RealView® Compilation Tools

Version 4.0

Essentials Guide

ARM®

ARM DUI 0202J (ID101213)

RealView Compilation Tools
Essentials Guide

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 RealView Developer Suite
January 2004	D	Non-Confidential	Release 2.1 for RealView Developer Suite
December 2004	E	Non-Confidential	Release 2.2 for RealView Developer Suite
May 2005	F	Non-Confidential	Release 2.2SP1 for RealView Developer Suite
March 2006	G	Non-Confidential	Release 3.0 for RealView Development Suite
March 2007	H	Non-Confidential	Release 3.1 for RealView Development Suite
September 2008	I	Non-Confidential	Release 4.0 for RealView Development Suite
23 January 2009	I	Non-Confidential	Update 1 for RealView Development Suite v4.0
10 December 2010	J	Non-Confidential	Update 2 for RealView Development Suite v4.0

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.

Unrestricted Access is an ARM internal classification.

Product Status

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

Web Address

`http://www.arm.com`

Contents
RealView Compilation Tools Essentials Guide

Preface

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

About this book

This book provides an overview of the ARM® RealView® Compilation Tools (RVCT).

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 ***Creating an Application***

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

Chapter 3 ***Differences between RVCT v4.0 and RVCT v3.1***

Read this chapter for information on the differences between the latest release and the previous release of RVCT.

Appendix A ***About Previous Releases***

Read this appendix for information on the differences between previous releases of RVCT.

This book assumes that the ARM software is installed 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.

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://infocenter.arm.com/help/index.jsp` for current errata sheets and addenda, and the *ARM Frequently Asked Questions* (FAQs).

ARM publications

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

- *RealView Compilation Tools Compiler User Guide* (ARM DUI 0205). This book describes the basic functions and compiler-specific features of the ARM compiler, `armcc`. It also provides you with an understanding of the NEON™ vectorizing compiler and explains how to take advantage of the automatic vectorizing features.
- *RealView Compilation Tools Compiler Reference Guide* (ARM DUI 0348). This book provides reference information for the ARM compiler, and describes the command-line options. It also gives reference material on the ARM implementation of C and C++ in the compiler.

- *RealView Compilation Tools Libraries and Floating Point Support Guide* (ARM DUI 0349). This book describes the ARM C and C++ libraries, compliance with the ISO standard, tailoring to target-dependent functions, and application-specific requirements. It also describes the ARM support for floating-point computations.
- *RealView Compilation Tools Assembler Guide* (ARM DUI 0204). This book provides reference and tutorial information on the ARM assembler, `armasm`.
- *RealView Compilation Tools Linker User Guide* (ARM DUI 0206). This book provides user information on the ARM linker, `armlink`. It also gives an overview of scatter-loading.
- *RealView Compilation Tools Linker Reference Guide* (ARM DUI 0381). This book provides reference information on the command-line options and steering files. It also describes the *Base Platform ABI for the ARM Architecture* (BPABI) and System V shared libraries and executables.
- *RealView Compilation Tools Utilities Guide* (ARM DUI 0382). This book provides information on the command-line options and working examples for the ARM librarian, `armar` and the ARM image conversion utility, `fromelf`.
- *RealView Compilation Tools Developer Guide* (ARM DUI 0203). This book provides tutorial information on writing code targeted at the ARM family of processors.
- *ARM Workbench IDE User Guide* (ARM DUI 0330). This book describes how to use the *Integrated Development Environment* (IDE) to configure and build projects for ARM targets.

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

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

- *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition* (ARM DDI 0406)
- *ARM7-M Architecture Reference Manual* (ARM DDI 0403)
- *ARM6-M Architecture Reference Manual* (ARM DDI 0419)
- ARM datasheet or technical reference manual for your hardware device.

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
- the version string of the tool, including the version number and date 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 did happen
- the commands you used, including any command-line options
- sample output illustrating the problem

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 numbers to which your comments apply
- a concise explanation of the problem.

General suggestions for additions and improvements are also welcome.

Chapter 1
Introduction

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

1.1 About RealView Compilation Tools

RVCT consists of a suite of tools, together with supporting documentation and examples. These tools enable you to write and build applications for the ARM family of processors.

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

1.1.1 Components of RVCT

This section provides an overview of the RVCT components.

Developer tools

The following developer tools are provided when you install RVCT:

armcc The ARM compiler. This compiles your C and C++ code.

armasm The ARM and Thumb® assembler. This assembles ARM and Thumb assembly language sources.

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.

Rogue Wave C++ library

The Rogue Wave library provides an implementation of the standard C++ library. For more information on the Rogue Wave library, see the HTML documentation on the CD ROM.

C++ libraries

The ARM C++ libraries provide:

- helper functions when compiling C++
- additional C++ functions not supported by the Rogue Wave library.

C libraries The ARM C libraries provide an implementation of the library features as defined in the C and C++ standards. See About the C and C++ libraries on page 2-2 in the *Libraries and Floating Point Support Guide* for more information.

C micro-libraries

The ARM C *micro-libraries* (Microlib) provide a highly optimized set of functions. These functions are for use with deeply embedded applications that have to fit into extremely small amounts of memory. See Chapter 3 The C Micro-library in the *Libraries and Floating Point Support Guide* for more information.

fromelf The ARM image conversion utility. This can also generate textual information about the input image, such as disassembly and its code and data size.

armar The ARM librarian. This enables sets of ELF format object files to be collected together and maintained in archives or libraries. You can pass such a library or archive to the linker in place of several ELF files. You may also use the archive for distribution to a third party for further application development.

Note

RealView Development Suite (RVDS) supports 64-bit Linux platforms but the RVCT tools have not been rebuilt in any special way to take advantage of this. All file I/O routines of RVCT use standard system calls with file sizes that fit in a 32-bit (**`signed`**) **`int`**. This means that the maximum image size is limited to 2GB, even when building on 64-bit platforms or machines with more than 2GB of memory fitted. If the size is exceeded, the linker reports an error message to indicate that there is not enough memory. This might cause confusion because sufficient memory is available but the application cannot access it.

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 providing it contains a symbol table member.

DWARF 3 DWARF 3 debug tables (DWARF Debugging Standard Version 3) are supported by all the tools in the RVDS.

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

ISO C The ARM compiler accepts the ISO C 1990 and 1999 source as input. See *Source language modes* in the *Compiler User Guide* for more information.

ISO C++ The ARM compiler accepts the ISO C++ 2003 source as input.

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

Note

The DWARF 2 and DWARF 3 standard is ambiguous in some areas such as debug frame data. This means that there is no guarantee that third-party debuggers can consume the DWARF produced by ARM code generation tools or that the RealView Debugger can consume the DWARF produced by third-party tools.

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 of these standards are open. Some are specific to the ARM architecture. They 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 as well as object libraries from different producers can work together.

ABI for the ARM Architecture (Base Standard) (BSABI) consists of a family of specifications including:

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

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

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.

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.

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

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.

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.

For more information on the base standard, software interfaces, and standards supported by ARM, see *install_directory*\Documentation_Specifications_4.0\PDF.

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.

Supporting software

To debug your programs under simulation, or on hardware based on an ARM core, use a suitable debugger, for example, the RealView Debugger. This is compatible with ELF, DWARF 2, and DWARF 3 as produced by GCC v3.4 or RVCT v2.2 and later.

To debug your programs under simulation, use the RealView ARMulator® ISS or the *Instruction Set System Model* (ISSM) supporting software. RealView Armulator ISS is an *Instruction Set Simulator* (ISS) supplied with RVDS. It communicates with the 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 the Cortex™ processors are available with this release. These models are accessible through the ISSM Debug Interface in RealView Debugger.

Code examples

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

1.2 Environment variables used by RVCT

Table 1-1 shows the environment variables used by RVCT.

Table 1-1 Environment variables used by RVCT

Environment variable	Setting
ARMROOT	Your installation directory root (*install_directory*). The default is C:\Program Files\ARM.
ARMLMD_LICENSE_FILE	The location of your ARM RealView license file. See the *FLEXnet for ARM Tools License Management Guide* for information on this environment variable.
RVCT40_ASMOPT	Additional ARM assembler options that are to be used outside of your regular makefile. For example: --licretry The options listed appear before any options specified for the armasm command in the makefile. Therefore, any options specified in the makefile might override the options listed in this environment variable.
RVCT40_CCOPT	Additional ARM compiler options that are to be used outside of your regular makefile. For example: --licretry The options listed appear before any options specified for the armcc command in the makefile. Therefore, any options specified in the makefile might override the options listed in this environment variable.
RVCT40_FROMELFOPT	Additional ARM fromelf options that are to be used outside of your regular makefile. For example: --licretry The options listed appear before any options specified for the fromelf command in the makefile. Therefore, any options specified in the makefile might override the options listed in this environment variable.
RVCT40_LINKOPT	Additional ARM linker options that are to be used outside of your regular makefile. For example: --licretry The options listed appear before any options specified for the armlink command in the makefile. Therefore, any options specified in the makefile might override the options listed in this environment variable.
RVCT40BIN	The RVCT program executables: *install_directory*\RVCT\Programs\...\win_32-pentium
RVCT40INC	The ARM compiler include files: *install_directory*\RVCT\Data\...\include\windows

Table 1-1 Environment variables used by RVCT (continued)

Environment variable	Setting
RVCT40LIB	The ARM compiler library files: *install_directory*\RVCT\Data\...\lib
RVDS_PROJECT	Identifies the project template directory.
RVDS_PROJECT_WORKDIR	Identifies the project working directory.

1.3 Getting more information

Depending on your installation, the full documentation suite is available in browser-based HTML format as well as the PDF format.

Note

A glossary of ARM terms used in the RVDS documentation is provided in the *RealView Development Suite Getting Started Guide*.

If you install the documentation suite, you can access the documentation using one of the following methods:

- Depending on your platform, to view the documentation suite:
  - on Windows, select:
    start → All Programs → ARM → Help viewer v1.0
  - on Red Hat Linux, select:
    Start Menu → Programs → ARM → Help viewer v1.0.

  This displays a standalone viewer where you can:
  - view the RVDS documentation in HTML format
  - perform text searches on all documents or a subset of documents
  - access the corresponding PDF file for each document.

  Note

  You cannot search all PDF documentation when viewing PDF documents from the standalone viewer.

- Depending on your platform, to view the PDF documentation:
  - on Windows, select:
    start → All Programs → ARM → RealView Development Suite v4.0 → RVDS v4.0 Documentation Suite
  - on Red Hat Linux, select:
    Start Menu → Programs → ARM → RealView Development Suite v4.0 → RVDS v4.0 Documentation Suite.

  This displays a PDF document containing links to the RVDS documentation in PDF format. You can also perform text searches on all the PDF documentation.

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 standard installation. See *Rogue Wave documentation* on page 1-10 for more information.

1.3.1 Rogue Wave documentation

The manuals for the Rogue Wave Standard C++ Library for RVCT are provided on the product CD ROM as HTML files. A standard web browser may be used 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.

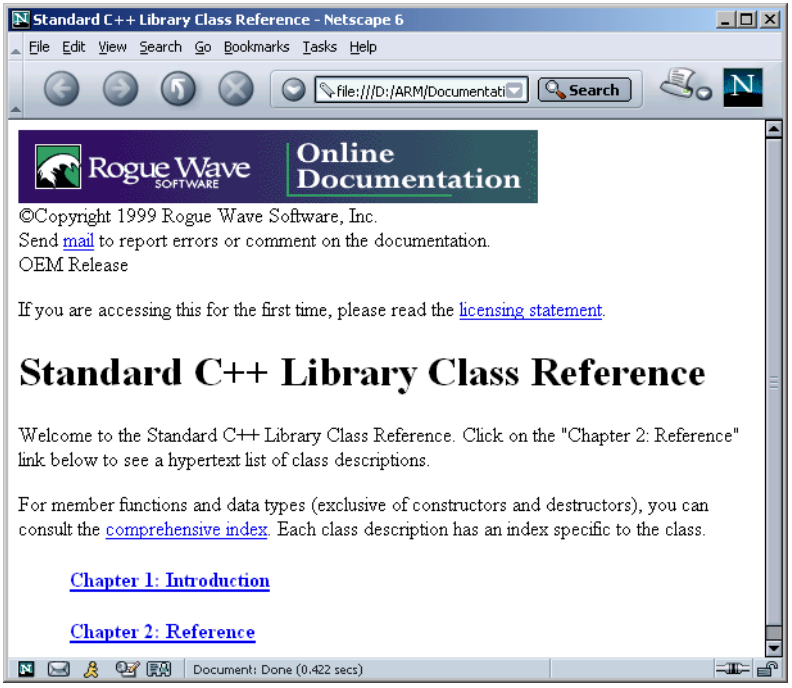

Figure 1-1 Rogue Wave HTML documentation

Chapter 2
Creating an Application

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

2.1 Using the ARM compilation tools

A typical application development might involve the following:

- C/C++ source code for main application (`armcc`)
- assembly source code for near-hardware components (`armasm`), such as interrupt service routines
- linking all objects together to generate an image (`armlink`)
- converting an image to flash format in plain binary, Intel Hex, and Motorola-S formats (`fromelf`)

Figure 2-1 shows how the ARM compilation tools are linked for the development of a typical application.

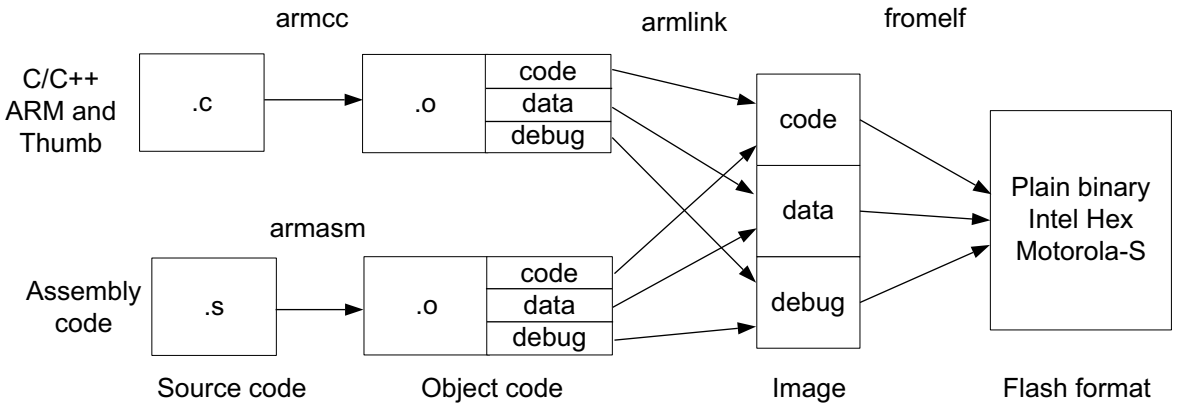

Figure 2-1 A typical tool usage flow diagram

2.2 Using the ARM compiler

The ARM compiler, armcc, can compile C and C++ source code into ARM and Thumb code.

Typically, you invoke the ARM compiler as follows:

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

You can specify one or more input files.

For full details on compiling for ARM and Thumb, and how the compiler startup configuration is adjusted by the filename extension you specify, see Chapter 2 *Getting started with the ARM Compiler* in the *Compiler User Guide*.

2.2.1 Building the Dhrystone example

Sample C source code for a range of applications is installed in the 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 the Dhrystone benchmark program is installed in the 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 -O3 -Otime --no_inline --no_multifile -DMSC_CLOCK \
       dhry_1.c dhry_2.c
   ```

   The following options are commonly used:

-c	Tells the compiler to compile only, and not link.
--debug	Tells the compiler to add debug tables for source-level debugging.
-O3	Tells the compiler to generate code with highest optimizations applied.
-Otime	Tells the compiler to optimize the code for speed, not space.

   The options --no_inline and --no_multifile are required to retain the spirit of the Dhrystone benchmark:

   - --no_inline is required to disable function inlining, because Dhrystone demands that no procedures are merged
   - --no_multifile is required to disable multifile compilation, because Dhrystone demands that the two source files are compiled independently.

The following options are also used in the build Dhrystone build files:

`-W` Tells the compiler to disable all warnings.

`-DMSC_CLOCK` Tells the compiler to use the C library function `clock()` for timing measurements.

For more information on the compiler options, see Chapter 2 *Compiler Command-line Options* in the *Compiler Reference Guide*.

Note

Be aware that `--arm` is the default compiler option. See *Compiling for ARM code* for more information.

2. Link the files together, see *Using the ARM linker* on page 2-6
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.

2.2.2 Compiling for ARM code

The following compiler options generate ARM code:

`--arm` Tells the compiler to generate ARM code in preference to Thumb code. However, `#pragma thumb` overrides this option. This is the default compiler option.

`--arm_only` Forces the compiler to generate only ARM code. The compiler behaves as if Thumb is absent from the target architecture. Any `#pragma thumb` declarations are ignored.

See also

- *#pragma thumb* on page 4-73 in the *Compiler User Guide*
- *--arm* on page 2-8 in the *Compiler Reference Guide*
- *--arm_only* on page 2-15 in the *Compiler Reference Guide*.

2.2.3 Compiling for Thumb code

To build a Thumb version use:

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

where:

`--thumb` Tells the compiler to generate Thumb code in preference to ARM code. However, `#pragma arm` overrides this option.

See also

- *#pragma arm* on page 4-59 in the *Compiler User Guide*
- *--thumb* on page 2-122 in the *Compiler Reference Guide*.

2.3 Using the ARM linker

The linker combines the contents of one or more object files with selected parts of one or more object libraries to produce an image or object file.

Typically, you invoke the ARM linker as follows:

`armlink [options] file_1 ... file_n`

For more information, see Chapter 2 *Getting Started with the ARM Linker* in the *Linker User Guide*.

2.3.1 Linking the Dhrystone example

For the Dhrystone example program, link the object files 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.

2.4 Using the ARM assembler

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

```
armasm [options] inputfile
```

For example, to assemble the code in a file called `myfile.s`, and to include debugging information in the resulting object file, type:

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

This generates an object file called `myfile.o`.

For full details on the options and syntax, see *Command syntax* on page 3-2 in the *Assembler Guide*.

2.4.1 Building an example from assembler source

Sample assembly language code is installed in the 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 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.

2.5 Using fromelf

The features of the ARM `fromelf` utility include:

- converting an executable image in ELF executable format to other file formats
- controlling debug information in output files
- disassembling either an ELF image or an ELF object file
- protecting IP in images and objects that are delivered to third parties
- printing information about an ELF image or an ELF object file.

For more information, see Chapter 2 *Using fromelf* in the *Utilities Guide*.

2.5.1 Using fromelf examples

The following examples show how to use `fromelf`:

```
fromelf --text -c -s --output=outfile.lst infile.axf
```

Creates a plain text output file that contains the disassembled code and the symbol table of an ELF image.

```
fromelf --bin --16x2 --output=outfile.bin infile.axf
```

Creates two files in binary format (`outfile0.bin` and `outfile1.bin`) for a target system with a memory configuration of a 16-bit memory width in two banks.

The output files in the last example are suitable for writing directly to a Flash device.

2.6 Using the ARM Workbench IDE

The ARM Workbench IDE enables you to use a graphical user interface to manage your software development projects. The ARM Workbench provides a fully integrated IDE that combines software development with the compilation and debug technology of all the RealView tools. The examples provided with the workbench contain some fully working projects for you to try. To use them, you must launch the workbench and import the examples into your workspace. See the *ARM Workbench IDE User Guide* for more information.

Chapter 3
Differences between RVCT v4.0 and RVCT v3.1

This chapter describes the major differences between ARM® RealView® Compilation Tools (RVCT) v4.0 and RVCT v3.1. It contains the following sections:

For differences between previous releases of RVCT, see Appendix A *About Previous Releases*.

3.1 RealView Compilation Tools v4.0 overview

The most important differences between RVCT v4.0 and RVCT v3.1 are:

- Support for Cortex™-A9 (RVDS Professional edition only) and Cortex-R4F processors. To see a full list of supported architectures and processors use the `--cpu=list` command-line option.
- Enhanced support for building Linux applications with RVCT and CodeSourcery tools.
- Enhanced symbol visibility.
- Enhanced `fromelf` functionality.
- Enhanced callgraph functionality in the linker.
- Generation of images suitable for prelinking.
- Architecture-aware disassembly.
- Initial bit-banding support for the Cortex™-M3 processor.
- Enhanced floating point performance for the Cortex™-M1 processor.
- Enhanced code optimization.
- Profiler-guided optimization.
- Link-time code generation.

3.2 Changes to the documentation in RVCT v4.0

The following changes have been made to the documentation in RVCT v4.0:

- The *Linker and Utilities Guide* is split into the following books:
  - *Linker User Guide*
  - *Linker Reference Guide*
  - *Utilities Guide*.
- The *NEON™ Vectorizing Compiler Guide* is incorporated into the *Compiler User Guide* and the *Compiler Reference Guide*.
- The *Developer Guide* has been restructured and updated to reflect the latest ARM architectures and processors.
- The `--licretry` option is documented. The option is supported by the ARM assembler, ARM compiler, ARM linker, and the `fromelf` utility.
- All documents have been updated to reflect the features in the related tools.

See the remaining sections in this chapter for details of the changes to the tools in this release.

3.3 Changes to the ARM compiler in RVCT v4.0

The following changes have been made to the compiler in RVCT v4.0:

- Support for Cortex-M0, Coretx-M4, Cortex-A9, and Cortex-R4F processors, using the `--cpu=Cortex-M0`, `--cpu=Cortex-M4`, `--cpu=Cortex-A9`, and `--cpu=Cortex-R4F` switches. See *Command-line options* on page 2-2 in the *Compiler Reference Guide*.
- Many new intrinsics have been added. See *Intrinsics* on page 4-2 in the *Compiler User Guide*.
- The `--arm_only` command-line option has been added to force the compiler to output only ARM code, and ignore any `#pragma thumb` declarations.

  See *Command-line options* on page 2-2 in the *Compiler Reference Guide*.
- The following command-line options have been added to support the building of Linux applications:
  - `--arm_linux`
  - `--arm_linux_config_file`
  - `--arm_linux_configure`
  - `--arm_linux_paths`
  - `--configure_cpp_headers`
  - `--configure_extra_includes`
  - `--configure_extra_libraries`
  - `--configure_gcc`
  - `--configure_gld`
  - `--configure_sysroot`
  - `--shared`
  - `--translate_g++`
  - `--translate_gcc`
  - `--translate_gld`.
- These options have been added to the `--fpu` command-line option:
  - `vfpv3_fp16`
  - `vfpv3_d16`
  - `vfpv3_d16_fp16`
  - `softvfp+vfpv3_fp16`
  - `softvfp+vfpv3_d16`
  - `softvfp+vfpv3_d16_fp16`.

The corresponding predefined macros have also been added. See *Using floating-point arithmetic* on page 5-31 in the *Compiler User Guide*.

- The `__attribute__((bitband))` type attribute has been added to support bit-banding on the Cortex-M3 processor. See *Type attributes* on page 4-43 in the *Compiler Reference Guide*.
- Enhanced floating point performance for the Cortex-M1 processor. Single-precisions arithmetic is at least six times faster, and double-precision arithmetic is at least 3.5 times faster.
- The ELF symbol visibility used for objects and functions that are explicitly or implicitly marked `dllexport` has changed. Symbols defined in `STV_DEFAULT` (preemptible) sections in RVCT v3.1 are defined in `STV_PROTECTED` (exported non-preemptible) sections in RVCT v4.0. However, symbols defined in `COMDAT` sections have not moved, and are still defined in `STV_DEFAULT` sections.

  The `--no_hide_all` compiler option continues to make the visibility of symbols `STV_DEFAULT` instead of `STV_HIDDEN`, so this is no longer the same as exporting them.
- The new `--retain` compiler option enables you to restrict the code transformations performed by the compiler. For example, you can prevent inline functions being removed if they are unused. The following attributes are also provided:
  - `__attribute__((notailcall))`
  - `__attribute__((nomerge))`.

  See *--retain=option* on page 2-113 in the *Compiler Reference Guide*.

  See *Function attributes* on page 4-31 in the *Compiler Reference Guide*.
- The following functions attributes are also provided:
  - `__attribute__((weakref("target")))`.
- The following variable attributes are also provided:
  - `__attribute__((noinline))`
  - `__attribute__((weakref("target")))`.
- The following pragma has been added:
  - `#pragma import(__use_smaller_memcpy)`
- Profiler-guided optimization exploits application profiles generated by the ARM Profiler. Profiles can be input to the compiler and linker using the new `--profile` option, to generate code that is smaller in size and faster in terms of performance.

- Link-time code generation is implemented using the new `--ltcg` compiler and linker option, and provides optimizations that include:
  - cross-module inlining to improve performance
  - sharing of base addresses to reduce code size.

  See *Function inlining* on page 5-18 in the *Compiler User Guide* for more details.
- Other command-line options that have been added are:
  - `--compatible`
  - `--default_definition_visibility=`*visibility*
  - `--device`
  - `--fp16_format`
  - `--gnu_defaults`
  - `--gnu_instrument`, `--no_gnu_instrument`
  - `--library_interface=rvct_c90`
  - `--licretry`
  - `--narrow_volatile_bitfields`
  - `--remove_unneeded_entities`, `--no_remove_unneeded_entities`.
- Functions to provide compatibility with GNU library header files have been added. See *GNU builtin functions* on page 4-195 in the *Compiler Reference Guide*.

3.4 Changes to library support in RVCT v4.0

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.

The functionality difference between the micro and standard libraries has been reduced. Also wide character IO is supported. See the *Libraries and Floating Point Support Guide* for more details.

3.5 Changes to the ARM linker in RVCT v4.0

The following changes to `armlink` have been made to the linker in RVCT v4.0 build 821:

Base Platform linking model that supports scatter-loading

The Base Platform linking model is a superset of the *Base Platform Application Binary Interface* (BPABI) model. You specify the Base Platform linking model with the `--base_platform` command-line option. You can also use scatter-loading with the Base Platform linking model.

String merging enhancements

The linker can merge strings where the address is generated by `MOVT` and `MOVW` instructions.

Placing and sorting input sections

Additional command-line options are available to control the placement and the sort order of input sections when using the `.ANY` module selector:

- `--any_contingency`
- `--any_placement=`*`algorithm`*
- `--any_sort_order=`*`order`*.

Also, the `ANY_SIZE` keyword is available for an execution region definition in a scatter-loading file.

The `--info=any` option is provided to show details of the sorting and placement of `.ANY` sections.

Using a tiebreaker when sorting input sections with equal properties

In RVCT 4.0 builds earlier than 821, the sorting algorithm used by the linker is based on the order that the linker creates sections in its section data structure.

In RVCT 4.0 build 821 and later, an additional sorting algorithm is available based on the order that sections appear on the linker command-line.

You select the algorithm using the `--tiebreaker` command-line option. For backwards compatibility, the section create order is the default.

The `--section_index_display=cmdline` option is provided to show the sections in the memory map output in the same order they appear on the command-line.

Controlling veneer sharing across execution regions

The `--[no_]crosser_veneershare` command-line option is available to enable and disable veneer sharing across execution regions.

Linker-defined symbols

The linker now supports the following linker-defined symbols:

`Load$$`*region_name*`$$ZI$$Base`
`Load$$`*region_name*`$$ZI$$Limit`
`Load$$`*region_name*`$$ZI$$Length`

Additional command-line options supported

The following command-line options have been added:

- `--[no_]as_needed`
- `--[no_]comment_section`
- `--emit_debug_overlay_relocs`
- `--emit_debug_overlay_section`
- `--emit_non_debug_relocs`
- `--emit_relocs`
- `--[no_]execstack`
- `--[no_]export_dynamic`
- `--force_explicit_attr`
- `--gnu_linker_defined_syms`
- `--[no_]import_unresolved`
- `--info=any`, `--info=compression`, and `--info=veneerpool`
- `--[no_]load_addr_map_info`
- `--pltgot_opts=crosslr`
- `--remarks`
- `--[no_]strict_enum_size`
- `--[no_]strict_flags`
- `--[no_]strict_ph`
- `--[no_]strict_symbols`
- `--[no_]strict_visibility`
- `--[no_]strict_wchar_size`
- `--symbolic`
- `--[no_]thumb2_library`
- `--undefined=`*symbol*
- `--undefined_and_export=`*symbol*
- `--use_definition_visibility`
- `--veneer_inject_type`
- `--veneer_pool_size`
- `--version_number`.

Changes to command-line options

The following changes have been made to existing command-line options:

- the `any` and `veneerpools` topics have been added to `--info`
- the `cmdline` type has been added to `--section_index_display`
- the `Alignment`, `BreadthFirstCallTree` and `LexicalState` algorithms have been added to `--sort`.

The following command-line options were added to the linker in the original RVCT v4.0 release:

- `--[no_]add_needed`
- `--arm_only`
- `--arm_linux`
- `--[no_]combreloc`
- `--device`
- `--filtercomment`
- `--info=visibility`
- `--no_largeregions`
- `--licretry`
- `--ltcg`
- `--[no_]muldefweak`
- `--[no_]prelink_support`
- `--privacy`
- `--profile`
- `--section_index_display=`*type*.

See also:

- *Using command-line options* on page 2-10 in the *Linker User Guide*
- *Command-line options listed alphabetically* on page 2-2 in the *Linker Reference Guide*.

3.6 Changes to the ARM assembler in RVCT v4.0

The following changes have been made to the assembler in RVCT v4.0:

- The `ALIAS` directive has been added.
- Support for GNU_STACK/.note.GNU-stack/PT_GNU_STACK has been added.
- These NEON™ and VFP instructions have been added:
  - `VCVT`, with half-precision extension
  - `VCVTB` and `VCVTT`, with half-precision extension.

  See *NEON general data processing instructions* on page 5-36 and *VFP instructions* on page 5-94 in the *Assembler Guide*.
- The `PLDW` instruction has been added for support of the Cortex-A9. See *PLD, PLDW, and PLI* on page 4-25 in the *Assembler Guide* for more details.
- These directives now support a symbol type:
  - `EXPORT`
  - `EXTERN`
  - `GLOBAL`
  - `IMPORT`.
- The following command-line switches have been added:
  - `--arm_only`
  - `--cpreproc`
  - `--device`
  - `--execstack, --no_exestack`
  - `--licretry`
  - `--report-if-not-wysiwyg`.

  See Chapter 3 *Assembler Reference* in the *Assembler Guide* for more details.

See also *General information* on page 5-14 in the *Assembler Guide*.

3.7 Changes to the fromelf utility in RVCT v4.0

The following options have been added to the `fromelf` utility in RVCT v4.0:

- `--base`
- `--bincombined`
- `--bincombined_base`
- `--bincombined_padding`
- `--cad`
- `--cadcombined`
- `--compare`
- `--cpu`
- `--datasymbols`
- `--diag_error`
- `--diag_remark`
- `--diag_warning`
- `--device`
- `--disassemble`
- `--emit`
- `--fpu`
- `--globalize`
- `--hide`
- `--hide_and_localize`
- `--ignore_section`
- `--ignore_symbol`
- `--info=instruction_usage`
- `--in_place`
- `--interleave`
- `--licretry`
- `--localize`
- `--only`
- `--qualify`
- `--reinitialize_workdir`
- `--relax_section`
- `--relax_symbol`
- `--rename`
- `--show`
- `--show_and_globalize`
- `--source_directory`

- `--version_number`
- `-w`
- `--workdir.`

A previously undocumented option, `--text -w`, is documented.

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

Architecture-aware disassembly is supported.

See Chapter 2 *Using fromelf* in the *Utilities Guide* for more details.

3.8 Deprecated features in RVCT v4.0

The following features are deprecated in RVCT v4.0:

- ARM/Thumb Synonym definitions.
- The `--memaccess` compiler option is deprecated.
- The following mathlib functions are deprecated:
  - `gamma()`
  - `gamma_r()`
  - `lgamma_r()`
  - `scalb()`
  - `significand()`.
- The following Bessel functions are deprecated:
  - `j0()`
  - `j1()`
  - `jn()`
  - `y0()`
  - `y1()`
  - `yn()`.
- The ARM assembler option `-0` (in uppercase, not lowercase).

3.9 Obsolete features in RVCT v4.0

The following features are obsolete in RVCT v4.0:

- These tools are obsolete:
  - armcpp
  - tcc
  - tcpp.

  Any references to armcpp, tcc, or tcpp in makefiles must be changed to armcc --cpp, armcc --thumb, or armcc --thumb --cpp respectively.
- Tailoring locale and CTYPE using C macros is obsolete.
- The ARM assembler option -D (the synonym for --depend) is obsolete.

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

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

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

Appendix A
About Previous Releases

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

It contains the following sections:

A.1 Differences between RVCT v3.1 and RVCT v3.0

This section describes the differences between RVCT v3.1 and RVCT v3.0.

A.1.1 RVCT v3.1 Overview

- RVCT v3.1 now supports the following new `--cpu` options:
  - ARMv7 is not a recognised ARM architecture. `--cpu=7` denotes the features that are common to all of the ARMv7-A, ARMv7-R, and ARMv7-M architectures. By definition, any given feature used with `--cpu=7` exists on all of the ARMv7-A, ARMv7-R, and ARMv7-M architectures.
  - Cortex™ processors:

    ```
    --cpu=Cortex-R4
    --cpu=Cortex-M1
    --Cortex-A8NoNEON.
    ```

  - Marvell Feroceon processors `--cpu=88FR`*xxx*.

  To see a full list of supported architectures and processors use `--cpu=list`. See *--cpu=list* on page 2-30 and *--cpu=name* on page 2-30 in the *Compiler Reference Guide* for more information.
- RVCT v3.1 removes the option `--apcs=/adsabi`. Compilation of ADS-compatible objects, and linking of legacy ADS objects and libraries is no longer possible.
- RVCT v3.1 includes full C99 language support, except for complex numbers and wide I/O. See *New features of C99* on page 5-45 in the *Compiler User Guide* for more information.
- RVCT v3.1 provides full support for NEON™ Technology. See Chapter 3 *Using the NEON Vectorizing Compiler* and *NEON Intrinsics* on page 4-9 in the *Compiler User Guide* for more information.
- RVCT v3.1 provides the C micro-library (microlib) as an alternative to the standard C library for use with deeply embedded applications that have to fit into extremely small amounts of memory. Microlib is not fully compliant with ISO C standards. See Chapter 3 *The C Micro-library* in the *Libraries and Floating Point Support Guide* for more information.
- RVCT v3.1 provides assembler macros to tailor locale and CTYPE functions. See *Tailoring locale and CTYPE using assembler macros* on page 2-42 in the *Libraries and Floating Point Support Guide* for more information.
- RVCT v3.1 provides specific compiler intrinsics for easy access to low-level features of ARM architecture-based processors in C and C++. See *Instruction intrinsics* on page 4-3 in the *Compiler User Guide* for more information.

- RVCT v3.1 contains support for the ETSI intrinsics. See *ETSI basic operations* on page 4-6 in the *Compiler User Guide* for more information.
- RVCT v3.1 contains support for the C55x intrinsics. See *TI C55x intrinsics* on page 4-8 in the *Compiler User Guide* for more information.
- RVCT v3.1 provides enhanced support for callgraphs, scatter-loading files and section placements to execution regions. See *--callgraph, --no_callgraph* on page 2-18 in the *Linker Reference Guide* for more information.
- RVCT v3.1 provides enhanced protection for IP in deliverable images and objects. See *Changes to the fromelf utility in RVCT v3.1* on page A-9 for more information.
- RVCT v3.1 provides the option to control the elimination of redundant path name information in file paths. This option can be used with the ARM compiler, linker and assembler. See *--reduce_paths, --no_reduce_paths* on page 2-70 in the *Linker Reference Guide* for more information.
- RVDS v3.1 now includes the Eclipse IDE as the main project management facility. The Eclipse Plug-ins for RVDS provides comprehensive configuration panels to setup all options for RVCT. See the *RealView Development Suite Eclipse Plugins User Guide* for more information.
- RVDS v3.1 has changed the `--verbose` command-line option to print diagnostic output to `stdout` instead of `stderr`. If you want to redirect this information to a file you must use `--list` instead of `--errors`.

A.1.2 Obsolete features in RVCT v3.1

Be aware of the following differences in RVCT v3.1:

- Solaris platforms are not supported in RVCT v3.1.
- Red Hat Linux Enterprise v3 32-bit host platform is no longer supported in RVCT v3.1.
- The compiler option `--apcs=/adsabi` is not supported in RVCT v3.1.
- All RVCT v3.0 deprecated features are now obsolete in RVCT v3.1. See *Deprecated features in RVCT v3.0* on page A-13 for more information.

A.1.3 Deprecated features in RVCT v3.1

Be aware of the following differences in RVCT v3.1:

- The following tool names are deprecated:
  - armcpp
  - tcc
  - tcpp.
- Instruction set switching using file extension, such as .ac and .tc, is deprecated
- The inline assembler is deprecated when compiling for ARMv7-A, ARMv7-R, and ARMv7-M architectures, and higher. The compiler generates a warning for ARMv7-A and ARMv7-R. When generating Thumb code, the compiler generates an error for ARMv6-M and ARMv7-M.
- The following options are deprecated.
  - --split_ldm
  - --memaccess.
- Tailoring locale and CTYPE using C macros is deprecated.
- The fromelf option --no_comment_section is deprecated, and is no longer documented. Use the --strip=comment option instead.
- The fromelf options --debug and --no_debug are deprecated, and are no longer documented. Use --emit=debug or --strip=debug,symbols instead.
- The following assembler features are deprecated:
  - The use of VFP vector mode is deprecated in new code, and vector notation is not supported in Unified Assembler Language. To use vector notation, you must use the old VFP mnemonics.
  - Use of CPSR to access user mode accessible bits is deprecated for later CPUs. Use APSR instead.
  - Use of PC-relative addressing in STC and STC2 for ARMv6T2 and above.
  - --checkreglist, use --diag_warning 1206 instead.
  - Use of 16-bit LDM and STM with the base register in the register list, and writeback specified.

A.1.4 Changes to the ARM compiler in RVCT v3.1

Be aware of the following differences in the compiler in RVCT v3.1:

- `--cpu` command-line option supports new architectures and processors. See *RVCT v3.1 Overview* on page A-2 for more information.
- `--vectorize` command-line option to generate NEON™ vector instructions directly from C or C++ code. See *--vectorize, --no_vectorize* on page 2-131 in the *Compiler Reference Guide* for more information.

  Note

  To use this option you must have a NEON vectorizing compiler license. This license is provided with RVDS Professional edition.

- `--c99` command-line option provides full C99 language support except for complex numbers and wide I/O.
- `--library_type=`*`lib`* command-line option to select the relevant runtime library. See *--library_type=lib* on page 2-81 in the *Compiler Reference Guide* for more information.
- `--bss_threshold=`*`num`* command-line option to control the placement of small global ZI data items.
- `--diag_suppress=optimizations` and `--diag_warning=optimizations` command-line options to control diagnostic messages for optimization.
- `--[no_]vla` command-line option to control support for variable length arrays.
- `--wchar16` and `--wchar32` command-line options to change the type of `wchar_t`.
- `--apcs=/adsabi` is not supported in RVCT v3.1.
- `--[no_]reduce_paths` option to control the elimination of redundant path name information in file paths.
- Provides NEON™ intrinsics to generate SIMD instructions from C and C++.
- Supports the emulation of selected Texas Instruments C55x intrinsics.
- Supports ETSI basic operations for implementations of speech codecs.
- `__attribute__((section("`*`name`*`")))` to encode placement addresses for use with the `--autoat` command-line option.
- Provides specific intrinsics for easy access to low-level features of ARM architecture-based processors for C and C++.

A.1.5 Changes to library support in RVCT v3.1

Be aware of the following differences in library support in RVCT v3.1:

- RVCT v3.1 has a new library naming convention.
- RVCT v3.1 includes library support for use with deeply embedded applications that have to fit into extremely small amounts of memory. Microlib introduces a new set of functions that are not fully standards-compliant but are highly optimized for minimum code size.
- RVCT v3.1 provides assembler macros to tailor locale and CTYPE functions.

See also:

- the *Libraries and Floating Point Support Guide*

A.1.6 Changes to the ARM linker in RVCT v3.1

Be aware of the following differences in the linker in RVCT v3.1:

- `--info summarysizes` attribute summarizes the image code and data sizes.
- `--cpu=`*`name`* command-line option specifies an upper limit for the selected ARM processor or architecture.
- `--fpu=`*`name`* command-line option specifies an upper limit for the selected FPU architecture.
- `--bpabi` command-line option generates BPABI executables.
- `--dll` command-line option generates BPABI DLLs.
- `--predefine=`"*`string`*" command-line option enables commands to be passed to the pre-processor specified on the first line of the scatter file.
- `--[no_]reduce_paths` option controls the elimination of redundant path name information in file paths.
- `--autoat` command-line option controls the automatic placement of `__at` sections to execution regions.
- `--verbose` command-line option prints diagnostic output to `stout`. If you want to redirect this information to a file you must use `--list`. See *--verbose* on page 2-98 in the Linker Reference Guide for more information.

- Variables can be assigned to `__at` sections by either explicitly naming the section using the `__attribute__((section("name")))` or by using the `__attribute__((at("name")))` which sets up the name of the section for you. See *Examples of specifying region and section addresses* on page 5-10 in the *Linker User Guide* for more information.
- `ALIGN alignment` attribute for boundary alignment in scatter-loading files.
- Priority ordering using `.ANYnum` for use when assigning input sections to execution regions.
- Attributes for execution region content:
  - `ZEROPAD` attribute to initialize sections in an ELF file
  - `PADVALUE` attribute to set values for padding bytes
  - `FILL` attribute to create a linker generated region containing a value.
- `--pltgot=type`, `--pltgot_opts=mode` and `--info pltgot` command-line options to generate tables corresponding to the different addressing modes of the BPABI
- `--library_type=lib` command-line option to select the relevant runtime library. See *--library_type=lib* on page 2-81 in the *Compiler Reference Guide* for more information.
- callgraph options:
  - `--callgraph_file=filename` defines the name of the output file
  - `--callgraph_output=fmt` defines the type of output file
  - `--cgfile=opt` controls the content of the callgraph
  - `--cgsymbol=type` controls the content of the callgraph
  - `--cgundefined=type` controls the content of the callgraph.
- The linker interprets relative based load regions differently in RVCT v3.1. It no longer adjusts the base address of a load region to account for the *Zero Initialized* (ZI) data in a previous load region.

  Example A-1 shows a scatter file with overlapping data.

Example A-1 Scatter file with a relative base address for LR2

```
LR1 0x8000
{
    er_progbits +0
    {
        *(+RO,+RW) ; Takes space in the Load Region
    }
    er_zi +0
```

```
    {
        *(+ZI) ; Takes no space in the Load Region
    }
}
LR2 +0 ; Load Region follows immediately from LR1
{
    er_moreprogbits +0
    {
        file1.o(+RO) ; Takes space in the Load Region
    }
}
```

RVCT v3.1 produces an error message if overlapping is detected. To correct this, an expression is required in the scatter file to calculate the base address of `LR2`. Example A-2 shows the corrected scatter file.

Example A-2 Scatter file with a calculated base address for LR2

```
LR1 0x8000
{
    er_progbits +0
    {
        *(+RO,+RW) ; Takes space in the Load Region
    }
    er_zi +0
    {
        *(+ZI) ; Takes no space in the Load Region
    }
}
LR2 ImageLimit(er_zi) ; Set the address of LR2 to limit of er_zi
{
    er_moreprogbits +0
    {
        file1.o(+RO) ; Takes space in the Load Region
    }
}
```

A.1.7 Changes to the ARM assembler in RVCT v3.1

Be aware of the following differences in the assembler in RVCT v3.1:

- `--cpu` command-line option supports new architectures and processors. See *RVCT v3.1 Overview* on page A-2 for more information.
- `--library_type=`*`lib`* command-line option to select the relevant runtime library.

- `--[no_]reduce_paths` option to control the elimination of redundant path name information in file paths.
- Use APSR for user mode bits of the CPSR.
- Use of VFP vector mode is deprecated in new code, and vector notation is not supported in Unified Assembler Language.
- Supports Intel Wireless MMX2™ instructions.
- `CODEALIGN` attribute can be set on the `AREA` directive.
- `--depend_format=`*`string`* changes the format of output dependency file.
- `--no_code_gen` option is for use with `--depend` and generates no object file.
- `UND` is a new directive that encodes an architecturally undefined instruction.

See the *Assembler Guide* for more information.

A.1.8 Changes to the fromelf utility in RVCT v3.1

Be aware of the following differences in RVCT v3.1:

- `--privacy` command-line option that changes section names
- `--strip` command-line option that protects IP in deliverable images and objects.

See also:

- the *Utilities Guide*

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

This section describes the differences between RVCT v3.0 and RVCTv2.2.

A.2.1 General changes in RVCT v3.0

The following changes were made in RVCT v3.0:

- 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 micro-controller 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 processors 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 *Single Instruction, Multiple Data* (SIMD) 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.

- 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 debug tables (DWARF Debugging Standard Version 3), 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 at run-time 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
  - RVCT30_ASMOPT.

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

Obsolete features in RVCT v3.0

The following changes were made 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 A Using Old Command-line Options in the *Compiler User 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 A Using Old Command-line Options in the *Compiler User Guide* for more information.
- 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.

Deprecated features in RVCT v3.0

The following changes were made 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 in RVCT 3.0 and is removed in RVCT 3.1.
- The following assembler options are deprecated and are removed in RVCT 3.1:
  - `--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.

A.2.2 Changes to the ARM compiler in RVCT v3.0

The following changes were made 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.
- 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 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()` gives improved code size (over `__user_initial_stackheap()`) because there is no requirement for a temporary stack.
- 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.

A.2.3 Changes to library support in RVCT v3.0

The following changes were made 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 that contain `ARM_LIB_STACK`, `ARM_LIB_HEAP` and/or `ARM_LIB_STACKHEAP` directives.
- 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.

A.2.4 Changes to the ARM linker in RVCT v3.0

The following changes were made 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.

- RVCTv3.0 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.

A.2.5 Changes to the ARM assembler in RVCT v3.0

The following changes were made 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`).
- 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 Intel 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.
- 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`.

A.2.6 Changes to the fromelf utility in RVCT v3.0

The following changes were made 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.

A.3 Differences between RVCT v2.2 SP1 and RVCT v2.2

This section describes the differences between RVCT v2.2 delivered with RealView Developer Suite 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 *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 = 0x8000000 };              /* is treated as unsigned int */
  ```

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

  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
...
```

A.4 Differences between RVCT v2.2 and RVCT v2.1

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

A.4.1 General changes in RVCT v2.2

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.

  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 in RVCT v2.2* on page A-28 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 in RVCT v2.2* on page A-26 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.
- Options that were deprecated in RVCT v2.1 are now obsolete in v2.2 (see *Obsolete features in RVCT v2.2* on page A-24).
- Some options that were supported in RVCT v2.1 are now deprecated in v2.2 (see *Deprecated features in RVCT v2.2* on page A-25).

Obsolete features in RVCT v2.2

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 A Using Old Command-line Options in the *Compiler User 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
  ```
- 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 in RVCT v2.2

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.

  The disassembler supports only the first form.

A.4.2 Changes to the ARM compiler in RVCT v2.2

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 */
    /* 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 supports 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()`.
- 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.4.3 Changes to library support in RVCT v2.2

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`.

  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 lockon the supplied mutex.

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

  _mutex_release()

  This causes the calling thread to release the suppliedmutex.

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

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

- 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.4.4 Changes to the ARM linker in RVCT v2.2

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.

- 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 the *Linker and Utilities Guide*.

A.4.5 Changes to the ARM assembler in RVCT v2.2

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`

A.4.6 Changes to the fromelf utility in RVCT v2.2

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.

A.5 Differences between RVCT v2.1 and RVCT v2.0

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

A.5.1 General changes in RVCT v2.1

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.
- The compiler supports the new `--min_array_alignment` option.
- 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`).

  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.5.2 Changes to the ARM compiler in RVCT v2.1

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 `--min_array_alignment` option to enable you to specify the minimum alignment of arrays.

- New `__breakpoint()` intrinsic.

- Noreturn functions.

- --old_cfe option is now obsolete.

A.5.3 Changes to library support in RVCT v2.1

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.5.4 Changes to the ARM linker in RVCT v2.1

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`*

A.5.5 Changes to the ARM assembler in RVCT v2.1

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.5.6 Changes to the fromelf utility in RVCT v2.1

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.

A.6 Differences between RVCT v2.0 and RVCT v1.2

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

A.6.1 General changes in RVCT v2.0

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.6.2 Changes to the ARM compiler in RVCT v2.0

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.
- 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.6.3 Changes to the ARM linker in RVCT v2.0

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.

A.6.4 Changes to the ARM assembler in RVCT v2.0

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}.