compiler to use the C library function `clock()` for timing measurements.

 Copyright © 2002-2006 ARM Limited. All rights reserved.

For full details on the options used here, see the chapter describing how to use the ARM compiler in *RealView Compilation Tools v3.0 Compiler and Libraries Guide*.

2. Link the image using the command:

   ```
   armlink dhry_1.o dhry_2.o -o dhrystone.axf --info totals
   ```

   where:

`-o`	Specifies the output file as `dhrystone.axf`.
`--info totals`	Tells the linker to display totals of the Code and Data sizes for input objects and libraries.

3. Use a compatible debugger, for example, RealView Debugger, to load and run the image.

See the `readme.txt` file that accompanies the example for information on the contents of `dhry_1.c` and `dhry_2.c` and how Dhrystone performance is calculated.

3.1.2 Building a Thumb version

To build a Thumb version use:

```
armcc --thumb ...
```

where:

`--thumb`	Tells the compiler to generate Thumb code. (The alternative option, `--arm`, tells the compiler to generate ARM code, and is the default.)

For backwards compatibility, you can still invoke the ARM compiler using one of the tool names supported in earlier compilation tools, for example, compile the C files `dhry_1.c` and `dhry_2.c` with:

```
tcc ...
```

This generates Thumb code for `.c` and `.cpp` files but still generates ARM code for `.ac` and `.acpp` files. If you use `armcc --thumb`, all files are compiled to produce Thumb code.

For full details on compiling for ARM and Thumb, and how the compiler startup configuration is adjusted by the filename extension you specify, see the chapter describing how to use the ARM compiler in *RealView Compilation Tools v3.0 Compiler and Libraries Guide*.

Copyright © 2002-2006 ARM Limited. All rights reserved.

3.1.3 Using the ARM linker

The default output from the linker is a non-relocatable image where the code starts at `0x8000` and the data section is placed immediately after the code. You can specify exactly where the code and data sections are located by using linker options or a scatter-loading description file.

For more information, see:

- `armlink` command syntax in *RealView Compilation Tools v3.0 Linker and Utilities Guide* for a detailed description of the linker options
- the chapter describing scatter-loading description files in *RealView Compilation Tools v3.0 Linker and Utilities Guide*
- the chapter describing how to develop embedded software in *RealView Compilation Tools v3.0 Developer Guide*.

3.1.4 Using fromelf

An executable image in ELF executable format can be converted to other file formats using the ARM `fromelf` utility. For more information, see the chapter describing `fromelf` in *RealView Compilation Tools v3.0 Linker and Utilities Guide*.

Copyright © 2002-2006 ARM Limited. All rights reserved.

3.2 Using the ARM assembler

The basic syntax to use the ARM assembler (`armasm`) is:

`armasm {`*`inputfile`*`}`

For example, to assemble the code in a file called `myfile.s`, type:

```
armasm --debug myfile.s
```

This generates debug tables to produce an object file called `myfile.o`.

For full details on the options and syntax, see *RealView Compilation Tools v3.0 Assembler Guide*.

3.2.1 Building an example

Sample assembly language code is installed in the main examples directory. The examples are accompanied by a `readme.txt` file that describes the examples and how to build them. For example, code for a simple program, `word.s`, is installed in the main examples directory, in `...\asm`.

To build the example:

1. Assemble the source file using the command:

   ```
   armasm --debug word.s
   ```

2. Link the file using the command:

   ```
   armlink word.o -o word.axf
   ```

3. Use a compatible debugger, for example, RealView Debugger, to load and test the image.

   Step through the program and examine the registers to see how they change (see your debugger documentation for details on how to do this).

Copyright © 2002-2006 ARM Limited. All rights reserved.

3.3 Using an IDE

An *Integrated Development Environment* (IDE) enables you to use a graphical interface to manage your software development projects. You can invoke the ARM tools to develop C, C++, and assembly language programs through an IDE. This enables you to configure the ARM tools to compile, assemble, and link your project code.

ARM has also licensed the CodeWarrior IDE from Metrowerks and is making this available with RealView Development Suite. However, this is not available on Sun Solaris or Red Hat Linux platforms. CodeWarrior IDE example projects are provided with RealView Development Suite in the main examples directory *install_directory*\RVDS\Examples.

See *RealView Development Suite Getting Started Guide* for more details on using IDEs.

 Copyright © 2002-2006 ARM Limited. All rights reserved.

Appendix A
About Previous Releases

This appendix describes the major differences between previous releases of *RealView® Compilation Tools* (RVCT).

It contains the following sections:

Copyright © 2002-2006 ARM Limited. All rights reserved.

A.1 Differences between RVCT v2.2 SP1 and RVCT v2.2

This section describes the differences between RVCT v2.2 delivered with RVDS v2.2 Service Pack 1 (SP1) and the previous release.

The differences were:

- RVCT v2.2 SP1 includes support for the ARM1136J(F)-S-rev-1 core. To see a full list of supported cores use:

  ```
  armcc --cpu list
  ```

- The ARM linker offers new options when building shared objects, that is, use:
  - `--pt_arm_exidx` to create a `PT_ARM_EXIDX` program header that describes the location of the exception tables. The linker assumes that a shared object containing a `PT_ARM_EXIDX` program header might throw an exception.
  - `--force_so_throw` to force the linker to assume that all shared objects might throw an exception.

  For full details of these command-line options, see the chapter describing System V shared libraries in *RealView Compilation Tools v3.0 Linker and Utilities Guide*.

- The preprocessing directive `#warning` is supported. This produces a user-defined warning at compilation time but does not halt compilation.

- The keywords **`__restrict`** and **`__restrict__`** are supported as synonyms for **`restrict`**. In this release, both keywords are available in all modes (not just with `--restrict`).

- The `deprecated` function attribute is supported in both GNU mode and ARM mode:

  ```
  int Function_Attributes_deprecated_0(int b) __attribute__ ((deprecated));
  ```

- The fromelf utility now supports a `--no_comment_section` option, to remove the `.comment` section from ELF output files.

- The storage type selection of the compiler option `--enum_is_int` has changed. If the range of the enumerator is greater than the range of a **`signed int`**, but is within the range of an **`unsigned int`**, the storage type of the **`enum`** is now **`unsigned int`**.

  For example:

  ```
  enum E { k = 0x80000000 };
  ```

  is treated as an **`unsigned int`**. In previous versions of RVCT a warning is generated.

- As part of the development of the ARM assembly language, the `SWI` instruction has been renamed to `SVC` (SuperVisor Call).

 Copyright © 2002-2006 ARM Limited. All rights reserved.

Instructions disassemble to SVC, with a comment to say that this was formerly SWI. For example, `fromelf --text -c` produces:

```
...
0x00000fbc:    e3a00010    ....    MOV      r0,#0x10
0x00000fc0:    ef123456    V4..    SVC      0x123456 ; formerly SWI
...
```


Copyright © 2002-2006 ARM Limited. All rights reserved.

A.2 Differences between RVCT v2.2 and RVCT v2.1

This section describes the differences between RVCT v2.2 and RVCT v2.1, and includes:

- *General changes*
- *Changes to the ARM compiler* on page A-8
- *Changes to library support* on page A-10
- *Changes to the ARM linker* on page A-12
- *Changes to the ARM assembler* on page A-14
- *Changes to the fromelf utility* on page A-15.

A.2.1 General changes

The following changes were made in RVCT v2.2:

- RVCT v2.2 supports the new Thumb®-2 instruction set. Thumb-2 introduces many new 32-bit instructions, and some new 16-bit instructions. Thumb-2 gives near ARM performance together with code size that is similar to the original Thumb on those processors that support it, such as the ARM1156T2(F)-S.

  The Thumb-2 instruction set includes older 16-bit Thumb instructions as a subset.

- RVCT v2.2 includes support for new ARMv6 cores, for example, the ARM1176JZ(F)-S, incorporating ARM TrustZone™ technology-optimized software, the ARM968EJ-S, the ARM1156T2(F)-S, and the ARM MPCore™.

  The RVCT v2.2 assembler provides support for MPCore instructions.

- RVCT v2.2 includes support for a new assembler syntax, while continuing to support the old syntax to enable the assembly of legacy code.

- RVCT v2.2 is fully compliant with the *Base Platform ABI for the ARM Architecture* [BPABI], so giving support for a range of operating systems, for example, ARM Linux and Symbian OS.

- RVCT v2.2 provides initial support for DWARF 3 (Draft Standard 9) debug tables, as described in the *ABI for the ARM Architecture (base standard)* [BSABI]. A new command-line option, `--dwarf3`, is now available to specify this format when compiling code.

- The command-line option `--debug` or `-g` switches on the generation of debug tables for the current compilation. Optimization options are specified by `-O`*num*. By default, using the `--debug` or `-g` option does not affect the optimization setting.

 Copyright © 2002-2006 ARM Limited. All rights reserved.

This is a change in behavior for RVCT v2.2 (`--debug` or `-g` alone was equivalent to `[--debug|-g] -O0` in RVCT v2.1). Apart from this new default, there is no change in the behavior of the optimization options, that is, `-O`*num*, `-Ospace`, or `-Otime`.

- RVCT v2.2 supports the command-line option `--apcs /fpic` to compile code that is compatible with System V shared libraries. Use this option to generate read-only position-independent code where relative address references are independent of the location where your program is loaded.
- The ARM linker supports building, and linking against, shared libraries. New command-line options are available to build SVr4 executable files and shared objects, and to specify how code is generated.
- The ARM linker conforms to the *Base Platform ABI for the ARM Architecture* [BPABI] and supports the GNU-extended symbol versioning model.
- The ARM implementation of floating-point computations has been changed to provide improved support for C99 functions. Where this changes behavior significantly, a *compatibility mode* has been introduced to aid developers to migrate code to use the new features. See *Changes to library support* on page A-10 for more information.
- The ARM compiler C implementation has been changed so that it now issues a Warning for out-of-range enumerator values. Such values are treated the same way as in C++. This means the size of **`enum`** types with out-of-range values might be different in C when you upgrade to the latest release of RVCT.

  There is no change if you are using C++, or the command-line options `--enum_is_int`, `--strict`, or `--strict_warnings`. See *Changes to the ARM compiler* on page A-8 for more information.
- The ARM libraries have been enhanced to provide improved support for multithreading. This is designed to help developers who are working with RTOS-based systems.
- The ARM compiler offers new options to provide greater control of how dynamic symbols are exported:
  - `--dllexport_all`
  - `--no_hide_all`.
- The ARM linker can perform some branch optimizations that are not available to other components of RVCT. Two new command-line options are available to handle tail calling sections to optimize the branch instruction at the end of the section.

Copyright © 2002-2006 ARM Limited. All rights reserved.

- Options that were deprecated in RVCT v2.1 are now obsolete in v2.2 (see *Obsolete features*).
- Some options that were supported in RVCT v2.1 are now deprecated in v2.2 (see *Deprecated features* on page A-7).

Obsolete features

The following changes were made in RVCT v2.2:

- All features and options that were deprecated in RVCT v2.1 are obsolete in RVCT v2.2, that is:
  - Legacy *Software Development Toolkit* (SDT) formats such as *ARM Object Format* (AOF) and libraries in *ARM Library Format* (ALF) format.
  - The use of single dashes for keywords, for example, `armlink -help`.
  - The compiler options `-ansi` and `-ansic`.
- Old compiler option names that were deprecated in RVCT v2.1 are obsolete in RVCT v2.2, for example, `-fy`, `-fd`, `-Ec`, and `-zo`,

  The compiler issues a warning where a preferred option name exists, for example:

  ```
  armcc -zo
  Warning: X0010W: Old syntax, please use '--split_sections'.
  ```

  See Appendix E in *RealView Compilation Tools v3.0 Compiler and Libraries Guide*for a list.
- Selected older ARM processors and architectures are obsolete:
  - ARM6
  - ARMv3 and ARMv3M.

  To see a full list of supported cores use:

  ```
  armcc --cpu list
  ```
- Selected older floating-point architectures are obsolete:
  - VFPv1 (the default is VFPv2)
  - FPA
  - Soft FPA.

  Unless specified, the default is Soft VFP.

  To see a full list of supported floating-point architectures use:

  ```
  armcc --fpu list
  ```

 Copyright © 2002-2006 ARM Limited. All rights reserved.

- The following compiler options are obsolete:
  - `--fpu fpa`, `--fpu softfpa`, and `--fpu vfpv1`
  - `--fa`
  - `--cpu 3` and `--cpu 3M`
  - `-Oldrd` and `-Ono_ldrd`
  - `-W`*`letter`* and `-E`*`letter`*.
- Synonyms for the `SXT` and `UXT` instructions (sign extend or zero extend) are not supported.
- Use of FPA registers `f0-f7` and `F0-F7` is obsolete.
- The use of the environment variable `RVCT21_CLWARN` to warn against deprecated options is not supported.

Deprecated features

The following changes were made in RVCT v2.2:

- The compiler supports the option `--apcs /adsabi` to compile code that is compatible with the old *ARM Developer Suite*™ (ADS) *Application Binary Interface* (ABI). This is deprecated and will be removed in a future release.
- The ARM linker and `fromelf` accept two forms of negated option, for example, `--no_debug` and `--nodebug`. However, the non-standard form, for example, `--nodebug`, is deprecated and will not be supported in the future. A warning is issued if you use the deprecated syntax.
- The C++ configuration option `--dll_vtbl` has been replaced by the new option `--export_all_vtbl`. The option `--dll_vtbl` is deprecated and will not be supported in the future.
- The RVCT assembler supports two forms of the Load Register EXclusive instruction:
  - `LDREX{B|D|H}{`*`cond`*`} `*`Rd`*`, [`*`Rn`*`]`
  - `LDR{B|D|H}EX{`*`cond`*`} `*`Rd`*`, [`*`Rn`*`]`

  The second is deprecated and will be removed in a future release.

  The disassembler supports only the first form.
- The RVCT assembler supports two forms of the Store Register EXclusive instruction:
  - `STREX{B|D|H}{`*`cond`*`} `*`Rd`*`, [`*`Rn`*`]`
  - `STR{B|D|H}EX{`*`cond`*`} `*`Rd`*`, [`*`Rn`*`]`

  The second is deprecated and will be removed in a future release.

Copyright © 2002-2006 ARM Limited. All rights reserved.

The disassembler supports only the first form.

A.2.2 Changes to the ARM compiler

The following changes were made in RVCT v2.2:

- The ARM compiler offers new options to provide greater control of how dynamic symbols are exported, that is, use:
  - `--export_all_vtbl` to export all virtual table functions and RTTI for classes with a key function
  - `--export_defs_implicitly` to export definitions where the prototype was marked `dllimport`.
- The ARM compiler offers new options to give symbol visibility when building shared objects or DLLs, that is, use:
  - `--dllexport_all` to provide dynamic visibility of all global symbols unless specified otherwise
  - `--no_hide_all` to export all `extern` definitions and import all undefined references.
- The new `__swi_indirect_r7` behaves like `__swi_indirect_r12` except that it uses `r7` instead of `r12`. Thumb applications on ARM Linux use `__swi_indirect_r7` to make kernel syscalls.
- The behavior of `--force_new_nothrow` has been enlarged so that it causes any overloaded global `operator new` to be treated as `throw()`.
- The ARM compiler C implementation has been changed so that it now issues a Warning for out-of-range enumerator values.

  In strict C, enumerator values must be representable as **`int`**s, for example, they must be in the range -2147483648 to +2147483647 (inclusive). In previous releases of RVCT out-of-range values were cast to **`int`** without a warning (unless you specified the `--strict` option).

  In RVCT v2.2, such values are treated the same way as in C++, that is, they are treated as **`unsigned int`**, **`long long`**, or **`unsigned long long`**. This means the size of **`enum`** types with out-of-range values might be different in C if you are using the latest release of RVCT. For example:

```
enum E1 { k1 = 0xffffffff };  /* value == 2147483648u; out-of-range in C */
    /* C: before 2.2: sizeof(enum E1) == 1 */
    /* C:        2.2: sizeof(enum E1) == 4 */
    /* C++:       all: sizeof(enum E1) == 4 */
enum E2 { k2 = (int)0xffffffff };  /* value == -1; in range */
    /* C: before 2.2: sizeof(enum E1) == 1 */
```


Copyright © 2002-2006 ARM Limited. All rights reserved.

```
    /* C:        2.2: sizeof(enum E1) == 1 */
    /* C++:      all: sizeof(enum E1) == 1 */
enum E3 { k3 = -1, k4 = 0xffffffff };  /* value == 2147483648u; out-of-range in C */
    /* C: before 2.2: sizeof(enum E1) == 1 */
    /* C:        2.2: sizeof(enum E1) == 8; use long long */
    /* C++:      all: sizeof(enum E1) == 8; use long long */
```

To ensure that out-of-range Warnings are reported, use the following command to change them into Errors:

```
armcc --diag_error 66 ...
```

There is no change if you are using C++, or the command-line options `--enum_is_int`, `--strict`, or `--strict_warnings`.

- The ARM compiler now supports the use of the `__attribute__` keyword in both ARM and GNU mode.
- The ARM compiler now supports a new variable attribute, `zero_init` or `__zero_init__`, to specify that a variable with no initializer is placed in a ZI data section.
- The ARM compiler provides a new option, `--dwarf3`, to specify DWARF 3 standard debug tables to describe ARM or Thumb programs when compiling code written in C or C++. If you do not specify a format, the compiler assumes DWARF 2.
- The ARM compiler support a new environment variable `RVCT22_CCOPT` that enables you to specify command-line options to the compiler.
- RVCT v2.2 supports the command-line option `--apcs /fpic` to compile code that is compatible with System V shared libraries. Use this option to generate read-only position-independent code where relative address references are independent of the location where your program is loaded.
- The ARM compiler provides new intrinsics to control interrupt handling:
  - `__enable_irq()` and `__disable_irq()`
  - `__enable_fiq()` and `__disable_fiq()`.
- The ARM compiler provides new intrinsics to control optimization:
  - `__schedule_barrier()`
  - `__force_stores()`
  - `__memory_changed()`.

Copyright © 2002-2006 ARM Limited. All rights reserved.

- The ARM compiler includes new options to enable or disable implicit determination, from context, whether a template parameter dependent name is a type or nontype, that is, use:
  - `--implicit_typename`
  - `--no_implicit_typename`.

  The default is `--no_implicit_typename`.
- When you invoke the compiler, you can use the `--show_cmdline` option to see how the compiler has processed the command line. The commands are shown normalized, and the contents of any via files are expanded. However, using this option does not trap accidental errors in your command line.
- The ARM compiler is no longer guaranteed to be ISO C and C++ standard-compliant with respect to floating-point arithmetic at optimization level `-O3`. You must use the `--fpmode=std` option to ensure ISO C and C++ compliance.
- The ARM compiler now places **`const volatile`** (and **`volatile const`**) data in RW or ZI sections, as mandated by the ISO C standard. Previously such data was placed in RO sections. This might affect your code if you have made assumptions about the placement of data by the compiler.

A.2.3 Changes to library support

The following changes were made in RVCT v2.2:

- The ARM implementation of floating-point computations has been changed to provide improved support for C99 functions. Where this changes behavior significantly, a *compatibility mode* has been introduced to aid developers to migrate code to use the new features. C99 functions that are new in fplib (or where behavior has changed) are:
  - `ilogb`, `ilogbf`, `ilogbl`
  - `logb`, `logbf`, `logbl`
  - `scalbn`, `scalbnf`, `scalbnl`, `scalbln`, `scalblnf`, `scalblnl`
  - `nextafter`, `nextafterf`, `nextafterl`, `nexttoward`, `nexttowardf`, `nexttowardl`.

  C99 functions that are new in mathlib (or where behavior has changed) are:
  - `fpclassify` and `signbit`
  - `isfinite`, `isinf`, `isnan`, and `isnormal`
  - `copysign`, `copysignf`
  - `isgreater`, `isgreaterequal`, `isless`, `islessequal`, `islessgreater`, and `isunordered`.

 Copyright © 2002-2006 ARM Limited. All rights reserved.

To help you to port your code, the new compatibility mode emulates the previous behavior of these functions and macros:

— `ilogb`, `ilogbf`, `ilogbl`

— `finite`

— `isnan`.

Note

This legacy support will be removed in a future release. ARM recommends that you migrate your use of these features to the equivalent functions in the latest release of the compiler.

- The ARM libraries have been enhanced to provide improved support for multithreading. This is designed to help developers who are working with RTOS-based systems.

  The user-overridable function `__user_libspace()` has been split into two wrapper functions:

  __user_perproc_libspace()

  This returns a pointer to the `__user_libspace` data area used to store data that is global to an entire process, that is, data shared between all threads.

  __user_perthread_libspace()

  This returns a pointer to the `__user_libspace` data area used to store data that is local to a particular thread.

  There are also three new user-overridable functions to manage the locking mechanisms used to prevent corruption of shared data due to concurrent access:

  _mutex_initialize()

  This accepts a pointer to a 32-bit word and initializes it as a valid mutex.

  ```
  int _mutex_initialize(mutex *m);
  ```

  _mutex_acquire()

  This causes the calling thread to obtain a lock on the supplied mutex.

  ```
  void _mutex_acquire(mutex *m);
  ```

  _mutex_release()

  This causes the calling thread to release the supplied mutex.

  ```
  void _mutex_release(mutex *m);
  ```

  This has resulted in a change in the way some functions behave in a multi threaded environment, making it easier for developers to use these functions in multiprocess systems.

Copyright © 2002-2006 ARM Limited. All rights reserved.

- The ARM librarian, `armar`, supports the command-line option `--diag_style` to specify how diagnostic messages are displayed, for example, to include the line number and character count, use:

  ```
  armar --diag_style ide
  ```

  The default is ARM format, that is, `--diag_style arm`.

A.2.4 Changes to the ARM linker

The following changes were made in RVCT v2.2:

- The ARM linker conforms to the *Base Platform ABI for the ARM Architecture* [BPABI] and supports the GNU-extended symbol versioning model. The linker offers new options to control symbol versioning, that is, use:
  - `--symver_script` *`file`* to turn on implicit symbol versioning and input *file* as a symbol version script.
  - `--symver_soname` to turn on implicit symbol versioning and version symbols in order to force static binding.

    Where a symbol has no defined version, the linker uses the `SONAME` of the file being linked.

  Note

  In general, symbol versioning is only useful when producing or linking against a DSO or shared library. It has no effect on static linking.

- The ARM linker can perform some branch optimizations that are not available to other components of RVCT.

  Two new command-line options control these optimizations, that is, use:
  - `--inline` to enable branch inlining. By default, inlining is off.
  - `--info inline` to display information each time a function is inlined and to see the total number of inlines.

  When you specify both `--inline` and `--feedback` *`file`*, functions that are inlined by the linker are also emitted in the feedback file.

- Two new command-line options are available to handle tail calling sections to optimize the branch instruction at the end of the section, that is, use:
  - `--tailreorder` to move tail calling sections above their target, if possible
  - `--info tailreorder` to display information about tail calling sections that have been moved.

 Copyright © 2002-2006 ARM Limited. All rights reserved.

- Two new command-line options enable you to control veneer generation in the ARM linker, that is, use:
  - `--no_inlineveneer` to disable inline veneers
  - `--no_veneershare` to prevent veneer sharing.
- The ARM linker supports building, and linking against, shared libraries. New command-line options are available to build SVr4 and BPABI executable files and shared objects, BPABI DLLs, and to specify how code is generated, that is, use:
  - `--sysv` to build an SVr4 formatted ELF file
  - `--shared` to build an SVr4 shared object
  - `--soname` *name* to specify the `SONAME` for a shared object
  - `--fpic` to link position-independent code
  - `--init` *symbol* to specify initialization code
  - `--fini` *symbol* to run code when unloading an executable file or shared object
  - `--linux_abitag` *id* to specify the minimum compatible Linux kernel version
  - `--dynamiclinker` *name* to change the default dynamic linker.
- A new linker option is available, `--startup`, to use alternative C libraries with a different startup symbol. Similarly, the `--cppinit` option is available for C++ initialization code.
- The ARM linker option `--symbols` lists local and global symbols used in the link step. In RVCT v2.2, this output no longer lists mapping symbols by default. A new command-line option, `--list_mapping_symbols`, now enables you to include mapping symbols in the output produced by `--symbols`.
- If the linker detects objects that specify ARMv3 (obsolete in RVCT v2.2), it upgrades these to ARMv4 to be usable with ARM libraries. The linker displays a warning message where it raises the target architecture level.
- The ARM linker enables you to refer to an input section by symbol name in your scatter load description files. See the description of *input_symbol_pattern* in *RealView Compilation Tools v3.0 Linker and Utilities Guide*.

Copyright © 2002-2006 ARM Limited. All rights reserved.

A.2.5 Changes to the ARM assembler

The following changes were made in RVCT v2.2:

- ARMv6T2 defines Thumb-2, a major enhancement of the Thumb instruction set. Thumb-2 provides almost exactly the same functionality as the ARM instruction set. It has both 16-bit and 32-bit instructions, and achieves ARM-like performance with Thumb-like code density.

  ARMv6T2 also defines several new instructions in the ARM instruction set.

  The assembler supports all the new instructions in both ARM and Thumb-2.

- RVCT v2.2 includes support for new ARMv6 architecture extensions:
  - ARMv6Z defines ARM Security Extensions (TrustZone), as used in the ARM1176JZ(F)-S core.
  - ARMv6K defines instructions for *Symmetric Multiprocessor* systems (SMP), as used in the ARM MPCore.

- The RVCT v2.2 assembler enables you to write source code that can be assembled to either ARM or Thumb-2 instructions. You can use the same language to write Thumb instructions for pre-Thumb-2 processors.

  However, RVCT v2.2 also supports the old assembly language syntax to enable the assembly of legacy code.

- The ARM assembler supports the `COMMON` directive to allocate a block of memory, of the defined size, at the specified symbol. You can also specify how the memory is aligned:

  `COMMON` *`symbol`*`{,`*`size`*`{,`*`alignment`*`}}`

- The ARM assembler provides a new option, `--dwarf3`, to specify DWARF 3 standard debug tables. DWARF 2 remains the default.

- The ARM assembler offers new options to give symbol visibility when building shared objects or DLLs, that is, use:
  - `--dllexport_all` to provide dynamic visibility of all global symbols unless specified otherwise
  - `--no_hide_all` to export all `extern` definitions and import all undefined references.

- The ARM assembler can output ELF symbols with a visibility set through the use of new attributes to the `IMPORT` and `EXPORT` directives:
  - `DYNAMIC`
  - `HIDDEN`
  - `PROTECTED`

Copyright © 2002-2006 ARM Limited. All rights reserved.

A.2.6 Changes to the fromelf utility

The following change was made in RVCT v2.2:

- The fromelf utility has been enhanced to support GNU-extended symbol versioning tables in the output from `--text=/s`. Use the new `--no_symbolversions` option to turn off the decoding of symbol version tables.

Copyright © 2002-2006 ARM Limited. All rights reserved.

A.3 Differences between RVCT v2.1 and RVCT v2.0

This section describes the differences between RVCT v2.1 and RVCT v2.0, and includes:

- *General changes*
- *Changes to the ARM compiler* on page A-17
- *Changes to library support* on page A-18
- *Changes to the ARM linker* on page A-18
- *Changes to the ARM assembler* on page A-19
- *Changes to the fromelf utility* on page A-19.

A.3.1 General changes

The following changes were made in RVCT v2.1:

- Full support for C++, including exceptions.
- Enhanced support for ARMv6 cores. To see a full list of supported cores use:

  ```
  armcc --cpu list
  ```

- The compiler, linker, assembler, and `fromelf` support the new `--diag_style` option to generate warnings and errors in a format that is more compatible with IDEs, such as Microsoft Visual Studio.
- The compiler and linker support the new facility to remove unused virtual functions from generated C++ code.
- Linker feedback is available, for the next time a file is compiled, to inform the compiler about unused functions. These are placed in their own sections for future elimination by the linker.
- The ARM® tools can share suitable strings across compile units using `SHF_STRINGS` sections, as defined by the *ELF for the ARM Architecture* standard [AAELF].
- Support for VFPv1 is deprecated. The new default is VFPv2. To see a full list of supported FPUs use:

  ```
  armcc --fpu list
  ```

- The use of single dashes for keywords is deprecated and will not be supported in the future. Use double dashes when working with the compilation tools.
- By default, the compilation tools warn against the use of deprecated options (such as the compiler option `--fpu softfpa`).

Copyright © 2002-2006 ARM Limited. All rights reserved.

In RVCT v2.1, change this behavior by setting the environment variable `RVCT21_CLWARN` to one of the following values:

0 Warn against old syntax and deprecated options.

1 Accept old syntax without a warning, but warn against deprecated options. This is the default.

2 Accept old syntax and deprecated options without a warning.

A.3.2 Changes to the ARM compiler

The following changes were made in RVCT v2.1:

- Support for GNU extensions when you run the compiler with the `--gnu` option. However, some extensions are also supported when you run the compiler without this option. These compilation modes are referred to as:

  ARM mode

  The default mode, that is, compilations without the `--gnu` option.

  GNU mode

  Compilations with the `--gnu` option.

  For a complete list of all GNU extensions, and the mode and language in which they are supported, see the chapter describing the compiler reference in *RealView Compilation Tools v2.1 Compiler and Libraries Guide*.

- Enhanced support for ISO C++ through the Edison Design Group (EDG) front end. This provides a full C++ parser that passes a program representation to the ARM compiler for code generation. This now includes support for throwing and catching C++ exceptions.

- Multifile compilation provides optimization across compile units. Use the new `--multifile` option to specify this behavior. Multifile compilation requires you to specify multiple files on the command line, for example:

  `armcc [options] --multifile ifile_1 ... ifile_n`

- New `-O3` optimization level, that includes multifile compilation by default.

- New `--cpu list` and `--fpu list` options to display details about supported CPUs and architectures.

- New `__breakpoint()` intrinsic.

- Noreturn functions.

Copyright © 2002-2006 ARM Limited. All rights reserved.

A.3.3 Changes to library support

The following changes were made in RVCT v2.1:

- The C++ libraries, that is, the Rogue Wave and C++ runtime libraries, now support C++ exceptions. The C++ libraries continue to support applications that do not require exceptions support.
- The C library now supports all of the `wchar.h` function, except file I/O, and the c99 hexadecimal floating-point support in `printf` and `scanf`.
- A new region tables format has been introduced to support compression algorithms. This new format no longer contains `ZISection$$Table`.

A.3.4 Changes to the ARM linker

The following changes were made in RVCT v2.1:

- Read/Write data compression is enabled by default to optimize ROM size.
- A new option is available, `--rosplit`, to output two RO execution regions, one for code and one for data.
- New command-line options are available to support C++ exception tables. Use the new option `--noexceptions` to ensure that your code is exceptions free.

  The new option `--exceptions_tables=unwind|nounwind` forces the linker to generate exception tables regardless of the content of the input files. The linker can create exception tables for C and assembly language objects with debug frame information using, for example, `--exceptions_tables=unwind`.
- A new option is available, `--userlibpath`, to specify where to search for user libraries.
- The linker is now stricter in checking alignment in object files. It ensures that any code that requires eight-byte alignment of the stack is only called, directly or indirectly, by code that preserves eight-byte alignment of the stack. The linker generates an error message if a stack alignment conflict is detected:

  `Error L6238E: `*`object_name.o(section_name)`* `contains invalid call from '~PRES8' function to 'REQ8' `*`function_name`*

  A similar warning message is generated where the address of an external symbol is referenced:

  `Warning L6306W: '~PRES8' section `*`object_name.o(section_name)`* `should not use the address of 'REQ8' `*`function_name`*

 Copyright © 2002-2006 ARM Limited. All rights reserved.

A.3.5 Changes to the ARM assembler

The following changes were made in RVCT v2.1:

- New `--cpu list` and `--fpu list` options to display details about CPU and architectures supported.
- The assembler includes the new `:RCONST:` unary operator to return the number of a given register.
- The assembler examines instructions that modify the Stack Pointer (SP) to decide whether code should be marked as `PRES8`. Where required, this change is made automatically (see the chapter describing the directives reference in *RealView Compilation Tools v2.1 Assembler Guide*).
- The assembler can give warnings about possible interlocks in your code. To enable this, use:

  ```
  armasm --diag_warning 1563
  ```

A.3.6 Changes to the fromelf utility

The following change was made in RVCT v2.1:

- New `--expandarrays` option to decode an ELF image so that arrays are expanded both inside and outside structures.

  This option can only be used in conjunction with the `--text=/a` option.

Copyright © 2002-2006 ARM Limited. All rights reserved.

A.4 Differences between RVCT v2.0 and RVCT v1.2

This section describes the differences between RVCT v2.0 and RVCT v1.2, and includes:

- *General changes*
- *Changes to the ARM compiler*
- *Changes to the ARM linker* on page A-21
- *Changes to the ARM assembler* on page A-22.

A.4.1 General changes

The following changes were made in RVCT v2.0:

- Support for ARM Architecture v6.
- Compliance with the *ABI for the ARM Architecture (base standard)* [BSABI]. For more information, see `http://www.arm.com`.
- For floating-point exceptions to occur, you must select `--fpmode ieee_full`. This is because the default setting is now `--fpmode std` and so floating-point exceptions are not generated by default.
- Support for double dashes "--" to indicate command-line keywords (for example, `--cpp`).

A.4.2 Changes to the ARM compiler

The following changes were made to the ARM compiler (`armcc`):

- There is a new front-end to the RVCT v2.0 compiler that includes changes to the command-line options. The options available in the older ARM compilers are supported for backwards compatibility.
- The four individual compilers, `armcc`, `tcc`, `armcpp` and `tcpp`, are now merged into a single compiler, `armcc`. However, to aid migration to the new compiler, you can invoke the RVCT v2.0 compiler using the individual compiler names.
- Support for ARMv6, and exploits the unaligned access behavior of ARMv6.
- A new embedded assembler to complement the inline assembler.
- ARM and Thumb® compilation on a per-function basis, using `#pragma arm` and `#pragma thumb`.
- Five floating-point models using the `--fpmode` option.

 Copyright © 2002-2006 ARM Limited. All rights reserved.

- The behavior of the `--list` option is different from that in the older compilers.
- C++ template instantiation.
- C++ namespaces.
- You can specify the level of pointer alignment.
- ROPI is no longer supported when compiling C++.
- Control and manipulation of diagnostic messages. Also, the numbering of diagnostic messages has changed. Messages now have the number format `#nnnn` or `#nnnn-D`. The message numbers for messages with the `-D` suffix can be used in those options that enable you to manipulate the diagnostic messages.
- Many old compiler options are not supported in the new interface. However, for backwards compatibility, these options are available if you use the `--old_cfe` option. See the appendix describing the older compiler options in *RealView Compilation Tools v2.0 Compiler and Libraries Guide* for more details. Where applicable, this appendix also shows how the old compiler options map to the new compiler options. For those messages listed in *RealView Compilation Tools v2.0 Compiler and Libraries Guide*, the appendix also shows the equivalent messages that are output by the new compiler interface.

  Note

  If you use the `--old_cfe` option, then the older numbering format is used for messages output by the compiler.

Other changes include the addition of new pragmas and predefined macros, additional C and C++ language extensions, and changes to the ARM C and C++ libraries.

A.4.3 Changes to the ARM linker

The following changes were made to the ARM linker (`armlink`):

- The `--unresolved` option is now applicable to partial linking.
- A new steering file command, `RESOLVE`, has been added, and is used when performing partial linking. `RESOLVE` is similar in use to the `armlink` option `--unresolved`.
- The option `--edit` now accepts multiple files.
- There is a new option `--pad` to specify a value for padding bytes.
- New scatter-loading attributes, `EMPTY` and `ZEROPAD`, have been added.

Copyright © 2002-2006 ARM Limited. All rights reserved.

A.4.4 Changes to the ARM assembler

The following changes were made to the ARM assembler (`armasm`):

- Support for new ARM architecture v6 instructions has been added. These include saturating instructions, parallel instructions, and packing and unpacking instructions.
- The `ALIGN` directive has an additional parameter, to specify the contents of any padding. This parameter is optional.
- There is a new `AREA` directive, `NOALLOC`.
- There are two new directives, `ELIF` and `FRAME RETURN ADDRESS`.
- There are four new built-in variables `{AREANAME}`, `{COMMANDLINE}`, `{LINENUM}`, and `{INPUTFILE}`.

Copyright © 2002-2006 ARM Limited. All rights reserved.