RealView® Developer Suite

Version 2.0

Getting Started Guide

ARM®

ARM DUI 0250A

RealView Developer Suite
Getting Started Guide

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Change
September 2003	A	RVDS Release v2.0

Proprietary Notice

Confidentiality Status

This document is Open Access. This document has no restriction on distribution.

Product Status

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

Web Address

`http://www.arm.com`

Contents
RealView Developer Suite Getting Started Guide

Preface

This preface introduces the *RealView® Developer Suite v2.0 Getting Started Guide*, that shows you how to start using RealView Developer Suite to manage software projects and to debug your application programs. It contains the following sections:

- *About this book* on page vi
- *Feedback* on page ix.

About this book

RealView Developer Suite provides tools for building, debugging, and managing software development projects targeting ARM processors. This book contains:

- an introduction to the software components that make up RealView Developer Suite
- a summary of the main features of the RealView Developer Suite components
- a summary of the differences between RealView Developer Suite v2.0 and ADS v1.2
- a glossary of terms for users new to RealView Developer Suite.

Intended audience

This book has been written for developers who are using RealView Developer Suite to manage development projects for ARM® architecture-based processors. It assumes that you are an experienced software developer, but might not be familiar with the ARM development tools.

Using this book

This book is organized into the following chapters:

Chapter 1 ***Introduction***

Read this chapter for an introduction to RealView Developer Suite v2.0 components, licensing, and documentation.

Chapter 2 ***Features of the RealView Developer Suite v2.0 Components***

Read this chapter for a description of the features of the component applications of RealView Developer Suite v2.0.

Chapter 3 ***Differences***

Read this chapter for a description of the differences between RealView Developer Suite v2.0 and *ARM Developer Suite*™ (ADS) v1.2.

Glossary An alphabetically arranged glossary that defines the special terms used.

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 both ARM Limited and third parties that provide additional information.

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

ARM publications

Refer to the following documentation for details on the FLEX*lm* license management system, supplied by GLOBEtrotter Inc., that controls the use of ARM applications:

- *ARM FLEXlm License Management Guide* (ARM DUI 0209).

This book is part of the RealView Developer Suite documentation suite. Other books in this suite include:

- *RealView Debugger Essentials Guide* (ARM DUI 0181)
- *RealView Debugger User Guide* (ARM DUI 0153)
- *RealView Debugger Target Configuration Guide* (ARM DUI 0182)
- *RealView Debugger Command Line Reference Guide* (ARM DUI 0175)
- *RealView Debugger Extensions User Guide* (ARM DUI 0174)
- *RealView Compilation Tools Essentials Guide* (ARM DUI 0202)

- *RealView Compilation Tools Developer Guide* (ARM DUI 0203)
- *RealView Compilation Tools Assembler Guide* (ARM DUI 0204)
- *RealView Compilation Tools Compiler and Libraries Guide* (ARM DUI 0205)
- *RealView Compilation Tools Linker and Utilities Guide* (ARM DUI 0206)
- *RealView ARMulator ISS User Guide* (ARM DUI 0207).

If you are using ADS v1.2, refer to the following books in the ADS document suite for more information:

- *Getting Started Guide* (ARM DUI 0064)
- *ADS Compilers and Libraries Guide* (ARM DUI 0067)
- *ADS Linker and Utilities Guide* (ARM DUI 0151)
- *CodeWarrior IDE Guide* (ARM DUI 0065)
- *AXD and armsd Debuggers Guide* (ARM DUI 0066)
- *ADS Assembler Guide* (ARM DUI 0068)
- *ADS Debug Target Guide* (ARM DUI 0058)
- *ADS Developer Guide* (ARM DUI 0056).

The following documentation provides general information on the ARM architecture, processors, associated devices, and software interfaces:

- *ARM Architecture Reference Manual* (ARM DUI 0100). This is available as a printed book:

  David Seal, *ARM Architecture Reference Manual, Second Edition*, 2001, Addison Wesley. ISBN 0-201-73719-1

- *ARM Reference Peripheral Specification* (ARM DDI 0062)

- *ARM-Thumb® Procedure Call Standard (ATPCS) Specification* (SWS ESPC 0002).

Refer to the following documentation for information relating to the ARM debug interfaces suitable for use with RealView Developer Suite:

- *ARM Agilent Debug Interface User Guide* (ARM DUI 0158)
- *RealView ICE Version 1.0.1 User Guide* (ARM DUI 0155)
- *Multi-ICE® Version 2.2 User Guide* (ARM DUI 0048).

Other publications

For a comprehensive introduction to ARM architecture see:

Steve Furber, *ARM system-on-chip architecture* (2nd edition, 2000). Addison Wesley, ISBN 0-201-67519-6.

Feedback

ARM Limited welcomes feedback on both RealView Developer Suite and its documentation.

Feedback on RealView Developer Suite

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

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

Note

If you have any problems with RealView Debugger, you can create a Software Problem Report using the **Help → Send a Problem Report...** menu. See the RealView Debugger documentation for more details.

Feedback on this book

If you have any comments on this book, send email to `errata@arm.com` giving:

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

General suggestions for additions and improvements are also welcome.

Chapter 1
Introduction

This chapter introduces RealView® Developer Suite v2.0. It describes the component applications, the additional licenses you can purchase to extend the features of RealView Developer Suite v2.0, and gives an overview of the documentation suite.

This chapter contains the following sections:

1.1 RealView Developer Suite v2.0 components

RealView Developer Suite v2.0 provides a coordinated development environment for embedded systems applications running on the ARM® family of RISC processors. It consists of a suite of tools, together with supporting documentation and examples. The tools enable you to write, build, and debug your applications, either on target hardware or software simulators.

This section includes:

- *RealView Debugger*
- *RealView Compilation Tools* on page 1-3
- *RealView ARMulator Instruction Set Simulator* on page 1-4.

1.1.1 RealView Debugger

RealView Debugger, together with a supported debug target, enables you to debug your application programs and have complete control over the flow of the program execution so that you can quickly isolate and correct errors.

You can also build your applications from RealView Debugger because it provides a build interface to the RealView Compilation Tools (see *RealView Compilation Tools* on page 1-3). From this interface you can specify the build options required by your applications. You can also edit your application source code using the built-in text editor.

The default license for RealView Debugger enables you to debug applications that run on a single processor. However, you can purchase additional licenses to extend the RealView Debugger functionality to debug applications running on multiple processors, and support debugging on DSP and RTOS. See *RealView Developer Suite v2.0 licensing* on page 1-5 for more details.

For more details on the features available in RealView Debugger, see *RealView Debugger features* on page 2-2.

RealView Debugger short path names

The RealView Debugger short path names shown in Table 1-1 are used throughout the debugger documentation. The *install_directory* shown is the default installation directory. If you specified a different installation directory, then the path names are relative to your chosen directory.

Table 1-1 RealView Debugger short path names

Short name	Default Directory
install_directory	C:\Program Files\ARM (on Windows) ~/arm (on Sun Solaris and Red Hat Linux)
program_directory	*install_directory*\RVD\Core*version**build_num**platform*
examples_directory	*install_directory*\RVD\Examples*version**build_num**platform*

1.1.2 RealView Compilation Tools

You can use the *RealView Compilation Tools* (RVCT) to build C, C++, or ARM assembly language programs. RVCT comprises the following tools:

- ARM and Thumb® assembler, armasm
- ARM and Thumb C and C++ compiler, armcc
- ARM linker, armlink
- ARM librarian, armar
- ARM image conversion utility, fromelf
- supporting libraries.

You can use the RVCT tools from the command line interface (CLI). However, RealView Debugger provides a self-contained environment for building your applications, and then debugging them. If you want to create libraries or convert images to another format, you can configure RealView Debugger to automatically run the armar and fromelf utilities after a successful build. See the *RealView Debugger User Guide* for details.

Note

RealView Debugger contains a built-in CLI. You can run the build tools from this built-in CLI without having to open another window on your system. To do this, use the RealView Debugger HOST command. You can also use this command in scripts. See the *RealView Debugger Command Line Reference Guide* for more details.

For more details on the features available in RealView Compilation Tools, see *RealView Compilation Tools features* on page 2-9.

RVCT short path names

The RVCT short path names shown in Table 1-2 are used throughout the RVCT documentation. The *install_directory* shown is the default installation directory. If you specified a different installation directory, then the path names are relative to your chosen directory.

Table 1-2 RVCT short path names

Short name	Default Directory
install_directory	C:\Program Files\ARM (on Windows) ~/arm (on Sun Solaris and Red Hat Linux)
program_directory	*install_directory*\RVCT\Programs*version**build_num**platform*
includes_directory	*install_directory*\RVCT\Data*version**build_num*\include
examples_directory	*install_directory*\RVCT\Examples*version*\release*platform*

1.1.3 RealView ARMulator Instruction Set Simulator

RealView ARMulator® Instruction Set Simulator, enables you to begin developing and debugging your embedded applications without target hardware. This is useful where hardware is still being developed, or there is a limited number of development boards available.

For more details on the features available in RealView ARMulator ISS, see *RealView ARMulator Instruction Set Simulator features* on page 2-12.

1.2 RealView Developer Suite v2.0 licensing

All licensing for RealView Developer Suite v2.0 is controlled by the FLEX*lm* license management system. Use the FLEX*lm* server software to track and control your RVDS licenses. See *ARM FLEXlm License Management Guide* for details.

Note

The licensing requirements for RealView Developer Suite are different from those mentioned in the *RealView Debugger Extensions User Guide*. These differences are as follows:

- The Trace, analysis and profiling feature is available by default in RealView Developer Suite, and you do not require a separate license.
- You must have a separate license to debug applications running on the Motorola M56621 DSP.

The RealView Developer Suite v2.0 licenses available are described in:

- *Multiprocessor debugging license*
- *DSP debugging license*
- *Motorola M56621 debugging license* on page 1-6
- *RTOS awareness* on page 1-6.

1.2.1 Multiprocessor debugging license

The multiprocessor debugging license enables you to debug software systems running on more than one processor. The processors can be on a single development board, or on multiple development boards. In both cases, RealView Debugger uses a different connection for each processor.

With multiprocessor debugging you can debug mixed core systems and synchronize processor operations.

For more details on multiprocessor debugging, see the chapter on working with multiple target connections in the *RealView Debugger Extensions User Guide*.

1.2.2 DSP debugging license

The *Digital Signal Processor* (DSP) support license enables you to debug applications running on Oak and Teaklite DSPs.

For more details on DSP support, see the chapter on DSP support in the *RealView Debugger Extensions User Guide*.

1.2.3 Motorola M56621 debugging license

The Motorola M56621 debugging license enables you to debug applications running on the Motorola M56621 DSP.

For more details on DSP support, see the chapter on DSP support in the *RealView Debugger Extensions User Guide*.

1.2.4 RTOS awareness

The RTOS extension is not licensed by ARM Limited. You must obtain the RealView Debugger support package for the RTOS you are using before you can use this extension. Select **Goto RealView RTOS Awareness Downloads** from the RealView Debugger Code window **Help** menu for information on how to do this.

For more details on RTOS support, see the chapter on RTOS support in the *RealView Debugger Extensions User Guide*.

1.3 RealView Developer Suite v2.0 documentation

This section describes the documentation provided with RealView Developer Suite. It contains the following sections:

- *List of documents*
- *Getting more information online*.

1.3.1 List of documents

The RealView Developer Suite documentation comprises:

- *RealView Debugger Essentials Guide*
- *RealView Debugger User Guide*
- *RealView Debugger Target Configuration Guide*
- *RealView Debugger Command Line Reference Guide*
- *RealView Debugger Extensions User Guide*
- *RealView Compilation Tools Essentials Guide*
- *RealView Compilation Tools Developer Guide*
- *RealView Compilation Tools Assembler Guide*
- *RealView Compilation Tools Compiler and Libraries Guide*
- *RealView Compilation Tools Linker and Utilities Guide*
- *RealView ARMulator ISS User Guide*
- *RealView ARMulator ISS User Guide Addendum*.

See the *Further Reading* sections in each book for related publications from ARM Limited, and from third parties.

1.3.2 Getting more information online

The full documentation suite is available online as PDF files and is installed for a Full installation or if you chose to install them during a Custom installation.

Other forms of online documentation are available depending on your platform:

- On Windows and Sun Solaris systems, the documentation is available online as DynaText electronic books. The content of the DynaText manuals is identical to that of the PDF documentation.

  The documentation is installed in the following directory:

  — On Windows:
  install_directory\Documentation*product**version*\release\windows.

  — On Sun Solaris:
  install_directory/Documentation/*product*/*version*/release/unix.

- On Red Hat Linux, the documentation is available online as XML formatted manuals. The content of the XML manuals is identical to that of the PDF documentation.

  The documentation is installed in the following directory:

  install_directory/Documentation/*product*/*version*/release/unix.

  To access the XML documentation, you must use either:

  — Netscape 6.2

  — Mozilla 1.0.

product is RVD, RVCT, and RVArmulatorISS (on Windows) or RVISS (on Sun Solaris and Linux).

On Windows, you can access the documentation from the Start menu:

Start → Programs → ARM

In addition, documentation for the Rogue Wave C++ library is available in HTML format. The documentation is available in:

install_directory\Documentation\RogueWave\1.0\release

1.4 ARM Developer Suite

RealView Developer Suite v2.0 also includes *ARM Developer Suite*™ (ADS) v1.2.1, for Windows only. This is not installed as part of the RealView Developer Suite v2.0 installation. You must install this separately, if you have to use ADS.

If you are recommended to use an older version of ADS, these are also provided on the same CD-ROM as ADS v1.2.1.

Note

Although you can install ADS in addition to RealView Developer Suite v2.0, you must exercise caution if you use both ADS ARMulator and RealView ARMulator ISS. See the *RealView Developer Suite v2.0 Release Notes* for details. However, if you have installed ADS, you can connect to RealView ARMulator ISS using the ADS debuggers.

Chapter 2
Features of the RealView Developer Suite v2.0 Components

This chapter introduces *RealView® Developer Suite v2.0* and describes its software components and documentation. It contains the following sections:

2.1 RealView Debugger features

This section describes the features available in RealView Debugger. It contains the following sections:

- *RealView Debugger concepts and terminology*
- *About the debugging environment* on page 2-3
- *Multi-core debugging* on page 2-6
- *OS awareness* on page 2-6
- *Extended Target Visibility (ETV)* on page 2-6
- *Advanced debugging facilities* on page 2-7
- *Trace, Analysis, and Profiling* on page 2-7
- *Project manager* on page 2-8
- *RealView Debugger downloads* on page 2-8.

2.1.1 RealView Debugger concepts and terminology

The following terminology is used throughout the RealView Debugger documentation suite to describe debugging concepts:

Debug target

A piece of hardware or a simulator that runs your application program. A hardware debug target might be a single processor, or a development board containing a number of processors.

Connection The link between the debugger program and the debug target.

Single connection access

The RealView Debugger base functionality enables you to carry out debugging tasks in single-processor debugging mode, that is where there is only one target connection.

Multiprocessor access

RealView Debugger has been developed as a fully-featured debugger for working with multiprocessor debug target systems. Multiprocessor access enables you to maintain one or more connections to debug targets. Multiprocessor access is a separately licensed feature of RealView Debugger (see *Multiprocessor debugging license* on page 1-5).

DSP RealView Debugger has been developed to provide full debugging functions when working with a range of debug target systems including *Digital Signal Processors* (DSPs). DSP-based debugging is a separately licensed feature of RealView Debugger (see *DSP debugging license* on page 1-5 and *Motorola M56621 debugging license* on page 1-6).

RTOS Operating systems provide software support for application programs running on a target. *Real Time Operating Systems* (RTOSs) are operating systems that are designed for systems that interact with real-world activities where time is critical. You must download the required RealView Debugger support package (see *RTOS awareness* on page 1-6).

Multithreaded operation

An RTOS process can include separate execution paths, known as *threads*. Each thread shares the same code and global data, but has its own stack. RealView Debugger enables you to:

- attach Code windows to threads to monitor one or more threads
- select individual threads to display the registers, variables, and code related to that thread
- change the register and variable values for individual threads.

2.1.2 About the debugging environment

This section describes the RealView Debugger debugging environment, and includes:

- *Graphical User Interface*
- *Target Vehicle Server* on page 2-4
- *RealView Connection Broker* on page 2-5
- *Debug target interface* on page 2-6.

Graphical User Interface

The graphical user interface (GUI) gives access to the main features of the debugger, command processing, and the Code windows. The GUI also includes a command line interface (CLI), where you can enter CLI commands, run command scripts, and define macros.

Figure 2-1 on page 2-4 shows the default state of the Code window when you run RealView Debugger for the first time after installation.

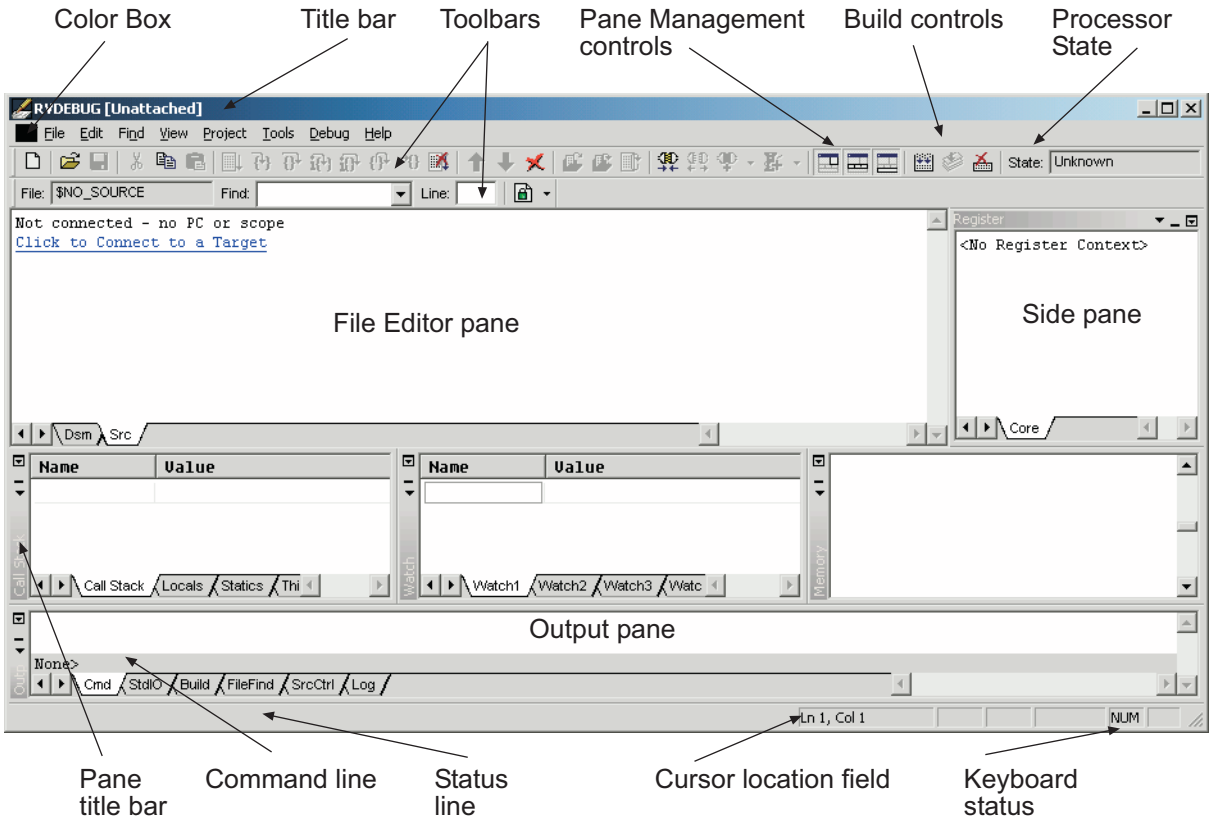

Figure 2-1 Default Code window

Target Vehicle Server

RealView Debugger maintains connections through the Target Vehicle Server (TVS) and plugins that support each combination of target processor and execution vehicle. Certain files, for example a board file (`*.brd`), and board-chip definition files (`*.bcd`), enable RealView Debugger to store advanced information about your target hardware or processor (see *Extended Target Visibility (ETV)* on page 2-6).

Note

When performing multiprocessor debugging with multiple development boards, you must connect to each development board using a different execution vehicle.

The execution vehicles and target processors are listed in the Connection Control window of RealView Debugger. Figure 2-2 shows an example of the Connection Control window. To access this window, you can click on the blue text in the File Editor pane.

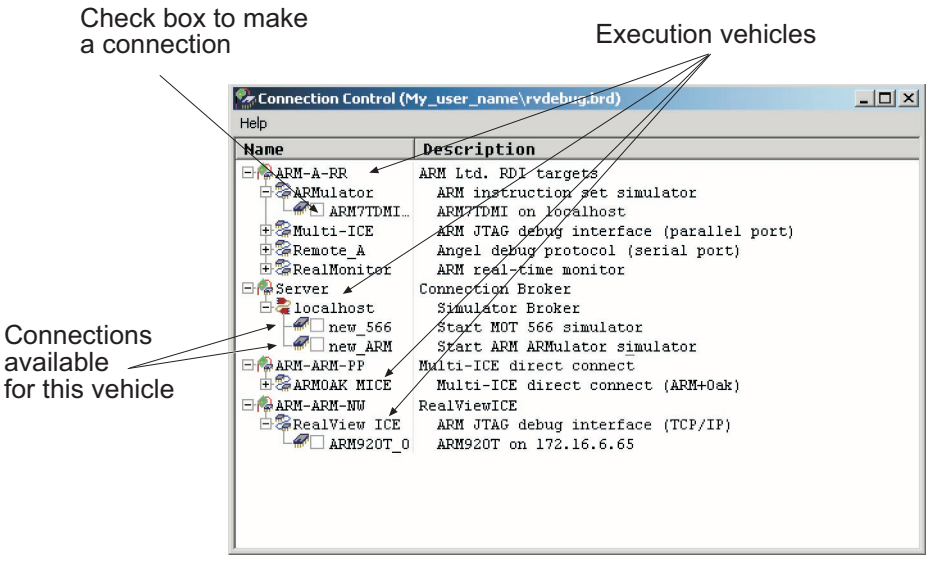

Figure 2-2 Connection Control window

Your Connection Control window might have different execution vehicles and target processors, depending on what execution vehicles you have installed, and what target processors you have configured.

RealView Connection Broker

RealView Connection Broker operates in two modes:

Local Operating as RealView Connection Broker, this runs on your local workstation and enables you to access local targets.

Remote Operating as RealView Network Broker, this runs on a remote workstation and makes specified targets on that workstation available to other workstations connected to the same network.

You make a RealView Connection Broker connection using the localhost execution vehicle in the Connection Control window (see Figure 2-2).

Debug target interface

RealView Debugger works with either a hardware or a software debug target. An ARM® development board communicating through RealView ICE is an example of a hardware debug target system. RealView ARMulator® Instruction Set Simulator is an example of a software debug target system.

The debug target interface processes requests from the client tools to the target. A debug interface might be a JTAG interface unit such as RealView ICE, or a ROM monitor.

2.1.3 Multi-core debugging

RealView Debugger provides a single debug kernel for mixed ARM core and DSP core debugging. The debugger provides full support for synchronized start and stop, stepping, and cross triggering of breakpoints.

In RealView Debugger you can have connections to multiple debug targets through one or more Code windows (see Figure 2-1 on page 2-4). When working in multiprocessor debugging mode, you can use one Code window to cycle through the connected targets, or multiple Code windows to view different targets.

Multiprocessor debugging mode is a separately licensed feature of RealView Debugger and is described in detail in *RealView Debugger Extensions User Guide*. For details on licensing, see *RealView Developer Suite v2.0 licensing* on page 1-5.

2.1.4 OS awareness

RealView Debugger enables you to:

- use RTOS debug including *Halted System Debug* (HSD)
- interrogate and display resources after execution has halted
- access semaphores and queues
- view the status of the current thread or other threads
- customize views of application threads.

2.1.5 Extended Target Visibility (ETV)

RealView Debugger provides visibility of targets such as boards and system-on-chip (SoC). You can configure targets using board-chip definition files. Preconfigured files for a variety of ARM targets are provided wqith RealView Debugger:

- ARM family files are provided as part of the installation
- customer/partner board files are provided through ARM DevZone™.

2.1.6 Advanced debugging facilities

RealView Debugger provides standard debug views and advanced debugging features. RealView Debugger includes:

- A command-line interface and scripting capability that includes macros support, conversion from ARM eXtensible Debugger (AXD) and armsd scripts, and history lists to record previous actions.
- A console (headless debugger) driven from the command line or from scripts.
- User control over panes in the Code window and the debug views displayed. RealView Debugger provides the option of using a single Code window to display a wide range of data views during debugging.
- The ability to program Flash modules as standard.
- The ability to enable memory mapping, if required, so that you can define the types of memory regions available on your hardware. Memory views are also colored to indicate the type of memory you have defined in your memory map.

2.1.7 Trace, Analysis, and Profiling

Some ARM processors include a block of logic called Embedded Trace Macrocell™ (ETM) to support tracing.

Trace support is available for:

- ARM ETM v1.0 (ETM7™ and ETM9™), including On-Chip Trace
- ARM ETM v2.0 (ETM10™)
- DSP Group On-Chip Trace (Oak and TeakLite)
- Motorola 56600 On-Chip Trace
- Intel XScale On-Chip Trace.

Trace and Profiling provides full trace support including simple and complex tracepoints and data filtering:

- viewing raw trace
- viewing code trace
- viewing data trace
- viewing disassembly trace
- tracing of function calls
- the profiling of time spent in each function
- the ability to filter captured trace data by field
- the ability to sort captured trace data by field.

You can set tracepoints directly in the source-level view and/or the disassembly-level view. The same functionality is available in the Memory pane so that you can select regions in memory to trace, or trace a specific memory value when it changes.

2.1.8 Project manager

RealView Debugger provides an *Integrated Development Environment* (IDE) including a project manager and build system.

The project manager includes a Configuration Summary window to display the command-line options that are passed to the compiler tools for build target configurations in the current project.

2.1.9 RealView Debugger downloads

ARM Limited provides a range of services to support developers using RealView Debugger. Among the downloads available are OS awareness modules to support RTOS developers and enhanced support for different hardware platforms through technical information and board description files. See `http://www.arm.com` to access these resources from ARM DevZone™.

2.2 RealView Compilation Tools features

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

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

2.2.1 Components of RVCT

RVCT consists of the following major components:

- *Development tools*
- *Utilities* on page 2-10
- *Supporting software* on page 2-11.

Development tools

The following development tools are provided:

armcc The ARM and Thumb® C and C++ compiler. The compiler is tested against the Plum Hall C Validation Suite for ISO conformance. It compiles:

- ISO C source into 32-bit ARM code
- ISO C++ source into 32-bit ARM code
- ISO C source into 16-bit Thumb code
- ISO C++ source into 16-bit Thumb code

armasm The ARM and Thumb assembler. This assembles both ARM assembly language and Thumb assembly language source.

armlink The ARM linker. This combines the contents of one or more object files with selected parts of one or more object libraries to produce an executable program in ELF format, or a partially-linked ELF object. The ARM linker creates ELF executable images.

Rogue Wave C++ library

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

support libraries

The ARM C libraries provide additional components to enable support for C++ and to compile code for different architectures and processors.

Utilities

The following utility tools are provided to support the main development tools:

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

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

fromELF can also generate text information about the input image, such as code and data size.

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

Supported standards

The industry standards supported by RVCT include:

ar UNIX-style archive files are supported by `armar`.

DWARF2 DWARF2 debug tables are supported by the compiler, linker, and ELF/DWARF2 compatible debuggers.

ISO C The ARM compiler accepts ISO C as input. The option `--strict` can be used to enforce strict ISO compliance.

C++ The ARM compiler supports the full ISO C++ language, except for exceptions.

ELF The ARM tools produce ELF format files. The fromELF utility can translate ELF files into other formats.

EABI The ARM *Embedded Application Binary Interface* (EABI) enables you to use the ARM and Thumb objects and libraries from different producers that support the EABI For more details, see the `ARM EABI home page` on the ARM DevZone. You can access the ARM DevZone from `http://www.arm.com`.

Supporting software

To debug your programs under simulation, or on ARM architecture-based hardware, use RealView Debugger, or any other ELF/DWARF2 compatible debugger.

2.3 RealView ARMulator Instruction Set Simulator features

RealView ARMulator Instruction Set Simulator (RealView ARMulator ISS) is the ARM core simulator. This provides instruction-accurate simulation of ARM processors, and enables ARM and Thumb executable programs to be run on non-native hardware. RealView ARMulator ISS provides a series of modules that:

- model the ARM processor core
- model the memory used by the processor.

There are alternative predefined models for each of these parts. However, you can create your own models if a supplied model does not meet your requirements. For more details, see the *RealView ARMulator ISS User Guide*.

Note

If you connect to RealView ARMulator ISS through RealView Connection Broker (see *RealView Connection Broker* on page 2-5), you can connect to multiple instances of RealView ARMulator ISS.

Chapter 3
Differences

This chapter describes the features of RealView® Developer Suite and highlights changes in functionality between RealView Developer Suite v2.0 and ARM Developer Suite™ v1.2.1. The changes are described in *Changes between RVDS v2.0 and ADS v1.2.1* on page 3-2.

3.1 Changes between RVDS v2.0 and ADS v1.2.1

This section describes the changes between RealView Developer Suite v2.0 and ADS v1.2.1. It contains the following sections:

- *Debugger tool support*
- *Build tool support*
- *ARM simulator support* on page 3-7
- *Target vehicle support* on page 3-7.

3.1.1 Debugger tool support

The differences between the debugging tools in RVDS v2.0 and ADS v1.2.1 are:

- Debugging is performed using RealView Debugger instead of *ARM® eXtended Debugger* (AXD) and *ARM Symbolic Debugger* (`armsd`).
- Project management is done in RealView Debugger instead of in the CodeWarrior IDE.

For a complete description of RealView Debugger and how to use it, see the RealView Debugger documentation listed in *RealView Developer Suite v2.0 documentation* on page 1-7.

3.1.2 Build tool support

The differences between the build tools in RVDS v2.0 and ADS v1.2.1 are described in the following sections:

- *General changes* on page 3-3
- *Changes to the ARM assembler* on page 3-4
- *Changes to the ARM compiler* on page 3-4
- *Changes to the ARM linker* on page 3-5
- *Changes to the fromELF utility* on page 3-6.

For a complete description of the RealView Compilation Tools tools and utilities, and how to use them, see the RealView Compilation Tools documentation listed in *RealView Developer Suite v2.0 documentation* on page 1-7.

General changes

The most important differences between the build tools in RVDS v2.0 and ADS v1.2.1 are:

- The RVCT compiler:
  - There is a new front-end to the RVCT 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 merged into a single compiler, `armcc`. However, to aid migration to the new compiler, you can invoke the RVCT compiler using the individual compiler names.
- Compliance with the ARM *Embedded Application Binary Interface* (EABI). See the `ARM EABI home page` on the ARM DevZone™. You can access the ARM DevZone from `http://www.arm.com/`.
- There is full ISO C++ support as defined by the *ISO/IEC 14822 :1998 International Standard for C++*, by way of the EDG (Edison Design Group) front-end. This includes namespaces, templates, and intelligent implementation of *Run-Time Type Information* (RTTI), but excludes exceptions.
- Full support for ARM architecture v6 instructions has been added.
- Support for double dashes -- to indicate command-line keywords (for example, `--cpp`) and single dashes - for command-line single-letter options, with or without arguments (for example, `-S`).

  Note

  The single-dash command-line options used in previous versions of ADS and RVCT are still supported for backwards-compatibility.
- Re-engineered inline assembler and new embedded assembler.
- ARM and Thumb® compilation on a per-function basis.
- Unicode and multibyte characters are supported.
- The *ARM Profiler* (`armprof`) is not provided with RVCT.
- The ARM Applications Library is not provided with RVCT.
- There are no temporary licenses.

Changes to the ARM assembler

The following changes have been made to the ARM assembler (armasm):

- 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}.

Changes to the ARM compiler

The major changes that have been made to the ARM compiler (armcc) are as follows:

- There is a new front-end to the RVCT 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 compiler using the individual compiler names.
- A new embedded assembler to complement the inline assembler.
- ARM and Thumb compilation can be done on a per-function basis, using #pragma arm and #pragma thumb.
- Five floating-point models are available using the --fpmode option.
- The behavior of the --list option is different from that in the older compilers.
- You can specify the level of pointer alignment.
- C++ template instantiation.
- C++ namespaces.
- Control of diagnostic messages, and re-assignment to a different level of severity. 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.

- Some old compiler options are not supported in the new front-end. However, for backwards compatibility, these options are available if you use the `--old_cfe` option. See the appendix describing the older compiler options in the *RealView Compilation Tools 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 the *RealView Compilation Tools 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.

Changes to the ARM linker

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

- The `-unresolved` option can now be used when performing 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.
- The linker is now stricter in checking alignment build attributes in object files, and emits an error message if a stack alignment conflict is detected:

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

  The ABI for the ARM Architecture demands that code maintains 8-byte stack alignment at its interfaces. This link error typically occurs when:

  — assembly language code that does not preserve 8-byte stack alignment calls compiled C/C++ code that requires 8-byte stack alignment

  — you attempt to link legacy SDT or ADS object with RVCT 2.x objects. SDT and some ADS objects do not have stack alignment build attributes and are treated as not preserving 8-byte alignment, regardless of whether or not they do preserve 8-byte alignment.

A similar warning message is emitted where the address of an external symbol is being referred to:

`L6306W: '~PRES8' section object_name.o(section_name should no use the address of 'REQ8' function function_name`

To fix this problem:

— Rebuild all your objects and libraries using RVCT 2.x, if possible. If you have assembly language files check that they preserve 8-byte alignment and correct them if required. For example, change:

```
STMFD sp!, {r0-r3, lr}  ; push an odd number of registers
```

to:

```
STMFD sp!, {r0-r3, r12, lr}  ; push an even number of registers
```

and add the `PRESERVE8` directive to the top of each assembly language source file:

```
PRESERVE8
AREA Init, CODE, READONLY
```

— If you cannot rebuild legacy objects or libraries, use `fromelf -c` to disassemble the object code and check to determine if it preserves 8-byte alignment.

If the code does preserve 8-byte alignment, use the linker option `--diag_suppress 6238` to suppress the error message, or `--diag_suppress 6306` to suppress the warning message.

Note

If you suppress the error message for an object that does not preserve 8-byte alignment, your code might fail at runtime.

For more information on linking legacy objects and libraries and the `--apcs /adsabi` compiler option, refer to `http://www.arm.com/support/rvct2_faq`.

Changes to the fromELF utility

The following changes have been made to the fromELF utility (`fromelf`):

- The `-ihf` option has been removed.

3.1.3 ARM simulator support

The ARM simulators supported in RVDS v2.0 and ADS v1.2.1 are described in the following sections:

- *RealView ARMulator ISS*
- *ADS ARMulator*.

Note

Although you can install ADS in addition to RealView Developer Suite v2.0, you must exercise caution if you use both RealView ARMulator ISS and ADS ARMulator. See the *RealView Developer Suite v2.0 Release Notes* for more details. However, if you have installed ADS, you can connect to RealView ARMulator ISS using the ADS debuggers.

RealView ARMulator ISS

RealView ARMulator ISS supports connections through RealView Connection Broker and RDI. When connecting to the simulator through RealView Connection Broker, you can have multiple connections to the simulator.

ADS ARMulator

ADS ARMulator supports connections only through RDI.

3.1.4 Target vehicle support

The target vehicle support in RVDS v2.0 and ADS v1.2 is shown in Table 3-1.

Table 3-1 Target vehicle support

RVDS v2.0 support	ADS v1.2 support
Multi-ICE	Multi-ICE
RealView ICE	MultiTrace
RealView Trace	

Glossary

The items in this glossary are listed in alphabetical order, with any symbols and numerics appearing at the end.

ADS *See* ARM Developer Suite.

American National Standards Institute (ANSI)
An organization that specifies standards for, among other things, computer software. This is superseded by the International Standards Organization.

Angel Angel is a software debug monitor that runs on the target and enables you to debug applications running on ARM-based hardware. Angel is commonly used where a JTAG emulator is not available.

ANSI *See* American National Standards Institute.

APCS ARM Procedure Call Standard.

ARM Developer Suite (ADS)
A suite of software development applications, together with supporting documentation and examples, that enable you to write and debug applications for the ARM family of *RISC* processors.

See also RealView Developer Suite.

armasm The ARM assembler.

armcc The ARM C compiler.

armlink The ARM C linker.

ARM-Thumb Procedure Call Standard (ATPCS)
Defines how registers and the stack are used for subroutine calls.

ARMulator ARMulator is the ARM instruction set simulator provided with ADS. It is a collection of modules that simulate the instruction sets and architecture of various ARM processors. You connect to the ADS ARMulator using the RDI interface. You can have only one connection to ADS ARMulator.

See also RealView ARMulator Instruction Set Simulator.

ATPCS ARM-Thumb Procedure Call Standard.

Big-endian Memory organization where the least significant byte of a word is at the highest address and the most significant byte is at the lowest address in the word.

See also Little-endian.

Board RealView Developer Suite uses the term *board* to refer to a target processor, memory, peripherals, and debugger connection method.

Board file The *board file* is the top-level configuration file, normally called `rvdebug.brd`, that references one or more other files.

Coprocessor An additional processor used for certain operations. Usually used for floating-point math calculations, signal processing, or memory management.

DWARF Debug With Arbitrary Record Format.

ELF Executable and Linking Format.

Embedded Trace Macrocell (ETM)
A block of logic, embedded in the hardware, that is connected to the address, data, and status signals of the processor. It broadcasts branch addresses, and data and status information in a compressed protocol through the trace port. It contains the resources used to trigger and filter the trace output.

ETM *See* Embedded Trace Macrocell.

Execution vehicle Part of the debug target interface, execution vehicles process requests from the client tools to the target.

Image An execution file that has been loaded onto a processor for execution.

Integrator A range of ARM hardware development platforms. *Core modules* are available that contain the processor and local memory.

Joint Test Action Group (JTAG)
An IEEE group focussed on silicon chip testing methods. Many debug and programming tools use a *Joint Test Action Group* (JTAG) interface port to communicate with processors. For further information refer to IEEE Standard, Test Access Port and Boundary-Scan Architecture specification 1149.1 (JTAG).

JTAG *See* Joint Test Action Group.

JTAG interface unit A protocol converter that converts low-level commands from RealView Developer Suite into JTAG signals to the EmbeddedICE logic and the ETM.

Little-endian Memory organization where the least significant byte of a word is at the lowest address and the most significant byte is at the highest address of the word.

See also Big-endian.

Multi-ICE A JTAG-based tool for debugging embedded systems.

Profiling Accumulation of statistics during execution of a program being debugged, to measure performance or to determine critical areas of code.

RDI *See* Remote Debug Interface.

RealView ARMulator Instruction Set Simulator (RealView ARMulator ISS)
RealView ARMulator Instruction Set Simulator is the latest instruction set simulator from ARM Limited. It is a collection of modules that simulate the instruction sets and architecture of various ARM processors. You connect to RealView ARMulator ISS using the RealView Connection Broker. Unlike the ADS ARMulator, you can have mutiple connections to RealView ARMulator ISS. See the *RealView ARMulator ISS User Guide* for more information.

See also ARMulator.

RealView Compilation Tools (RVCT)
RealView Compilation Tools is a suite of tools, together with supporting documentation and examples, that enables you to write and build applications for the ARM family of *RISC* processors.

RealView Debugger Trace
Part of RealView Debugger that extends the debugging capability with the addition of real-time program and data tracing.

Remote Debug Interface (RDI)
The *Remote Debug Interface* (RDI) is an ARM standard procedural interface between a debugger and the debug agent. RDI gives the debugger a uniform way to communicate with:

- a simulator running on the host (for example, ADS ARMulator)

- a debug monitor running on ARM architecture-based hardware accessed through a communication link (for example, Angel)
- a debug agent controlling an ARM processor through hardware debug support (for example, Multi-ICE).

RealView Developer Suite
: The latest suite of software development applications from ARM Limited, together with supporting documentation and examples, that enable you to write and debug applications for the ARM family of *RISC* processors.

  See also ARM Developer Suite.

RealView ICE (RVI)
: RealView EmbeddedICE interface. An EmbeddedICE logic debug solution from ARM Limited, which enables you to debug software running on ARM processor cores that include the EmbeddedICE logic.

RISC
: Reduced Instruction Set Computer.

RTOS
: Real Time Operating System.

RVCT
: *See* RealView Compilation Tools.

Simulator
: A simulator executes non-native instructions in software (simulating a core).

  See also ARMulator and RealView ARMulator Instruction Set Simulator.

Target
: The target hardware, including processor, memory, and peripherals, real or simulated, on which the target application is running.

Target Vehicle Server (TVS)
: Essentially the RealView Debugger itself, this contains the basic debugging functionality. TVS contains the run control, base multitasking support, much of the command handling, target knowledge, such as memory mapping, lists, rule processing, board-files and `.bcd` files, and data structures to track the target environment.

Tracing
: The real-time recording of processor activity (including instructions and data accesses) that occurs during program execution. Trace information can be stored either in a trace buffer of a processor, or in an external trace hardware unit. Captured trace information is returned to the Analysis window in RealView Debugger where it can be analyzed to help identify a defect in program code.

TVS
: *See* Target Vehicle Server.

Index

The items in this index are listed in alphabetical order, with symbols and numerics appearing at the end. The references given are to page numbers.

A

B

C

D

E

F

G

L

M

N

O

P

Q

R

S

T

