

# RealView<sup>®</sup> Developer Suite

Version 2.2

## Getting Started Guide



# RealView Developer Suite

## Getting Started Guide

Copyright © 2003, 2004 ARM Limited. All rights reserved.

### Release Information

The following changes have been made to this book.

| Change History |       |                   |
|----------------|-------|-------------------|
| Date           | Issue | Change            |
| September 2003 | A     | RVDS v2.0 Release |
| January 2004   | B     | RVDS v2.1 Release |
| December 2004  | C     | RVDS v2.2 Release |

### Proprietary Notice

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

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

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

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

### Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

### Product Status

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

### Web Address

<http://www.arm.com>

# Contents

## RealView Developer Suite Getting Started Guide

### Preface

|                       |    |
|-----------------------|----|
| About this book ..... | vi |
| Feedback .....        | x  |

### Chapter 1

#### Introduction

|     |                                                   |      |
|-----|---------------------------------------------------|------|
| 1.1 | RealView Developer Suite v2.2 components .....    | 1-2  |
| 1.2 | RealView Developer Suite v2.2 licensing .....     | 1-9  |
| 1.3 | RealView Developer Suite v2.2 documentation ..... | 1-10 |
| 1.4 | RealView Developer Suite Examples .....           | 1-11 |
| 1.5 | ARM Developer Suite .....                         | 1-13 |
| 1.6 | Target vehicle support .....                      | 1-14 |

### Chapter 2

#### Differences

|     |                                                |     |
|-----|------------------------------------------------|-----|
| 2.1 | Changes between RVDS v2.2 and RVDS v2.1 .....  | 2-2 |
| 2.2 | Changes between RVDS v2.1 and RVDS v2.0 .....  | 2-4 |
| 2.3 | Changes between RVDS v2.2 and ADS v1.2.1 ..... | 2-6 |

### Chapter 3

#### Getting Started with RealView Developer Suite

|     |                                            |     |
|-----|--------------------------------------------|-----|
| 3.1 | Building and debugging task overview ..... | 3-2 |
| 3.2 | Using the example projects .....           | 3-5 |

## Chapter 4

### Getting Started with CodeWarrior for RVDS

|      |                                                                       |      |
|------|-----------------------------------------------------------------------|------|
| 4.1  | About CodeWarrior for RVDS .....                                      | 4-2  |
| 4.2  | Starting to use CodeWarrior for RVDS .....                            | 4-3  |
| 4.3  | Working with an existing project .....                                | 4-5  |
| 4.4  | Overview of the ARM project stationery supplied with RVDS .....       | 4-10 |
| 4.5  | Summary of the build target configuration panels .....                | 4-14 |
| 4.6  | Using the Thumb ARM Interworking Image stationery .....               | 4-27 |
| 4.7  | Creating binary ROM images .....                                      | 4-33 |
| 4.8  | Creating a project that builds an object library and ROM images ..... | 4-34 |
| 4.9  | Creating a subproject within another project .....                    | 4-41 |
| 4.10 | Running batch files with the batch runner .....                       | 4-49 |
| 4.11 | Creating your own project stationery .....                            | 4-53 |
| 4.12 | Upgrading CodeWarrior for ADS projects .....                          | 4-56 |
| 4.13 | Creating projects from XML formatted files .....                      | 4-59 |
| 4.14 | Importing makefiles .....                                             | 4-60 |
| 4.15 | Exporting CodeWarrior for RVDS projects in XML format .....           | 4-63 |
| 4.16 | Using CodeWarrior for RVDS from the command line .....                | 4-64 |

## Appendix A

### Using the armenv Tool

|     |                             |     |
|-----|-----------------------------|-----|
| A.1 | About the armenv tool ..... | A-2 |
| A.2 | Using the armenv tool ..... | A-3 |

## Appendix B

### Using the cwfileassoc Tool

|     |                                  |     |
|-----|----------------------------------|-----|
| B.1 | Using the cwfileassoc tool ..... | B-2 |
|-----|----------------------------------|-----|

## Glossary

# Preface

This preface introduces the *RealView® Developer Suite v2.2 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 x.

## 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 differences between RealView Developer Suite v2.2 and *ARM Developer Suite™* (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 v2.2 components, licensing, and documentation.

### **Chapter 2 *Differences***

Read this chapter for a description of the differences between RealView Developer Suite v2.2 and ADS v1.2.

### **Chapter 3 *Getting Started with RealView Developer Suite***

Read this chapter for an overview of the main tasks that you can do with the RVDS tools. It also describes the example projects provided with RVDS.

### **Chapter 4 *Getting Started with CodeWarrior for RVDS***

Read this chapter for instructions on how to start using CodeWarrior for RVDS to create projects and build images targeted at ARM architecture-based processors.

### **Appendix A *Using the armenv Tool***

Read this appendix for a description of how to use the armenv tool.

## Appendix B *Using the cwfileassoc Tool*

Read this appendix for a description of how to use the `cwfileassoc` tool.

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

## Typographical conventions

The following typographical conventions are used in this book:

|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>italic</i>           | Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>bold</b>             | 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.                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <u>monospace</u>        | Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <i>monospace italic</i> | Denotes arguments to commands and functions where the argument is to be replaced by a specific value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>monospace bold</b>   | Denotes language keywords when used outside example code.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| ...                     | At the end of a path name ... denotes that the directories of interest are below the last-specified directory name. The unspecified path names are usually those that are different between operating systems. For example:<br><code>install_directory\ARM\RVD\Examples\...</code><br><br>In the middle of a path name ... denotes that additional directories exist between the directory names specified. The unspecified path names are usually version and build numbers and platform-specific directory names. For example:<br><code>install_directory\ARM\RVD\Core\...\etc</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

See the following documentation for details of the FLEXlm license management system, supplied by GLOBEtrouter Inc., that controls the use of ARM applications:

- *ARM FLEXlm License Management Guide v3.2* (ARM DUI 0209).

Make sure that you use version 3.2 of this document for details on license management in RealView Developer Suite v2.2.

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

- *RealView Debugger v1.8 Essentials Guide* (ARM DUI 0181)
- *RealView Debugger v1.8 User Guide* (ARM DUI 0153)
- *RealView Debugger v1.8 Target Configuration Guide* (ARM DUI 0182)
- *RealView Debugger v1.8 Command Line Reference Guide* (ARM DUI 0175)
- *RealView Debugger v1.8 Extensions User Guide* (ARM DUI 0174)
- *RealView Compilation Tools v2.2 Essentials Guide* (ARM DUI 0202)
- *RealView Compilation Tools v2.2 Developer Guide* (ARM DUI 0203)
- *RealView Compilation Tools v2.2 Assembler Guide* (ARM DUI 0204)
- *RealView Compilation Tools v2.2 Compiler and Libraries Guide* (ARM DUI 0205)
- *RealView Compilation Tools v2.2 Linker and Utilities Guide* (ARM DUI 0206)
- *RealView ARMulator ISS v1.4 User Guide* (ARM DUI 0207)
- *RealView Developer Suite AXD and armsd Debuggers Guide* (ARM DUI 0066).

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

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

For general information on software interfaces and standards supported by ARM, see *install\_directory\Documentation\Specifications\*.

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

- *RealView ICE User Guide* (ARM DUI 0155)
- *Multi-ICE® User Guide* (ARM DUI 0048)
- *ARM MultiTrace™ User Guide* (ARM DUI 0150)
- *ARM Agilent Debug Interface User Guide* (ARM DUI 0158).

## 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](mailto: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 (RVDS) v2.2. It describes the component applications, the additional licenses you can purchase to extend the features of RVDS v2.2, and gives an overview of the documentation suite.

This chapter contains the following sections:

- *RealView Developer Suite v2.2 components* on page 1-2
- *RealView Developer Suite v2.2 licensing* on page 1-9
- *RealView Developer Suite v2.2 documentation* on page 1-10
- *RealView Developer Suite Examples* on page 1-11
- *ARM Developer Suite* on page 1-13
- *Target vehicle support* on page 1-14.

# 1.1 RealView Developer Suite v2.2 components

RVDS v2.2 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 Developer Suite installation, examples, and documentation directories*
- *CodeWarrior for RVDS* on page 1-3
- *RealView Debugger* on page 1-4
- *RealView Compilation Tools* on page 1-6
- *RealView ARMulator Instruction Set Simulator* on page 1-7
- *RVDS example projects* on page 1-7
- *ARM eXtended Debugger* on page 1-7
- *ARM Symbolic Debugger* on page 1-8.

## 1.1.1 RealView Developer Suite installation, examples, and documentation directories

Various RVDS directories that are installed on your system contain useful files. The RVDS documentation refers to these directories where necessary.

All directories can be found below the main installation directory. Also, many of the examples used in the documentation are contained in a single examples directory. Any exceptions are identified where they are referenced.

The main installation, examples, and documentation directories are identified in Table 1-1. 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 Developer Suite directories

| Directory                  | Windows default path                        | Sun Solaris and Red Hat Linux default path  |
|----------------------------|---------------------------------------------|---------------------------------------------|
| <i>install_directory</i>   | C:\Program Files\ARM                        | ~/arm                                       |
| Examples <sup>a</sup>      | <i>install_directory</i> \RVDS\Examples\... | <i>install_directory</i> /RVDS/Examples/... |
| Documentation <sup>b</sup> | <i>install_directory</i> \Documentation\... | <i>install_directory</i> /Documentation/... |

a. See *RealView Developer Suite Examples* on page 1-11 for a summary of the examples provided with RealView Developer Suite.

- b. See *RealView Developer Suite v2.2 documentation* on page 1-10 for more information on accessing the documentation.

### 1.1.2 CodeWarrior for RVDS

CodeWarrior for RVDS is based on Metrowerks CodeWarrior IDE version 5.6. In RVDS v2.2, CodeWarrior for RVDS is supported on Windows XP and Windows 2000 systems only. It is not supplied with RVDS v2.2 on Sun Solaris and Red Hat Linux.

CodeWarrior for RVDS:

- Provides a graphical user interface for managing your software development projects. You can use the CodeWarrior for RVDS to develop C, C++, and ARM assembly language code targeted at ARM and Thumb processors. It speeds up your build cycle by providing:
  - comprehensive project management capabilities
  - code navigation routines to help you locate routines quickly.
- Enables you to configure the ARM tools to compile, assemble, and link your project code.
- Enables you to organize source code files, library files, and configuration settings into a *project*. Each project enables you to create and manage multiple configurations of build target settings. For example, you can compile a build target for debugging and a build target for release, and target your code at hardware based on an ARM7TDMI®. Build targets can share files in the same project while using their own settings.
- Provides:
  - a source code editor that provides syntax coloring, and is integrated with the CodeWarrior for RVDS browser
  - a source code browser that keeps a database of symbols defined in your code, and enables you to navigate through your source code quickly and easily
  - search and replace capabilities that enable you to use grep-style regular expressions, and perform batch searches through multiple files
  - file comparison capabilities that enable you to locate, and optionally merge the differences from one text file to another, and to compare the contents of directories.

For details on how to get started with CodeWarrior for RVDS, see Chapter 4 *Getting Started with CodeWarrior for RVDS*.

### 1.1.3 RealView Debugger

RealView Debugger together with a supported debug target (see *Target vehicle support* on page 1-14), 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.

———— **Note** ————

If you are using RealView Debugger on Sun Solaris or Red Hat Linux, see the appendix that describes RealView Debugger on Sun Solaris and Red Hat Linux in the *RealView Debugger User Guide* for information specific to these operating systems.

RealView Debugger includes the support for:

- multiprocessor debugging (see *Multiprocessor debugging* on page 1-5)
- Digital Signal Processor (DSP) debugging (see *DSP debugging* on page 1-5)
- trace, analysis and profiling (see *Trace, analysis, and profiling* on page 1-5)
- Real-Time Operating System (RTOS) awareness by downloading vendor-specific plugins (see *RTOS awareness* on page 1-5).

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 1-6). 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. See *RealView Developer Suite v2.2 licensing* on page 1-9 for more details.

For more details on the features available in RealView Debugger, see the *RealView Debugger Essentials Guide*.

For a complete description of RealView Debugger and how to use it, see the RealView Debugger documentation. The documentation is listed in *RealView Developer Suite v2.2 documentation* on page 1-10.

## Multiprocessor debugging

Multiprocessor debugging 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.

If you use the same processor on multiple board connections, you might have to create new target descriptions. For more details on creating custom targets, see the *RealView Debugger Target Configuration Guide*.

Multiprocessor debugging requires a separately purchased license. See *RealView Developer Suite v2.2 licensing* on page 1-9 for details.

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

## DSP debugging

RealView Debugger provides support for debugging the Motorola M56621 DSP.

DSP debugging requires a separately purchased license. See *RealView Developer Suite v2.2 licensing* on page 1-9 for details.

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

## Trace, analysis, and profiling

RealView Debugger provides support for tracing with either trace hardware or a hardware simulator. Trace hardware can be processors with Embedded Trace Macrocell™ (ETM), on-chip trace buffers, or a Joint Test Action Group (JTAG) interface unit such as RealView ICE. Basic trace support is provided by RealView ARMulator® ISS hardware simulator.

For more details on the trace features available in RealView Debugger, see the chapter that describes tracing in the *RealView Debugger Extensions User Guide*.

## RTOS awareness

RTOS awareness is an extension that is built into RealView Debugger. You must obtain the plugin for the RTOS you are using before you can use this RealView Debugger extension. With an RTOS plugin, RealView Debugger references RTOS threads and

resources (such as queues, mailboxes, and semaphores) in addition to the C or assembler source-level symbolic debug information. See *RealView Debugger downloads* for details on how to obtain the plugin for your RTOS.

RTOS awareness in RealView Debugger is vendor-independent. You can download and use RTOS plugins from more than one vendor. Therefore, you can develop applications for different OS platforms. If you want to use the RTOS plugins from more than one vendor in the same debugging session, you might have to create new target descriptions. For more details on creating custom targets, see the *RealView Debugger Target Configuration Guide*.

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

## RealView Debugger downloads

You can access various RealView Debugger downloads from the RealView Debugger **Help** menu:

### **Help → ARM on the Web → Goto RTOS Awareness Downloads**

Displays the *RealView Debugger OS Awareness* web page. From here you can locate and download the RTOS plugin of your choice.

### **Help → ARM on the Web → Goto Update and Utility Downloads**

Displays the *RealView Debugger - Updates & Utilities* web page. From here you can locate and download any software updates and utilities.

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

For more details on the features available in RealView Compilation Tools, see the *RealView Compilation Tools Essentials Guide*.

For a complete description of the RealView Compilation Tools tools and utilities, and how to use them, see the RealView Compilation Tools documentation. The documentation is listed in *RealView Developer Suite v2.2 documentation* on page 1-10.

1.1.5 RealView ARMulator Instruction Set Simulator

*RealView ARMulator Instruction Set Simulator* (RVISS) simulates the instruction sets and architecture of ARM processors, together with a memory system and peripherals.

RVISS enables you to begin developing and debugging your embedded applications without target hardware. This is useful where hardware is still being developed, or if there is a limited number of development boards available.

Table 1-2 shows the RVISS interface connections that are available from the ARM debuggers on Windows, Sun Solaris, and Red Hat Linux. If you connect to RVISS through RealView Connection Broker, you can connect to multiple instances of RVISS to simulate a multiprocessor system.

Table 1-2 RealView ARMulator ISS connections supported on each platform

| ARM Debugger      | Remote Debug Interface Connections     | RealView Connection Broker Connections |
|-------------------|----------------------------------------|----------------------------------------|
| RealView Debugger | Windows <sup>a</sup>                   | Windows, Sun Solaris and Red Hat Linux |
| AXD               | Windows                                | Not available                          |
| armsd             | Windows, Sun Solaris and Red Hat Linux | Not available                          |

a. The Remote Debug Interface connection to RVISS in RealView Debugger is deprecated in this release.

For more details on the features available in RVISS, see the *RealView ARMulator ISS User Guide*.

1.1.6 RVDS example projects

Example projects are provided with RVDS. These are located in the directory:

`install_directory\RVDS\Examples\...`

To explore the example projects directory from the Windows **Start** menu, select:

**Programs → ARM → ARM RealView Developer Suite v2.2 → Examples**

1.1.7 ARM eXtended Debugger

*ARM eXtended Debugger* (AXD) is a single-processor debugger, and is provided for Windows only. AXD together with a supported debug target (see *Target vehicle support* on page 1-14), enables you to debug your application programs and have control over the flow of the program execution so that you can quickly isolate and correct errors.

---

**Note**

---

AXD is provided for legacy ARM7 and ARM9 support.

---

For details on how to use AXD, see the *RealView Developer Suite AXD and armsd Debuggers Guide*.

### 1.1.8 ARM Symbolic Debugger

*ARM Symbolic Debugger* (armsd) is a single-processor debugger that you run from a command-line interface. It is available for Windows, Sun Solaris, and Red Hat Linux.

With armsd you can debug your application programs using the *Remote Debug Interface* (RDI) of RVISS (see *RealView ARMulator Instruction Set Simulator* on page 1-7).

---

**Note**

---

RVISS with armsd supports only ARM7 and ARM9 targets.

---

For details on how to use armsd, see the *RealView Developer Suite AXD and armsd Debuggers Guide*.

## 1.2 RealView Developer Suite v2.2 licensing

All licensing for RealView Developer Suite v2.2 is controlled by the FLEXlm license management system. Use the FLEXlm server software to track and control your RVDS licenses. You can now request licenses using the ARM Web Licensing page at <http://license.arm.com>. See the *ARM FLEXlm License Management Guide* for details.

The RealView Developer Suite v2.2 licenses that are separately available for RealView Debugger features are described in:

- *Multiprocessor debugging license*
- *Motorola M56621 DSP debugging license.*

### 1.2.1 Multiprocessor debugging license

The multiprocessor debugging license enables you to debug software systems running on more than one processor (see *Multiprocessor debugging* on page 1-5).

### 1.2.2 Motorola M56621 DSP debugging license

The Motorola M56621 DSP debugging license enables you to debug applications running on the Motorola M56621 DSP (see *DSP debugging* on page 1-5).

## 1.3 RealView Developer Suite v2.2 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 Developer Suite Getting Started Guide* (this document)
- *ARM FLEXlm License Management Guide*
- *RealView Debugger Essentials Guide*
- *RealView Debugger User Guide*
- *RealView Debugger Project Management 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 Developer Suite AXD and armsd Debuggers Guide.*

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. These are installed for a Typical installation, or if you choose to install them during a Custom installation. The documentation is installed in the documentation directory shown in Table 1-1 on page 1-2.

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

On Windows, you can access the documentation from the Start menu, **Start** → **Programs** → **ARM**.

## 1.4 RealView Developer Suite Examples

The code for many of the examples in the RealView developer Suite documentation is located in the main examples directory (see *RealView Developer Suite installation, examples, and documentation directories* on page 1-2).

In addition, the directory contains example code that is not described in the documentation. Read the `readme.txt` in each example directory for more information. The examples are installed in the following subdirectories:

|             |                                                                                                                                                                                                                                         |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| asm         | Some examples of ARM assembly language programming. The examples are used in the <i>RealView Compilation Tools Assembler Guide</i> .                                                                                                    |
| cached_dhry | Examples of routines to initialize cache and TCMs, built around the Dhrystone example.                                                                                                                                                  |
| cpp         | Some simple C++ examples.                                                                                                                                                                                                               |
| databort    | Design documentation and example code for a standard Data Abort handler.                                                                                                                                                                |
| dcc         | Example code that demonstrates how to use the Debug Communications Channel. The example is described in the <i>RealView Compilation Tools Developer Guide</i> .                                                                         |
| dhrystone   | The Dhrystone Benchmark. The example is used in the RealView Debugger documentation.                                                                                                                                                    |
| dsp         | A small source file that is required to make full use of the header file <code>dspfns.h</code> . This file defines a set of DSP-type primitive operations, and demonstrates how to use the inline assembly feature in the ARM compiler. |
| emb_sw_dev  | The example projects referenced in the chapter that describes embedded software development in the <i>RealView Compilation Tools Developer Guide</i> . The following subdirectories are included:                                       |
| buildn      | Batch and make files to build the example projects. See the related <code>readme.txt</code> file for a description of each project.                                                                                                     |
| dhry        | Source files for the Dhrystone benchmarking program. This program provides the code base for the example projects in the individual <code>buildn</code> directories.                                                                    |
| include     | User defined header files.                                                                                                                                                                                                              |
| scatter     | Scatter files used to build the example projects.                                                                                                                                                                                       |
| source      | All other source files needed to build the example projects.                                                                                                                                                                            |

|            |                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| explasm    | Additional ARM assembly language examples.                                                                                                                                                                                                                                                                                                                                                                                       |
| fft_5te    | Fast Fourier Transform code optimized for ARM architecture v5TE.                                                                                                                                                                                                                                                                                                                                                                 |
| inline     | Examples that show how to use the inline assembler when compiling ARM C and C++ code. See the chapter that describes mixing C, C++ and assembly language in the <i>RealView Compilation Tools Developer Guide</i> for details.                                                                                                                                                                                                   |
| interwork  | <p>Examples that show how to interwork between ARM code and Thumb code. See the chapter that describes interworking ARM and Thumb in the <i>RealView Compilation Tools Developer Guide</i> for details.</p> <p>The sources for this example are also used in the CodeWarrior tutorial described in <i>Using the Thumb ARM Interworking Image stationery</i> on page 4-27.</p>                                                    |
| linux      | The source code for a library that retargets the ARM C library to run under ARM Linux.                                                                                                                                                                                                                                                                                                                                           |
| mmugen     | The source and documentation for the MMUgen utility. This utility can generate MMU pagetable data from a rules file that describes the virtual to physical address translation required.                                                                                                                                                                                                                                         |
| picpid     | An example of how to write position-independent code.                                                                                                                                                                                                                                                                                                                                                                            |
| primes     | An example that demonstrates Trace support. The program calculates the nth prime number. However, as given, it deliberately fails with a Data Abort error at runtime.                                                                                                                                                                                                                                                            |
| sorts      | Example code that compares an insertion sort, shell sort, and the quick sort used in the ARM C libraries.                                                                                                                                                                                                                                                                                                                        |
| swi        | An example SWI handler.                                                                                                                                                                                                                                                                                                                                                                                                          |
| unicode    | Example code that enables you to evaluate multibyte character support.                                                                                                                                                                                                                                                                                                                                                           |
| vfpsupport | <p>Example code for enabling and carrying out <i>Vector Floating Point</i> (VFP) operations. Also included are various utility files for configuring the debug system when using VFP, and a PDF of <i>Application Note 133 Using VFP in RVDS</i>.</p> <p>The sources for this example are used in the CodeWarrior tutorial described in <i>Creating a project that builds an object library and ROM images</i> on page 4-34.</p> |

## 1.5 ARM Developer Suite

RealView Developer Suite v2.2 also includes the full version of *ARM Developer Suite™* (ADS) v1.2.1. ADS v1.2.1 is not installed as part of the RealView Developer Suite v2.2 installation. If you have to use ADS v1.2.1, you must install it separately.

---

### Note

---

Although you can install ADS in addition to RealView Developer Suite v2.2, you must exercise caution if you use both the ADS ARMulator that is installed with ADS and RealView ARMulator ISS. See the *RealView Developer Suite v2.1 Release Notes* for details. However, if you have installed ADS, you can connect to RealView ARMulator ISS using the ADS debuggers (see *RealView ARMulator Instruction Set Simulator* on page 1-7).

You can use both CodeWarrior for ADS and CodeWarrior for RVDS on the same workstation.

---

### 1.5.1 Installing the older ADS compilation tools

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.

For Windows, an installer is provided for ADS v1.1 and v1.0.1.

For Sun Solaris and Red Hat Linux, only the compilation tools are provided for ADS v1.1 and v1.0.1. No installer is included for these, and you must copy the files manually. The files are in the `ads_1_1` and `ads_1_0_1` directories on the CD-ROM. You must:

1. Decide what components you need for which platforms, and where you want to copy them.
2. Choose the correct executables from platform-dependent directories.
3. Copy the libraries you require.
4. Make sure that your path includes the location of the executables, and that you set the following environment variables:
  - `ARMINC`, to the directory containing the ARM include files
  - `ARMLIB`, to the directory containing the ARM library files.

## 1.6 Target vehicle support

The target vehicles supported by the ARM debuggers in the RVDS v2.2 and ADS v1.2.1 are shown in Table 1-3.

Table 1-3 Target vehicles supported by the ARM debuggers

| Target vehicle                                 | RVDS v2.2 support                   |                           |          | ADS v1.2.1 support |
|------------------------------------------------|-------------------------------------|---------------------------|----------|--------------------|
|                                                | RealView Debugger v1.8 <sup>a</sup> | AXD v1.3.1 (Windows only) | armsd    | AXD v1.3           |
| RealView Trace <sup>b</sup>                    | Yes                                 |                           |          |                    |
| RealView ICE                                   | Yes                                 |                           |          |                    |
| RealView ARMulator ISS                         | Yes <sup>c</sup>                    | RDI only                  | RDI only | RDI only           |
| RealMonitor <sup>d</sup>                       | Yes                                 | Yes                       |          | Yes                |
| MultiTrace <sup>tm</sup>                       | Yes                                 | Yes                       |          |                    |
| Multi-ICE                                      | Yes <sup>e</sup>                    | Yes                       |          | Yes                |
| Agilent Debug Interface (Gateway) <sup>f</sup> | Yes <sup>e</sup>                    | Yes                       |          | Yes                |
| Angel debug monitor (Remote_A)                 | Yes <sup>e</sup>                    | Yes                       | Yes      | Yes                |

- a. RDI connections are not supported on Sun Solaris and Red Hat Linux.
- b. The RealView Trace software is automatically installed with RealView Debugger.
- c. Only the RealView Connection Broker (localhost) connection is supported on Sun Solaris and Red Hat Linux.
- d. You can now use RealMonitor with RealView ICE in RealView Debugger.
- e. This target is not supported on Sun Solaris and Red Hat Linux.
- f. Agilent probe support is available in RVDS as a custom installation option.

## Chapter 2

# Differences

This chapter summarizes the main differences between *RealView® Developer Suite* (RVDS) v2.2, RVDS v2.1, RVDS v2.0, and *ARM Developer Suite™* (ADS) v1.2.1. The changes are described in:

- *Changes between RVDS v2.2 and RVDS v2.1* on page 2-2
- *Changes between RVDS v2.1 and RVDS v2.0* on page 2-4
- *Changes between RVDS v2.2 and ADS v1.2.1* on page 2-6.

## 2.1 Changes between RVDS v2.2 and RVDS v2.1

This section describes the changes between RVDS v2.2 and RVDS v2.1. It contains the following sections:

- *IDE support*
- *Debugger tool support*
- *Build tool support* on page 2-3
- *Agilent Probe support* on page 2-3.

### 2.1.1 IDE support

The CodeWarrior IDE is now provided to replace the RealView Debugger IDE. The CodeWarrior IDE in RVDS v2.2 is based on Metrowerks CodeWarrior v5.6.

---

**Note**

In RVDS v2.2, CodeWarrior for RVDS is supported on Windows XP and Windows 2000 systems only, and is not supplied for Sun Solaris and Red Hat Linux.

---

See *CodeWarrior for RVDS* on page 1-3 for more information on CodeWarrior for RVDS.

---

**Note**

If you are a user of CodeWarrior for ADS, see *CodeWarrior IDE changes* on page 2-6.

---

### 2.1.2 Debugger tool support

The main differences between the debugging tools in RVDS v2.2 and RVDS v2.1 are with RealView Debugger, which has:

- an improved menu structure
- an improved pane handling mechanism
- improved data navigation with the new Data Navigator pane
- internationalization support
- improved source code coloring
- trace, analysis, and profiling enhancements
- enhanced RTOS support
- support for gcc built images
- additional CLI commands, PRINTDSM and TRACEEXTCOND.

Also, support for standalone editors and the Vi editing mode has been removed from RealView Debugger.

For a detailed list of changes, see the *RealView Debugger Essentials Guide*.

### 2.1.3 Build tool support

The main differences between the build tools in RVDS v2.2 and RVDS v2.1 are:

- 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™.
- Available in RVCT v2.2, the new Thumb®-2 instruction set introduces many new 32-bit instructions, and some new 16-bit instructions.  
The Thumb-2 instruction set includes older 16-bit Thumb instructions as a subset.
- RVCT v2.2 is fully compliant with the *Base Platform ABI for the ARM Architecture* [BPABI] (unpublished DRAFT).
- RVCT v2.2 provides initial support for DWARF3 (Draft Standard 9) debug tables, as described in the *ABI for the ARM Architecture (base standard)* [BSABI].
- The command-line option `-g` switches on the generation of debug tables for the current compilation. Optimization options are specified by `-O`. By default, using the `-g` option does not affect the optimization setting.  
This is a change in behavior for RVCT v2.2.
- RVCT v2.2 supports the command-line option `--apcs /fpic` to compile code that is compatible with System V shared libraries.
- 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 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.
- RVCT v2.2 supports building of Linux applications and shared libraries.

For a detailed list of changes, see the *RealView Compilation Tools Essentials Guide*.

### 2.1.4 Agilent Probe support

Agilent Probe support is now available as a custom installation option in RVDS v2.2.

## 2.2 Changes between RVDS v2.1 and RVDS v2.0

This section describes the changes between RVDS v2.1 and RVDS v2.0. It contains the following sections:

- *Debugger tool support*
- *Build tool support.*

### 2.2.1 Debugger tool support

The main differences between the debugging tools in RVDS v2.1 and RVDS v2.0 are:

- *ARM® eXtended Debugger (AXD)* is included (see *ARM eXtended Debugger* on page 1-7)
- *ARM Symbolic Debugger (armsd)* is included (see *ARM Symbolic Debugger* on page 1-8)
- RealView Debugger has:
  - trace and profiling enhancements
  - enhanced RTOS support
  - new toolbar buttons and menu changes that mean you now have quick access to commonly used features.

### 2.2.2 Build tool support

The main differences between the build tools in RVDS v2.1 and RVDS v2.0 are:

- Increased compliance with the *Application Binary Interface for the ARM Architecture (Base Standard)* (ABI for the ARM Architecture (Base Standard)). See the ABI for the ARM Architecture page at <http://www.arm.com/>.
- C++ exception handling is supported. Therefore, with the exception of export templates, the remainder of ISO C++ is supported as defined by the *ISO/IEC 14822 :1998 International Standard for C++*.
- More optimization features are included, such as multifile compilation and linker feedback.
- Compression of read/write data areas is provided, to further reduce the image size.
- Some GNU C and C++ extensions are supported.
- Many new command-line options have been added to the build tools.

- The single-dash keyword and some command-line options are deprecated.

---

**Note**

---

The tools are now stricter in preserving eight-byte alignment. The compiler generates code with PRESERVE8 and REQUIRE8. The linker checks that code that requires eight-byte alignment only calls code that preserves eight-byte alignment. Therefore, this has implications for your legacy assembler code, object files and libraries. You must check that your existing assembly files, object files, or libraries preserve eight-byte alignment and correct them if required. For more details, see the *RealView Compilation Tools Assembler Guide* and *RealView Compilation Tools Linker and Utilities Guide* for more details.

---

## 2.3 Changes between RVDS v2.2 and ADS v1.2.1

This section describes the changes between RealView Developer Suite v2.2 and ADS v1.2.1. It contains the following sections:

- *CodeWarrior IDE changes*
- *Debugger changes* on page 2-7
- *Build tool changes* on page 2-7
- *ARM simulator changes* on page 2-9.

### 2.3.1 CodeWarrior IDE changes

The changes between CodeWarrior for RVDS and CodeWarrior for ADS are:

- CodeWarrior for ADS was based on CodeWarrior v4.2. CodeWarrior for RVDS is now based on CodeWarrior v5.6.
- The ARM tool-specific configuration panels are tailored to RVDS v2.2.
- The separate ARM compilers are combined into a single compiler in RVDS v2.2, therefore there is only one compiler configuration panel in RVDS v2.2.
- You can now run and debug your image with RealView Debugger, in addition to AXD and armsd.
- You can now concatenate libraries.
- You can now import CodeWarrior for ADS projects into CodeWarrior for RVDS.
- The default ARM stationery in CodeWarrior for RVDS does not include a DebugRel build target. However, a DebugRel build target is created if you import a CodeWarrior for ADS project, to preserve any settings you might have configured for that build target.
- Unlike the ADS compiler, the RVCT compiler does not generate browser information. This functionality is now provided by the builtin language parser of CodeWarrior.
- Code formatting.
- Code completion, including code completion for C++ template classes.
- Go to next/previous function.
- Word wrap when printing.
- Support for source-relative #includes.

- Find inside/outside of comments.
- Improved language parser speed and feedback.
- New editor bindings.
- Ability to show and hide the Code and Data columns in the project window.
- Support for workspaces.
- Object writing.
- Form editor changes.
- Smart variable formatting.

---

**Note**

---

All target connection and debugging features in the CodeWarrior IDE are not available in CodeWarrior for RVDS. You must run one of the ARM debuggers to perform these functions.

---

### 2.3.2 Debugger changes

The main differences between the debugging tools in RVDS v2.2 and ADS v1.2.1 are:

- RealView Debugger is the latest ARM debugger, which enables you to perform advanced debugging functions such as:
  - multiprocessor debugging
  - OS-aware debugging
  - extended target visibility
  - trace, analysis, and profiling
  - access to the RealView ICE JTAG control unit over Ethernet and USB.
- AXD is enhanced to be able to debug C and C++ programs built with the new RealView Compilation Tools provided with RVDS v2.2.

### 2.3.3 Build tool changes

The main differences between the build tools in RVDS v2.2 and ADS v1.2.1 are as follows:

- Compliance with the new ABI for the ARM Architecture (Base Standard). See the ABI for the ARM Architecture page at <http://www.arm.com/>. This is different to the old ADS ABI. Some compatibility is provided with the `--apcs /adsabi` command line option.

- 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 exceptions, namespaces, templates, and intelligent implementation of *Run-Time Type Information* (RTTI), but excludes the export of templates.
- Support for some GNU language extensions.
- ARM and Thumb® compilation on a per-function basis.
- Re-engineered inline assembler, and a new embedded assembler that enables you to include out-of-line assembly code.
- Linker feedback to remove unused functions.
- Full support for ARM architecture v6 instructions has been added.
- Read/write data compression enables the optimization of ROM size.
- Removal of unused C++ virtual functions.
- Multifile compilation, which performs optimizations across multiple compilation units.
- You can specify a library search path, to indicate where to search for your user libraries.
- You can separate RO code and data into different execution regions.
- There are new scatter-loading attributes.
- Unicode and multibyte characters are supported.
- Compiler intrinsics are available to access the return address of a function, the current stack pointer value, and the current program counter value. An additional intrinsic enables you to insert the BKPT instruction in your C or C++ code.
- You can identify a function that does not return, so that the compiler generates more efficient code.
- The C++ name mangling scheme has changed.
- The *ARM Profiler* (armprof) is not provided with RVCT.
- The ARM Applications Library is not provided with RVCT.
- Unlike the ADS compiler, the RVCT compiler does not generate browser information.
- There are changes to the assembler, compiler, and linker command-line options.

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.

---

- The fromelf option -ihf has been removed.

---

**Note**

---

The tools are now stricter in preserving eight-byte alignment. The compiler generates code with PRESERVE8 and REQUIRE8. The linker checks that code that requires eight-byte alignment only calls code that preserves eight-byte alignment. Therefore, this has implications for your legacy assembler code, object files and libraries. You must check that your existing assembly files, object files, or libraries preserve eight-byte alignment and correct them if required. For more details, see the *RealView Compilation Tools Assembler Guide* and *RealView Compilation Tools Linker and Utilities Guide* for more details.

---

### 2.3.4 ARM simulator changes

RealView ARMulator® ISS is the latest version of the ARM simulator. It supports connections through RealView Connection Broker and RDI. When connecting to the simulator through RealView Connection Broker under RealView Debugger, you can have multiple connections to the simulator. You can connect to the RDI interface of RealView ARMulator ISS using RealView Debugger, AXD v1.3, and armsd.

---

**Note**

---

Although you can install ADS in addition to RealView Developer Suite v2.2, you must exercise caution if you use both RealView ARMulator ISS and ADS ARMulator. See the *RealView Developer Suite v2.2 Release Notes* for more details.

---



# Chapter 3

## Getting Started with RealView Developer Suite

The component products provided with RealView® Developer Suite (RVDS) enable you to build and debug one or more images that make up your application. This chapter introduces you to the basic tasks for building and debugging with the RVDS tools. It contains the following sections:

- *Building and debugging task overview* on page 3-2
- *Using the example projects* on page 3-5.

### 3.1 Building and debugging task overview

Table 3-1 is a high-level procedure showing the main tasks for building and debugging applications with the RVDS tools, and where to find the details.

The tasks referred to in the referenced documentation are not necessarily described in the order presented in Table 3-1. If you are using the RVDS tools for the first time, it is suggested that you work through the tasks in the order described in the referenced documents. The sequence presented in Table 3-1 reflects the order in which the tasks might usually be performed.

**Table 3-1 Main building and debugging tasks**

| Step | Description                                                                                                                                                                                                                                                                      | Reference                                                                                            |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| 1    | Choose the RVDS application you want to use to manage and build your projects: <ul style="list-style-type: none"> <li>if you want to use CodeWarrior for RVDS, continue at step 3</li> <li>if you want to build from the command line using RVCT, continue at step 2.</li> </ul> |                                                                                                      |
| 2    | If you want to use the RVCT build tools directly, then create makefiles or Windows command files containing the required build commands.<br>Continue at step 9 to load and debug your image in RealView Debugger.                                                                | <i>RealView Compilation Tools Essentials Guide</i>                                                   |
| 3    | Start CodeWarrior for RVDS.                                                                                                                                                                                                                                                      | <i>Starting CodeWarrior for RVDS</i> on page 4-3                                                     |
| 4    | If a CodeWarrior for RVDS project already exists, continue at step 6.<br><br>Otherwise, create a CodeWarrior for RVDS project for your application.                                                                                                                              | <i>Working with an existing project</i> on page 4-5<br><br><i>Creating a new project</i> on page 4-3 |
| 5    | Set up the build target settings as required to build the image for your application. Continue at step 7.                                                                                                                                                                        | <i>Predefined build targets</i> on page 4-5                                                          |
| 6    | Open the existing CodeWarrior for RVDS project.                                                                                                                                                                                                                                  | <i>Working with an existing project</i> on page 4-5                                                  |
| 7    | Build the image for the CodeWarrior for RVDS project.                                                                                                                                                                                                                            | <i>Building the image for your project</i> on page 4-6                                               |

**Table 3-1 Main building and debugging tasks (continued)**

| Step | Description                                                                                                                                                                                                                          | Reference                                                                                                                                                                                                                                                                                |
|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 8    | Decide what image to use: <ul style="list-style-type: none"> <li>If you want to use an existing image, such as a prebuilt example image, continue at step 9.</li> <li>If you want to build a new image, return to step 1.</li> </ul> | <i>Using the example projects on page 3-5</i>                                                                                                                                                                                                                                            |
| 9    | Start RealView Debugger.                                                                                                                                                                                                             | <i>RealView Debugger Essentials Guide</i>                                                                                                                                                                                                                                                |
| 10   | Configure your debug target and connections as required.                                                                                                                                                                             | <i>RealView Debugger Target Configuration Guide</i>                                                                                                                                                                                                                                      |
| 11   | Connect to your debug target.                                                                                                                                                                                                        | <ul style="list-style-type: none"> <li><i>RealView Debugger Essentials Guide</i></li> <li><i>RealView Debugger Target Configuration Guide.</i></li> </ul>                                                                                                                                |
| 12   | Load the image ready for debugging.                                                                                                                                                                                                  | <ul style="list-style-type: none"> <li><i>RealView Debugger Essentials Guide</i></li> <li><i>RealView Debugger User Guide.</i></li> </ul>                                                                                                                                                |
| 13   | Prepare any debugging facilities, such as breakpoints and tracepoints.                                                                                                                                                               | <ul style="list-style-type: none"> <li><i>RealView Debugger Essentials Guide</i></li> <li><i>RealView Debugger User Guide</i></li> <li><i>RealView Debugger Extensions User Guide.</i></li> </ul>                                                                                        |
| 14   | Run the image.                                                                                                                                                                                                                       | <ul style="list-style-type: none"> <li><i>RealView Debugger Essentials Guide</i></li> <li><i>RealView Debugger User Guide.</i></li> </ul>                                                                                                                                                |
| 15   | Perform the required debugging and monitoring tasks, such as stepping, and displaying contents of variables and memory. If using tracepoints, use the trace analysis facilities of RealView Debugger to analyze the trace output.    | <ul style="list-style-type: none"> <li><i>RealView Debugger Essentials Guide</i></li> <li><i>RealView Debugger User Guide</i></li> <li><i>RealView Debugger Extensions User Guide.</i></li> </ul>                                                                                        |
| 16   | What is the result of the debugging session? <ul style="list-style-type: none"> <li>If there are problems, continue at step 17.</li> <li>If there are no problems, rebuild your image for final release.</li> </ul>                  | <ul style="list-style-type: none"> <li><i>Building the image for your project on page 4-6</i></li> <li><i>RealView Debugger Essentials Guide</i></li> <li><i>RealView Debugger Project Management User Guide</i></li> <li><i>RealView Compilation Tools Essentials Guide.</i></li> </ul> |
| 17   | Decide how to fix any problems in your source code:                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                          |

Table 3-1 Main building and debugging tasks (continued)

| Step | Description                                                                                                                                                                                                                                                          | Reference                                                                                                                                                 |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
|      | <ul style="list-style-type: none"><li>• use CodeWarrior for RVDS</li></ul>                                                                                                                                                                                           | <ul style="list-style-type: none"><li>• Chapter 4 <i>Getting Started with CodeWarrior for RVDS</i></li><li>• <i>CodeWarrior IDE User Guide</i>.</li></ul> |
|      | <ul style="list-style-type: none"><li>• use another source editor of your choice.</li></ul>                                                                                                                                                                          |                                                                                                                                                           |
| 18   | When you have fixed the problem, then you must rebuild, reload, and debug the image: <ul style="list-style-type: none"><li>• if you are using CodeWarrior for RVDS, then return to step 6</li><li>• if you are using RVCT directly, then return to step 2.</li></ul> |                                                                                                                                                           |

## 3.2 Using the example projects

Although your aim is to build and debug your own application images, the tasks described in the CodeWarrior for RVDS, RealView Debugger, and RVCT documentation use the Dhrystone project and some of the other example projects provided with RVDS (see *RVDS example projects* on page 1-7).

Until you are familiar with the features of CodeWarrior for RVDS, RealView Debugger, and RVCT, it is suggested that you follow the instructions as described. However, many tasks described in the user documentation require that you modify the files in the examples. Before you do this, make a backup copy of the example project files and directories.

### 3.2.1 CodeWarrior for RVDS example project files

Some projects in the RVDS examples directory contain CodeWarrior for RVDS project files (.mcp).

For more details on using the existing CodeWarrior for RVDS project files, see *Working with an existing project* on page 4-5.

### 3.2.2 The example Dhrystone project

The Dhrystone project is in the RealView Developer Suite examples directory `install_directory\RVDS\Examples\...\dhrystone`.

This contains:

- a predefined RealView Debugger project file
- a precompiled image
- a makefile and a Windows command file to enable you to rebuild the image.

This example project is used in the tasks described in the *RealView Debugger Essentials Guide* and the *RealView Compilation Tools Essentials Guide*.



# Chapter 4

## Getting Started with CodeWarrior for RVDS

This chapter provides a tutorial on how to start using CodeWarrior for RealView® Developer Suite (RVDS). It contains:

- *About CodeWarrior for RVDS* on page 4-2
- *Starting to use CodeWarrior for RVDS* on page 4-3
- *Working with an existing project* on page 4-5
- *Overview of the ARM project stationery supplied with RVDS* on page 4-10
- *Summary of the build target configuration panels* on page 4-14
- *Using the Thumb ARM Interworking Image stationery* on page 4-27
- *Creating binary ROM images* on page 4-33
- *Creating a project that builds an object library and ROM images* on page 4-34
- *Creating a subproject within another project* on page 4-41
- *Running batch files with the batch runner* on page 4-49
- *Creating your own project stationery* on page 4-53
- *Upgrading CodeWarrior for ADS projects* on page 4-56
- *Creating projects from XML formatted files* on page 4-59
- *Importing makefiles* on page 4-60
- *Exporting CodeWarrior for RVDS projects in XML format* on page 4-63
- *Using CodeWarrior for RVDS from the command line* on page 4-64.

## 4.1 About CodeWarrior for RVDS

CodeWarrior for RVDS is based on Metrowerks CodeWarrior IDE version 5.6. It has been tailored to support the RealView Compilation Tools (RVCT) provided with RVDS. It provides:

- ARM® tool-specific configuration panels that enable you to configure the RVCT tools from within CodeWarrior for RVDS (see *Summary of the build target configuration panels* on page 4-14)
- ARM project stationery targeted at the RVCT tools that enables you to create basic ARM and Thumb® projects from CodeWarrior for RVDS (see *Overview of the ARM project stationery supplied with RVDS* on page 4-10).

Although most of the ARM toolchain is tightly integrated with CodeWarrior for RVDS, the ARM debuggers are not tightly integrated. This means, for example, that you cannot set breakpoints or watchpoints from within CodeWarrior for RVDS. Instead you must use one of the following ARM debuggers supplied with RVDS:

- RealView Debugger
- ARM eXtended Debugger (AXD)
- ARM Symbolic Debugger (armsd).

---

**Note**

AXD and armsd are supported for ARM7 and ARM9 targets only.

---

See the documentation for your debugger for more information on the debugging features provided by that debugger.

### 4.1.1 RVCT22BIN environment variable

CodeWarrior for RVDS uses the RVCT22BIN environment variable to locate the RVCT build tools, not the PATH environment variable. This is set up for you by the RVDS installer.

## 4.2 Starting to use CodeWarrior for RVDS

This section describes how to start CodeWarrior for RVDS, and how to begin creating a new project. It includes:

- *Starting CodeWarrior for RVDS*
- *Creating a new project.*

### 4.2.1 Starting CodeWarrior for RVDS

To start CodeWarrior for RVDS, select the following from the Windows **Start** menu:

**Programs → ARM → RealView Developer Suite v2.2 → CodeWarrior for RVDS**

### 4.2.2 Creating a new project

To create a new project, select **File → New** from the CodeWarrior main menu. This displays the New dialog box shown in Figure 4-1.

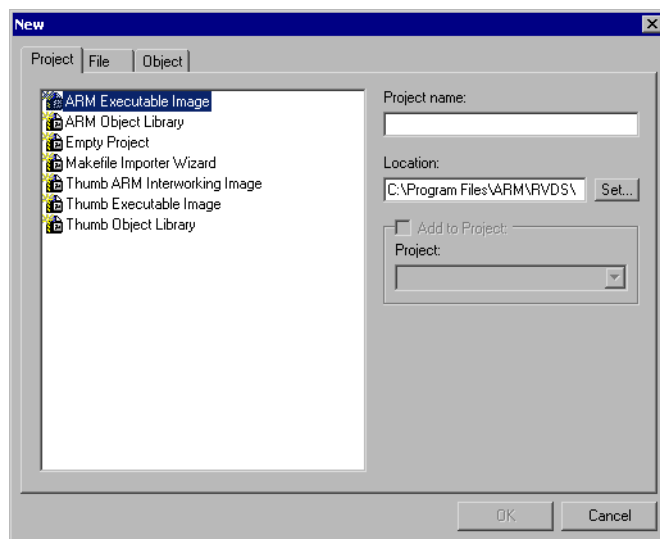


**Figure 4-1** New dialog box

You can:

- Create a project using the ARM stationery supplied with RVDS (see *Overview of the ARM project stationery supplied with RVDS* on page 4-10). This is the recommended method if you are a new user.
- Create a project from an existing makefile using the Makefile Importer Wizard.

- Create an empty project, and set up the project yourself.  
Be aware that before you can assign source files to your empty project, you must have file mappings set up. To do this:
  1. Select the Target Settings panel.
  2. Change the Linker to **ARM RealView Linker**.  
CodeWarrior automatically creates file mappings for your project. See *Target panels* on page 4-15 for more details on the File Mappings panel.

You can also create a new CodeWarrior project from a .xml file (see *Creating projects from XML formatted files* on page 4-59).

For examples of how to create a project using the ARM stationery supplied with RVDS, see:

- *Using the Thumb ARM Interworking Image stationery* on page 4-27
- *Creating a project that builds an object library and ROM images* on page 4-34.

For details on how to create a project from an empty project, or by using the Makefile Importer Wizard, see the *CodeWarrior User Guide*.

## 4.3 Working with an existing project

This section describes how to work with an existing CodeWarrior project. It includes the following sections:

- *Predefined build targets*
- *Opening an existing CodeWarrior project*
- *Building the image for your project* on page 4-6
- *Running the image in the Debugger* on page 4-6.

### 4.3.1 Predefined build targets

The RVDS example project files define the following build targets:

- |                |                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Debug</b>   | This build target is configured to build output binaries that give the best debug view, at the expense of optimization. This build target provides the best debug view while you are developing your code. It is also configured to output basic image information in an error and messages window.<br><br>This is the default build target for new projects created with the ARM project stationery. |
| <b>Release</b> | This build target is configured to build output binaries that are fully optimized, at the expense of a poor debug view. This build target outputs optimized code suitable for release.                                                                                                                                                                                                                |

### 4.3.2 Opening an existing CodeWarrior project

To open an existing CodeWarrior project file:

1. Select **File** → **Open...** from the main menu to display the Open dialog box.
2. Locate the project file for the project you want to open.  
In this example, locate the project file for the Dhrystone project:  
`install_directory\RVDS\Examples\...\dhrystone\dhrystone.mcp`
3. Click **Open** to open the project.

If no build subdirectories exist for the project, CodeWarrior creates them under the project directory. For this example, `dhrystone_Data`. This subdirectory contains a directory for each build target, for example, `Debug` and `Release`.

### 4.3.3 Building the image for your project

To build the image for your CodeWarrior project, select **Project** → **Make** from the main menu.

When you build an image for a particular build target, CodeWarrior for RVDS places the image in the subdirectory for that build target. For example, if you build the RVDS example `dhystone.axf` image for the Debug build target, it is placed in the directory:

```
install_directory\RVDS\Examples\...\dhystone\dhystone_Data\Debug
```

### 4.3.4 Running the image in the Debugger

To run the image for your project:

1. Select the debugger to use for the project (see *Selecting a debugger for the project*).
2. Prepare the connection between your debugger and the target (see *Preparing the debugger connection* on page 4-7).
3. Run the project image (see *Running the project image* on page 4-8)

#### Selecting a debugger for the project

You can choose the debugger that your project is to use by default. This can be one of the following:

- RealView Debugger (this is the default for new projects, and for the example RVDS projects)
- AXD
- armsd.

---

#### **Note**

AXD and armsd are supported for legacy ARM7 and ARM9 targets only.

---

You can set up the required debugger on the following panels (see *Debugger panels* on page 4-25 for details):

- **RealView Debugger**
- **RealView Runner.**

To select a debugger for your project:

1. Select **Edit** → **TargetName Settings...** from the main menu to display the settings panels for the current build target, where **TargetName** is the current build target.

2. Select **RealView Debugger** in the Debugger panels of the Target Settings Panels list.
3. In the **Choose Debugger** tab, select the debugger you want to use. The default is **RVD (RealView Debugger)**.
4. Select **RealView Runner** in the Debugger panels of the Target Settings Panels list.
5. In the **Choose Debugger** tab, select the debugger you want to use. The default is **RVD (RealView Debugger)**.

---

**Note**

---

For RealView Debugger, there is no difference between starting it from the **RealView Debugger** panel or from the **RealView Runner** panel.

---

### Preparing the debugger connection

Before you can run your image in the debugger, you must set up a connection between the debugger and the required target. How you do this depends on the debugger you are using, see:

- *Preparing RealView Debugger*
- *Preparing the ARM eXtended Debugger* on page 4-8
- *Preparing the ARM Symbolic Debugger* on page 4-8.

#### ***Preparing RealView Debugger***

To prepare a connection in RealView Debugger:

1. Start RealView Debugger:  
**Start → Programs → ARM → RealView Developer Suite v2.2 → RealView Debugger v1.8**
2. Set up the connection to your debug target.  
 For details on setting up a connection to a debug target in RealView Debugger, see *RealView Debugger Essentials Guide* and the *RealView Debugger Target Configuration Guide*.
3. Exit RealView Debugger.

When you next start RealView Debugger from CodeWarrior, the connection is established and your image is loaded automatically.

### ***Preparing the ARM eXtended Debugger***

To prepare a connection in AXD, for example:

1. Start AXD:  
**Start → Programs → ARM → RealView Developer Suite v2.2 → AXD Debugger v1.3.1**
2. Set up the connection to your debug target. The default is ARM7TDMI.  
For details on setting up a connection to a debug target in AXD, see *RealView Developer Suite AXD and armsd Debuggers Guide*.
3. Exit AXD.

When you next start AXD from CodeWarrior, the connection is established and your image is loaded automatically.

### ***Preparing the ARM Symbolic Debugger***

To prepare a connection for armsd, do the following from the **Choose Debugger** tab (see *Selecting a debugger for the project* on page 4-6):

1. Select **armsd (ARM Symbolic Debugger)**.
2. Select the **Armsd1** tab to display the Armsd1 panel.
3. Select the required Debug Target.
4. Select the required Target Processor.
5. Click **OK** to save the settings and close the settings panel.

### **Running the project image**

To run the image in the chosen debugger:

1. Prepare the debugger connection if you have not done this already (see *Preparing the debugger connection* on page 4-7).
2. Run the debugger using one of the following options:
  - Select **Project → Run** from the main menu, to run the debugger you set up in the **RealView Runner** panel.
  - Select **Project → Debug** from the main menu, to run the debugger you set up in the **RealView Debugger** panel.

The debugger starts, establishes the connection you have set up, and loads the project image to the target. Run and debug the image using the features provided with the debugger. For details on how to use the debugging features for the chosen debugger, see:

- the RealView Debugger documentation set
- the *RealView Developer Suite AXD and armsd Debuggers Guide*.

## 4.4 Overview of the ARM project stationery supplied with RVDS

This section describes the ARM project stationery that is supplied with RVDS. It includes:

- *What is project stationery?*
- *The project stationery folder*
- *Project stationery supplied with RVDS.*

### 4.4.1 What is project stationery?

Project stationery is typically a minimal, preconfigured template project file. You can use project stationery to create a new project quickly. When you create a new project or open a project stationery file, CodeWarrior for RVDS creates a new project and, optionally, a new folder for the project. It then copies all the files related to the stationery project to the new folder.

Project stationery can include:

- preconfigured build target settings for the project
- predefined build targets, subprojects, and build dependencies
- all files included in the stationery project.

### 4.4.2 The project stationery folder

The project stationery folder is located in the RVDS installation folder:

`install_directory\IDEs\CodeWarrior\CodeWarrior\...\Stationery`

The project stationery files supplied with RVDS are for common types of projects, and they are located in subdirectories in the project stationery directory.

You can create your own project stationery by saving preconfigured projects, together with their support files, in the project stationery directory. See *Creating your own project stationery* on page 4-53 for more information.

### 4.4.3 Project stationery supplied with RVDS

CodeWarrior for RVDS includes default project stationery to enable you to create an ARM, Thumb, or Thumb ARM interworking executable quickly and easily.

All the supplied project stationery templates use:

- default target settings (little-endian ARM7TDMI®)
- default *Procedure Call Standard for the ARM Architecture* (AAPCS) options for the compiler and assembler

- separate build targets for debug and release options (see *Predefined build targets in ARM project stationery* on page 4-13).

This means that you must reconfigure the target options if, for example, you want to target a different ARM processor.

The following project stationery is supplied with CodeWarrior for RVDS:

### ARM Executable Image

Use this stationery to build an executable ELF image from ARM code. It is configured to use:

- The compiler to compile all C or C++ files in ARM state. If you want to compile C code in C++ mode, you must set the Source Language to **ISO Standard C++ (1998)** in the **Source** tab of the RealView Compiler configuration panel.
- The assembler to assemble all files with a .s filename extension.
- The linker to link a simple executable ELF image.
- RealView Debugger to both debug and run executable images output by the project.

### Thumb Executable Image

Use this stationery to build an executable ELF image from Thumb code. It is configured to use:

- The compiler to compile all Thumb C or C++ files in Thumb state. If you want to compile Thumb C code in Thumb C++ mode, you must set the Source Language to **ISO Standard C++ (1998)** in the **Source** tab of the RealView Compiler configuration panel.
- The assembler to assemble all files with a .s filename extension. By default, the Thumb Executable Image stationery configures the assembler to assemble for Thumb. See the *RealView Compilation Tools Assembler Guide* for information on switching the assembler to compile ARM code.
- The linker to link a simple executable ELF image.
- RealView Debugger to both debug and run executable images output by the project.

### ARM Object Library

Use this stationery to build a library of ARM objects. The library contains ELF object format members. This stationery is similar to the ARM Executable Image project template. The major differences are:

- it is configured to use the armar utility to output an object library.

- you cannot debug or run a standalone library file until it is linked into an image.

### **Thumb Object Library**

Use this stationery to build a library of Thumb objects. The library contains ELF object format members. This stationery is similar to the Thumb Executable Image stationery. The major differences are:

- it is configured to use the armar utility to output an object library
- you cannot debug or run a standalone library file until it is linked into an image.

### **Thumb ARM Interworking Image**

Use this stationery to build an executable ELF image from interworking ARM and Thumb code. It is configured to use:

- Separate build targets for ARM code and Thumb code. The output from the Thumb build targets is chained with the corresponding ARM build targets. See *Assign the build target dependencies* on page 4-38 for information on defining build dependencies between build targets.
- The compiler to compile C and C++ code included in the ARM build targets.
- The compiler to compile Thumb C and C++ code included in the Thumb build targets.
- The assembler to assemble both ARM and Thumb assembly language source.
- The linker to link the output from the Thumb and ARM build targets into an executable ELF image.
- The AAPCS interworking option for the assembler and compiler.

## Predefined build targets in ARM project stationery

The Release and Debug build targets are defined in the following ARM stationery:

- ARM Executable Image
- ARM Object Library
- Thumb Executable Image
- Thumb Object Library.

These use Debug as the default build target.

The Thumb ARM Interworking Image stationery uses the ThumbRelease, ThumbDebug, ARMRelease, and ARMDebug build targets. The ARMRelease build target is a dependency of ThumbRelease, and the ARMDebug build target is a dependency of ThumbDebug. The default build target for this stationery is ThumbDebug.

See *Predefined build targets* on page 4-5 for details on differences between the Release and Debug build targets.

## 4.5 Summary of the build target configuration panels

This section gives a summary of the build target configuration panels in CodeWarrior for RVDS. It includes:

- *Displaying the build target configuration panels*
- *Getting help on the build target configuration panels*
- *Factory settings*
- *Target panels* on page 4-15
- *Language Settings panels* on page 4-23
- *Linker panels* on page 4-24
- *Editor panels* on page 4-25
- *Debugger panels* on page 4-25
- *Miscellaneous panels* on page 4-26.

### 4.5.1 Displaying the build target configuration panels

To display the configuration panels for a build target:

1. Open an existing project (see *Working with an existing project* on page 4-5), or create a new project (see *Overview of the ARM project stationery supplied with RVDS* on page 4-10).
2. Set the build target to be the current build target.
3. Select **Edit** → **TargetName Settings...** from the main menu.

Alternatively, select the **Targets** tab, and double-click on the required build target.

### 4.5.2 Getting help on the build target configuration panels

To get help on the build target configuration panels, you can:

-  • Use the help button on the build target settings title bar to get help on an individual element of a panel.
-  • Use the help button on the panels that are specific to the ARM tools for help on the whole panel.

### 4.5.3 Factory settings

You can use the **Factory Settings...** button on a panel to reset the configuration to that panel back to a base configuration. However, reverting to the factory settings on the panels specific to the ARM tools does not necessarily revert the settings back to the defaults for the ARM stationery.

#### 4.5.4 Target panels

The following Target panels are provided:

##### Target Settings

This panel enables you to set the following:

##### Target Name

The name of the build target. This is the name you specified when you created the build target. You can change this name if required.

##### ———— Note ————

If you have also specified an Output Directory with the same name as the build target, then make sure you change the Output Directory accordingly.

##### Linker

Specify the linker to use.  
If you want the build target to create an executable image or an object library, then make sure that the Output Type in the RealView Target panel is set to **Linker Output**.

This setting can be one of the following:

**None** No link stage is required for the target. When you select this, the Linker panels are removed from the Target Settings Panels list (see *Linker panels* on page 4-24).

##### ARM RealView Linker

To create an executable image. When you select this option, the RealView Linker panel entry appears in the Linker panels list (see *Linker panels* on page 4-24).

This is the default for the ARM stationery.

##### ARM RealView Librarian

To create an ARM library. When you select this, the RealView Linker panel entry is removed from the Linker panels list (see *Linker panels* on page 4-24).

##### Pre-linker

Not supported in CodeWarrior for RVDS.

## Post-linker

Specify a postlinker.

**None** No post-link stage is required for the target. When you select this, the RealView FromELF panel entry is removed from the Linker panels list (see *Linker panels* on page 4-24). Use this option, for example, if you want to create object files only.

## ARM RealView FromELF

To convert an executable image to binary format, or to generate text output. When you select this option, the RealView FromELF panel entry appears in the Linker panels list (see *Linker panels* on page 4-24).

To create a binary output, the build target must create an executable image. Therefore, make sure that:

- the Linker is set to **ARM RealView Linker** in the Target Settings panel
- the Output Type is set to **Linker Output** in the RealView Target panel.

This is the default for the ARM stationery.

## Batch File Runner

Use the Batch File Runner to run a DOS batch file as the final step in the link process. The batch file runner runs the first, and only the first, .bat file in the link order view of the project. See *Running batch files with the batch runner* on page 4-49 for detailed information on using the Batch File Runner.

When you select this option, the RealView FromELF panel entry is removed from the Linker panels list (see *Linker panels* on page 4-24).

## Output Directory

The directory where the output files are to be created. By default this is the main project directory.

If you specify a directory, the path is shown relative to the project directory. For example, {Project}Debug specifies the Debug directory below the project directory. If the directory specified here does not exist when you open the project, then it is created.

**Save project entries using relative paths**

This checkbox determines whether or not the project entries are to be located relative to the project paths. If you use relative paths, then you can easily move an entire project to another machine.

**Access Paths**

This panel enables you to specify paths to define directories that CodeWarrior for RVDS searches for libraries, header files, and source files:

**User Paths**

By default, the User path is set to the main location of your project. The ARM tools search these paths for:

- User header files. These are header files that you include with a `#include "..."` statement.
- User libraries. These are libraries that correspond to your `#include "..."` header files.
- Your source files. When you add a source file to your project from any directory, CodeWarrior for RVDS adds the path for the source file to the User Paths list automatically.

By default, the User Paths setting contains {Project}. This is the folder that contains the open project.

**System Paths**

When you create a project using one of the ARM stationery files, the System path {RVCT22INC} is included to point to the standard header files, and is taken from the corresponding environment variable. You do not have to include a System path for the RVCT22LIB environment variable.

**Always Search User Paths**

By default, CodeWarrior searches only System Paths for a file included with `#include <...>`. To have it search User Paths in addition to System Paths, click this checkbox.

`#include "..."` always searches both sets of paths.

**————— Note —————**

It is strongly recommended that you select this option for complex, or multi-user projects.

If this option is not selected, and any source file is found in a system search path, it is effectively promoted to an unchanging system file. This means that if a later version of the source file is placed in a user search path, it is not found by CodeWarrior for RVDS until this option is selected.

---

———— **Note** —————

When adding or changing a path, do not use the **Compiler Relative Path** Type on the Browse for Folder dialog box.

---

## **Build Extras**

This panel enables you to change the way a project builds, including:

- how project information is cached
- whether browser information is generated
- whether a third-party debugger is used.

Select values for the following options:

### **Use modification date caching**

This option is disabled by default in the ARM stationery.

Select this option:

- to instruct the CodeWarrior IDE not to check the modification dates of files you have changed outside the CodeWarrior IDE
- if you edit files with the CodeWarrior IDE editor only, and are working in a single user environment.

Selecting this option reduces compilation time. However, if you have modified any sources outside of CodeWarrior, then it is unable to detect the changes to those sources.

Deselect this option:

- if you have configured the CodeWarrior IDE to use a third-party editor
- you are working on a multi-user project with shared access to source files.

### **Cache Subprojects**

This option is disabled by default in the ARM stationery.

Select this option to:

- Improve multiproject updating and linking.

- Enable the CodeWarrior IDE browser to include browser information from target subprojects.

Deselecting this option reduces the amount of memory required by the CodeWarrior IDE.

### **Generate browser data from**

Select this option if you want the IDE to generate symbolics information the next time your project is built. Choose:

**None** If you do not want browser data to be generated.

**Compiler** This option is not supported.

### **Language parser**

If you want the data to be generated from the language parser. Without this information, you cannot open browser windows for your project.

The language parser:

- Updates browser data as you edit.
- Generates browser information describing types of global variables and class data (C/C++).
- Generates members and return types of global functions and class member functions, so that the code completion feature can list members accessed by . or -> on an object or function return type.
- Generates information for global function prototypes, for example from system headers accessible and used by the code completion feature. Currently they do not show up in browser contents window, although they are syntax-colored in the IDE editor
- Generates #include dependencies at parse-time. New #include statements are dynamically added so that you do not have to compile to generate new browser data.

### **Use External debugger**

Select this checkbox if you are using a third-party debugger instead of the ARM debuggers. You must specify the following options:

#### **Application**

Click **Browse...** to search for the non-ARM debugger application you want to use. The application you select is automatically placed in the Application field.

Arguments

If you want the selected debugger to use arguments whenever it is run from within the CodeWarrior IDE, enter them in this field. See the documentation accompanying the third-party debugger you are using for details on supported arguments.

Initial directory

Click **Browse...** to search for the working (initial) directory in which the debugger you have selected will run. The working directory should be the root directory of your project. This facilitates, for example, the process of searching for a file when you attempt to open it from the selected debugger.

Runtime Settings

Not supported in CodeWarrior for RVDS.

File Mappings

This panel enables you to specify which tools to use for the various source files that make up your project. You can add new file mappings and remove existing file mappings as required. Table 4-1 shows the default file mappings for the ARM stationery.

Table 4-1 File mappings

| Extension                 | File Type | Compiler               | Description            |
|---------------------------|-----------|------------------------|------------------------|
| .c or .ac                 | Text      | ARM RealView Compiler  | C source coded file    |
| .C++, .acpp, .cp, or .cpp | Text      | ARM RealView Compiler  | C++ source coded file  |
| .h                        | Text      | ARM RealView Compiler  | C header file          |
| .hpp                      | Text      | ARM RealView Compiler  | C++ header file        |
| .inc                      | Text      | ARM RealView Assembler | Assembler include file |
| .s                        | Text      | ARM RealView Assembler | Assembler source file  |
| .tc                       | Text      | ARM RealView Compiler  | Thumb C source file    |
| .tcpp                     | Text      | ARM RealView Compiler  | Thumb C++ source file  |
| .scat                     | Text      | None                   | Scatter-loading file   |
| .a or .lib                |           | ARM ELF Importer       | ARM library file       |

**Table 4-1 File mappings (continued)**

| <b>Extension</b> | <b>File Type</b> | <b>Compiler</b>  | <b>Description</b>                       |
|------------------|------------------|------------------|------------------------------------------|
| .axf             |                  | ARM ELF Importer | ARM image                                |
| .bat             | Text             | None             | DOS batch command file for post linker   |
| .o               |                  | ARM ELF Importer | Object source file                       |
| .sym             |                  | ARM ELF Importer | A file containing symbol definitions     |
| .txt             |                  | None             | Text file                                |
| .tdv             |                  | ARM ELF Importer | Intermediate build file for a subproject |

### Source Trees

This panel enables you to define target-specific source trees (root paths) for use in your projects. You can define your project access paths and build target output in terms of source trees.

When you create a project using one of the ARM stationery templates, the RVCT22INC environment variable is specified in this panel. You can also specify a location as an absolute path or from a registry key. You do not have to include the RVCT22LIB environment variable.

### RealView Target

Enables you to specify what the build target generates:

#### Output Name

The name of the output file that is to be generated by this build target. This name can include a path.

By default the output file name is the same as the project name. To specify a different name:

1. Uncheck the **Use project name** checkbox.
2. Enter the required name.
3. Click **Apply** to apply the change.

You might want to do this if the build target is to be a subtarget of another build target.

## Output Type

The type of output can be one of:

### Directory of objects

Only object files are produced. This is the usual setting if the build target is to be a subtarget of another build target. The subtarget generates the object files, and the parent build target links the object files to create an executable image.

For example, the ARM subtarget of the Thumb ARM Interworking Image stationery (see *Using the Thumb ARM Interworking Image stationery* on page 4-27).

### Linker Output

The output depends on the Linker setting in the Target Settings panel:

- if the Linker is set to **ARM RealView Linker**, then the output from this build target is either an executable image or a partially-linked object
- if the Linker is set to **ARM RealView Librarian**, then the output from this build target is an ARM object library.

### Echo ARM tool command-lines verbosely

To echo the build commands to the Errors & Warnings dialog box, select this checkbox.

By default build commands are not echoed to the Errors & Warnings dialog box when you build a target. In this case, the dialog box is displayed only if any errors or warning messages are issued, or if you have specified that remarks are to be displayed.

### Show object sizes

To show the code and data sizes for the source files in the **Files** tab and the **Link Order** tab of the project window, select this checkbox.

This setting is disabled by default.

#### ———— Note ————

Enabling this option might slow down project builds.

### 4.5.5 Language Settings panels

The following Language Settings panels are provided:

#### RealView Assembler

The RealView Assembler panel enables you to specify the assembler options. The options are grouped in the following tabs:

- **Target**
- **PCS**
- **Options**
- **Predefines**
- **Listing Control**
- **Diagnostics**
- **Extras.**

The Initial State setting on the **Target** tab has the following options:

- **[auto based on CPU]** (the default for the ARM stationery)
- **ARM**, which inserts the `--arm` assembler option
- **Thumb**, which inserts the `--thumb` assembler option
- **Legacy Thumb**, which inserts the `--16` assembler option.

The default assembler options for the ARM stationery are:

- **Source Line Debug** (`--debug`)
- **Keep Symbols** (`--keep`).

See the *RealView Compilation Tools Assembler Guide* for more details on using the RealView assembler.

#### ————— Note —————

In the **Predefines** tab, if you want use the SETS directive to specify a predefined variable, you can use a quoted string in the String Value field, for example, "My string". When you add the variable, CodeWarrior automatically inserts the escape character prefix `\` for the quotes, for example

```
--predefine="STRING_DEF SETS \"My string\""
```

However, if you want to add this to the Equivalent Command Line field directly, you must include the escape character yourself.

#### RealView Compiler

The RealView Compiler panel enables you to specify the compiler options. The options are grouped in the following tabs:

- **Target**

- **PCS**
- **Source**
- **Debug/Opt**
- **Preprocessor**
- **Code Gen**
- **Diagnostics**
- **Extras.**

See the *RealView Compilation Tools Compiler and Libraries Guide* for more details on using the RealView compiler.

---

**Note**

You must not add include files to a CodeWarrior for RVDS project by using the `-I` option in the Equivalent Command Line of this panel. If you do, CodeWarrior for RVDS has no knowledge of the included files and you lose the Browse information and Error processing associated with the included files.

Instead, add the include file paths using the Access Paths panel for the build target (see *Target panels* on page 4-15).

---

## 4.5.6 Linker panels

The following Linker panels are provided:

### RealView Linker

The RealView Linker panel enables you to specify the linker options. The options are grouped in the following tabs:

- **Output**
- **Options**
- **Listing**
- **Diagnostics**
- **Extras.**

This panel is available when you set the linker to **ARM RealView Linker** in the Target Settings panel (see *Target panels* on page 4-15).

See the description of the Linker in the *RealView Compilation Tools Linker and Utilities Guide*.

## RealView FromELF

The RealView FromELF panel enables you to specify the fromELF options, when fromELF is run as a post linker. The options are grouped in the following tabs:

- **Output**
- **Diagnostics**
- **Extras.**

The default output is to create a plain binary file (--bin), using a default output name.

This panel is available when you set the post-linker to **ARM RealView FromELF** in the Target Settings panel (see *Target panels* on page 4-15).

See the description of the fromELF utility in the *RealView Compilation Tools Linker and Utilities Guide*.

### 4.5.7 Editor panels

The following Editor panel is provided:

#### Custom Keywords

This enables you to add or remove keywords for your project.

### 4.5.8 Debugger panels

The following Debugger panels are provided:

#### Analyzer Connections

Not supported in CodeWarrior for RVDS.

#### Other Executables

Not supported in CodeWarrior for RVDS.

#### Debugger Settings

Not supported in CodeWarrior for RVDS.

#### Remote Debugging

Not supported in CodeWarrior for RVDS.

#### RealView Debugger

Enables you to specify the ARM debugger to use to debug the image created by the build target. You can specify one of the following:

- **RVD (RealView Debugger)**, the default for new projects

- **AXD (ARM eXtended Debugger)**
- **armsd (ARM Symbolic Debugger).**

You can also specify arguments, if your image requires them.

For AXD and armsd, you can choose to start it in Debug or Run mode. Debug is the default.

If you choose to use armsd, you can configure additional settings, such as a debug target to use to run the image.

### **RealView Runner**

Enables you to specify the ARM debugger to use to run the image created by the build target. You can specify one of the following:

- **RVD (RealView Debugger)**, the default for new projects
- **AXD (ARM eXtended Debugger)**
- **armsd (ARM Symbolic Debugger).**

You can also specify arguments, if your image requires them.

For AXD and armsd, you can choose to start it in Debug or Run mode. Run is the default.

If you choose to use armsd, you can configure additional settings, such as a debug target to use to run the image.

## **4.5.9 Miscellaneous panels**

The following Miscellaneous panels are provided:

### **RealView Disassembly**

Enables you to configure the fromELF disassembly features. See the description of the fromELF --text option in the *RealView Compilation Tools Linker and Utilities Guide*.

## 4.6 Using the Thumb ARM Interworking Image stationery

The Thumb ARM Interworking Image stationery is an example of a complex project that uses multiple, dependent, build targets to compile ARM and Thumb code separately, and then link the output into an interworking executable image. This section describes how to use the interworking stationery.

The Thumb ARM Interworking Image stationery is set up to create an ARM Thumb interworking project. The project also has ARM build targets as dependencies of Thumb build targets.

The main steps to create and set up an ARM Thumb interworking project are:

1. Preparing the tutorial (see *Before you begin*).
2. Creating the project (see *Creating an ARM Thumb interworking project* on page 4-28).
3. Adding your source files to the build targets (see *Adding source files to the build targets* on page 4-30).
4. Building the image (see *Building the image* on page 4-32).
5. Running the image in the debugger (see *Running the image in the debugger* on page 4-32).

### 4.6.1 Before you begin

Before you begin this tutorial, create a Tutorial directory in a location of your choice, for example:

C:\Myprojects\Tutorial

#### Sources used in this tutorial

The tutorial uses the source files from the RVDS interwork example project:

`install_directory\RVDS\Examples\...\interwork`

There are two sets of sources in this project:

- Use the following source files for an image with `main()` in ARM state:
  - `armmain.c`
  - `thumbsub.c`.
- Use the following source files for an image with `main()` in Thumb state:
  - `thumbmain.c`

— armsub.c.

Do not copy these sources to the Tutorial directory at the moment. You are instructed to do this as part of the tutorial.

#### 4.6.2 Creating an ARM Thumb interworking project

To create the project for ARM Thumb interworking:

1. Select **File** → **New** from the main menu to display the New dialog box, shown in Figure 4-2.

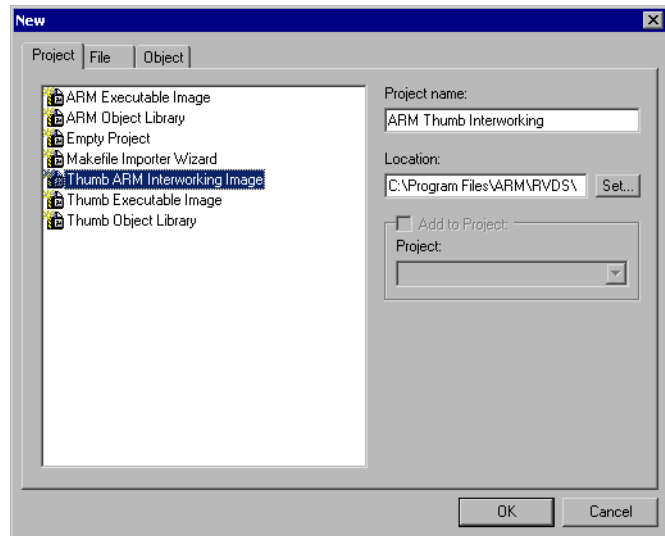


**Figure 4-2** New project dialog box

2. Select the **Thumb ARM Interworking Image** stationery. See *Project stationery supplied with RVDS* on page 4-10 for information about the stationery supplied with RVDS.
3. Click the **Set...** button to display the Create New Project dialog box, and locate the directory that you want to use to hold your new project directory. If you are using the Tutorial directory you created in *Before you begin* on page 4-27, select:
 

C:\Myprojects\Tutorial
4. In the Create New Project dialog box:
  - a. Enter the project name **ARM Thumb Interworking** in the File name field.
  - b. Make sure that the Save as type is set to **Project Files (\*.mcp)**.
  - c. Make sure that the **Create Folder** checkbox is selected.

- d. Click **Save** to close the dialog box.
5. The Project name and Location fields in the New project dialog are set to the values you specified in the Create New Project dialog box.

**Note**

The project name has the same name as the project file.

6. Click **OK** to create the project.

CodeWarrior for RVDS:

- Creates a project directory in the location you specified in step 3. This directory has the same name as the project name you specified in step 4a.
- Sets up the project with the settings defined in the Thumb ARM Interworking Image stationery template.
- Displays the Files view for the new project, shown in Figure 4-3.

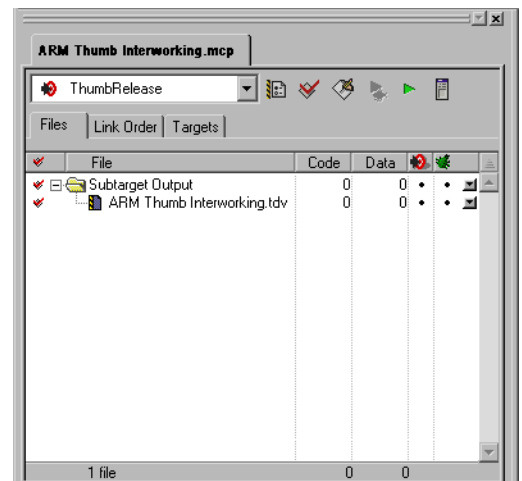


**Figure 4-3 Interworking project Files view**

**Note**

A file with the extension `.tdv` is created as the output for each subtarget. This is an intermediate build file for that subtarget, which CodeWarrior uses in the link stage to create the final output for the main build target.

### 4.6.3 Adding source files to the build targets

To add the sources for the interworking project, you must:

1. Copy the source files to the project directory (see *Copying the source files to the project directory*).
2. Add the ARM source files to the ARM build targets (see *Adding the ARM sources*).
3. Add the Thumb source files to the Thumb build targets (see *Adding the Thumb sources* on page 4-31).

---

#### Note

When you open a project, if any messages are produced, then the Project Messages dialog box has focus. To be able to add files to your project, you must first select the project tab (see Figure 4-3 on page 4-29 for example).

---

### Copying the source files to the project directory

Copy the `armmain.c` and `thumbsub.c` source files from the RVDS interwork example directory (see *Sources used in this tutorial* on page 4-27) to the project directory `directory\Tutorial\ARM Thumb Interworking`.

Alternatively, you can use the `thumbmain.c` and `armsub.c` source files.

### Adding the ARM sources

To add your ARM source files to the ARM build targets:

1. Select **Project** → **Add Files...** from the main menu and select the `armmain.c` source file from the standard file dialog box. CodeWarrior for RVDS displays an **Add Files** dialog.
2. Deselect the Thumb build targets and click **OK** (Figure 4-4 on page 4-31).

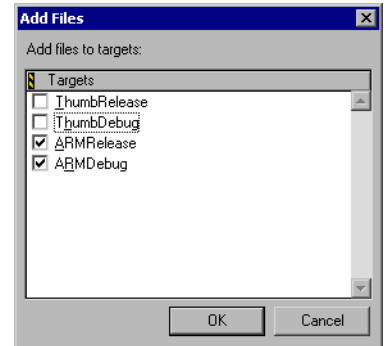


Figure 4-4 Add ARM files

CodeWarrior for RVDS adds the file to the ARM build targets only.

### Adding the Thumb sources

Add your thumbsub.c source file to the Thumb build targets. Follow the same procedure as that described in *Adding the ARM sources* on page 4-30, but select the Thumb build targets when you add the file. Figure 4-5 shows an example of the Files view for the ThumbRelease target. The Files view shows all the files in the project. Files not in the current build target do not have a black dot in the Target column, and have a code size of n/a.

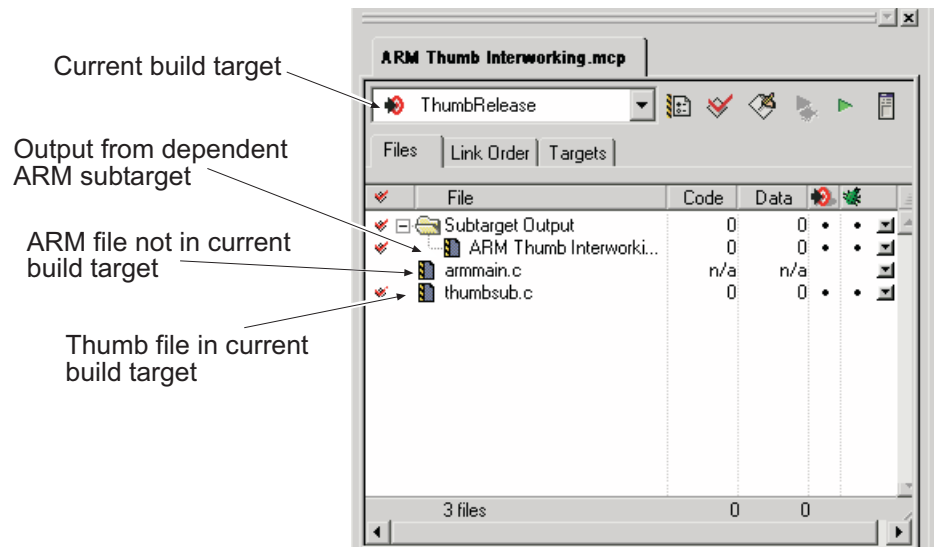


Figure 4-5 Interworking project Files view

#### 4.6.4 Building the image

To build the ThumbDebug build target:

1. Change the current build target to **ThumbDebug**.
2. Select **Project** → **Make** from the main menu.

CodeWarrior for RVDS:

1. builds the dependent ARMDebug build target
2. builds the ThumbDebug build target
3. links the output from the two build targets.

To build the other interworking targets, select the appropriate build target from the **Targets** drop-down list. If you select an ARM build target, only the ARM part of your project source is built.

When you build an image for a particular build target, CodeWarrior for RVDS places the image in the subdirectory for that build target. In this example, the image is placed in the directory:

*directory\Tutorial\ARM Thumb Interworking\ThumbDebug\ARM Thumb Interworking.axf*

You can change this location as described in *Changing the build location*.

#### Changing the build location

If you want to change the build location:

1. Select **Edit** → **TargetName Settings...** from the main menu to display the *TargetName* Settings panel.
2. In the Target Settings panel, click the **Choose...** button to display the Please Select an Access Path dialog box.
3. Locate the directory where you want the project image to be built, then click **OK**.  
The Output Directory path is shown relative to the project directory. For example, if you have created an *images* subdirectory in your project directory, the path is:  
`{Project}\images`

#### 4.6.5 Running the image in the debugger

To run the image in your chosen debugger, follow the instructions given in *Running the image in the Debugger* on page 4-6.

## 4.7 Creating binary ROM images

The default project stationery supplied by ARM Limited is configured to generate a semihosted ELF executable image. To convert a project to produce a simple binary image suitable for embedding in ROM you must:

1. Configure the Target Settings panel to call fromELF as a postlinker. This is the default setting.
2. Configure the ARM fromELF panel to convert the executable ELF image output by the linker to the binary format of your choice. The default output format is **Plain binary**.
3. Configure the ARM Linker panel to create the image structure you require, or use a scatter-load description file to specify your image structure. For more information, see:
  - the example ROM projects in:  
`install_directory\RVDS\Examples\emb_sw_dev`
  - the chapter that describes embedded software development in the *RealView Compilation Tools Developer Guide*.

Also, see the tutorial *Creating a project that builds an object library and ROM images* on page 4-34.

## 4.8 Creating a project that builds an object library and ROM images

This tutorial demonstrates how to create a project that:

- builds an object library
- builds an executable image from the object library and other source files
- converts the executable image to ROM images.

This section contains:

- *Summary of the project*
- *Creating the CodeWarrior project* on page 4-35
- *Setting up the build target for the library* on page 4-36
- *Creating the application build target* on page 4-37
- *Assign the build target dependencies* on page 4-38
- *Adding the sources to the appropriate build targets* on page 4-39
- *Creating the application.bin ROM image* on page 4-40
- *Creating the application.m32 ROM image* on page 4-40.

### 4.8.1 Summary of the project

This tutorial uses the `install_directory\ARM\RVDS\Examples\...\vfpsupport\Build4` example project of RVDS. Analysis of the build options in `build.bat` or `makefile` shows that two build targets are required to build the entire project. The build targets are listed in Table 4-2.

**Table 4-2 Required build targets and their purpose**

| Build target | Purpose                                                                                                                                                                                                                                             |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| application  | To compile and assemble the main project sources, and to create the application.bin Flash image.<br><br>You must modify this to create the application.m32 Flash image, as described in <i>Creating the application.m32 ROM image</i> on page 4-40. |
| vfp_support  | To compile and assemble the library sources, and to create the vfp_support.a object library.                                                                                                                                                        |

---

**Note**

---

When you have build targets that are dependencies of another build target, they must not have the same output name as the dependent build target. Therefore, you might want to use the project name as the Output Name for a build target, and specify a unique Output Name for each dependency of that build target. For clarity, a build target that is a dependency of another build target is subsequently referred to as a subtarget.

The corresponding build target name has been used as the output name in this tutorial, but you can use any unique name.

---

## 4.8.2 Creating the CodeWarrior project

To create the project:

1. Copy the `vfpsupport` example project from the RVDS examples (see *RealView Developer Suite Examples* on page 1-11) to a location of your choice. This copy is used for the rest of the tutorial, and is referred to as:  
`backup_location\vfpsupport`  
If you want to, you can remove the `build1`, `build2`, `build3`, `build5`, `build6`, `build7`, and `util` subdirectories from your copy.
2. Start CodeWarrior for RVDS, and close any other projects that are open.
3. Select **File** → **New...** from the main menu.
4. Select **ARM Executable Image**.
5. Click **Set...** to display the Create New Project dialog box.
6. Locate your `backup_location\vfpsupport` folder.

---

**Note**

---

If you want to use this project to create stationery, you must create the project file in your `backup_location\vfpsupport` folder, rather than the `build4` subdirectory (see *Creating your own project stationery* on page 4-53 for more details). You might also want to remove any files that are not required, such as `*.bat` and `makefile` from your backup directory.

---

7. Enter **vfpsupport**, and deselect **Create Folder**.
8. Click **Save**.
9. Click **OK**. The project file (`vfpsupport.mcp`) and the `Debug`, `Release`, and `vfpsupport_Data` directories are created.

### 4.8.3 Setting up the build target for the library

This project is to use a build target to build the `vfp_support.a` library. To set up the build target for the library:

1. Select the **Targets** tab.
2. Double-click on the Release build target to display the Release Settings dialog box.
3. Select **Target Settings** in the Target Settings Panels list:
  - a. Change the Target Name to **vfp\_support**.
  - b. Change the Linker to **ARM RealView Librarian**.
  - c. Change the Post-linker to **None**.
  - d. For the Output Directory, click **Choose...**, to display the Please Select an Access Path dialog box.
  - e. Select the `vfp_support` subdirectory, then click **OK**. The Output Directory changes to `{Project}\vfp_support`.
  - f. Click **Apply** to apply the settings.

The build target name changes to `vfp_support`.

4. Select **RealView Target** in the Target Settings Panels list:
  - a. Uncheck **Use project name**.
  - b. Enter **vfp\_support** as the Output Name.
  - c. Click **Apply** to apply the settings.
5. Select **RealView Assembler** in the Target Settings Panels list:
  - a. Set the Architecture or Processor to **ARM1020E**.
  - b. Set the Floating Point to **VFPv2**.
  - c. Enter **--fpmode ieee\_full** in the Equivalent Command Line after any other command line options.
  - d. Click **Apply** to apply the settings.
6. Select **RealView Compiler** in the Target Settings Panels list:
  - a. Set the Architecture or Processor to **ARM1020E**.
  - b. Set the Floating Point to **VFPv2**.
  - c. Click the **Debug/Opt** tab.
  - d. Set the Optimization Level to **2 (poor debug view, better code)**.
  - e. Set the Floating Point Models to **Full IEEE**.

- f. Click **Apply** to apply the settings.
7. Click **OK** to save the settings and close the dialog box.
8. Click and drag the `vfp_support` build target so that it appears first in the Targets list.

#### 4.8.4 Creating the application build target

To create the application build target:

1. Double-click on the Debug target to display the Debug Settings dialog box.
2. Select **Target Settings** in the Target Settings Panels list:
  - a. Change the Target Name to **application**.
  - b. Set Linker to **ARM RealView Linker**.
  - c. Make sure Post-linker is set to **ARM RealView FromELF**.
  - d. Click **Apply** to apply the settings.
3. Select **RealView Target** in the Target Settings Panels list:
  - a. Uncheck **Use project name**.
  - b. Enter **application** as the Output Name.
  - c. Set the Output Type to **Linker Output**.
  - d. Click **Apply** to apply the settings.
4. Select **RealView Assembler** in the Target Settings Panels list:
  - a. Set the Architecture or Processor to **ARM1020E**.
  - b. Set the Floating Point to **VFPv2**.
  - c. Enter **--fpmode ieee\_full** in the Equivalent Command Line after any other command line options.
  - d. Click **Apply** to apply the settings.
5. Select **RealView Compiler** in the Target Settings Panels list:
  - a. Set the Architecture or Processor to **ARM1020E**.
  - b. Set the Floating Point to **VFPv2**.
  - c. Click the **Debug/Opt** tab.
  - d. Set the Optimization Level to **2 (poor debug view, better code)**.
  - e. Set the Floating Point Models to **Full IEEE**.
  - f. Select the **Preprocessor** tab.

- g. Enter **USE\_SERIAL\_PORT**, then click **Add**.
  - h. Click **Apply** to apply the settings.
6. Select **RealView Linker** in the Target Settings Panels list:
  - a. Select **Scattered** for the Linktype.
  - b. Enter **build4\scatter.sc** in the Scatter description file field.
  - c. Click the **Options** tab.
  - d. Enter **\_\_main** for the Image entry point.
  - e. Click **Apply** to apply the settings.

This builds the build4\application.axf image.
7. Select **RealView FromELF** in the Target Settings Panels list:
  - a. Make sure that **Plain binary** is the Output format. This is the default setting.
  - b. Enter **application.bin** as the Output file name.
  - c. Click **Apply** to apply the settings.

This converts the Debug\application.axf image to the application.bin ROM image.
8. Click **OK** to save the settings and close the dialog box.

#### 4.8.5 Assign the build target dependencies

To assign the build targets that are to be subtargets of the application build target:

1. Click the **Targets** tab to display the Targets view for the project.
2. Drag and drop the **vfp\_support** build target to **application**. A subtarget is created under **application**.
3. Expand the **application** build target.
4. For the **vfp\_support** subtarget, click the link column so that a black dot appears opposite the subtarget.
5. Set the **application** build target to be the current build target.

#### 4.8.6 Adding the sources to the appropriate build targets

Table 4-3 shows the source files that must be added to each build target.

**Table 4-3 Source files belonging to each build target**

| Build target | Source files | Location                                      |
|--------------|--------------|-----------------------------------------------|
| vfp_support  | *.C<br>*.S   | <i>backup_location\vfpsupport\vfp_support</i> |
| application  | main.c       | <i>backup_location\vfpsupport</i>             |
|              | *.C<br>*.S   | <i>backup_location\vfpsupport\build4</i>      |
|              | *.C<br>*.S   | <i>backup_location\vfpsupport\vfp_init</i>    |

To add the source files to the appropriate build target, do the following:

1. Select **Project** → **Add Files...** from the main menu. The Select files to add... dialog box is displayed.
2. Locate the source file or files you want to add.

**Note**

To select multiple files, press the Ctrl key and select the required files.

3. Click **Open**.
4. Select the build target to which you want to add the file. Make sure you deselect all the other build targets.
5. Click **OK**.
6. Repeat steps 1 to 5 for each directory listed in Table 4-3.

#### Confirming the files are added to the correct build targets

To confirm that you have added the files to the correct build target:

1. Click the **Files** tab to display the Files view for the project.
2. Set a build target to be the current build target.

3. Check that the file or files you have added to the current build target (see Table 4-3 on page 4-39):
  - have a marker (red tick) in the Touch column
  - do not have n/a in the Code and Data columns.Only these sources are built directly for that build target. The remaining source files are built when the associated subtargets are built.
4. Repeat step 2 and 3 for the remaining build targets.

#### 4.8.7 Creating the application.bin ROM image

To create the application.bin ROM image:

1. Click the **Targets** tab to display the Targets view for the project.
2. Set the application to be the current build target.
3. Select **Project** → **Make** from the main menu.

The ROM image is built in the main vfpsupport project directory.

#### 4.8.8 Creating the application.m32 ROM image

To create the application.m32 ROM image:

1. Double-click on the application target to display the application Settings dialog box.
2. Select **RealView FromELF** in the Target Settings Panels list:
  - a. Select **Motorola 32 bit Hex** as the Output format.
  - b. Enter **application.m32** as the Output file name.
  - c. Click **Apply** to apply the settings.
  - d. Click **OK** when CodeWarrior informs you that you must rebuild the target.This converts the Debug\application.axf image to the application.m32 ROM image.
3. Click **OK** to save the settings and close the dialog box.
4. Select **Project** → **Make** from the main menu.

The ROM image is built in the main vfpsupport project directory.

## 4.9 Creating a subproject within another project

A subproject is a standalone project that is nested within another project file. You can use subprojects to organize your build into separate project files that can be separately maintained. For example, you can use subprojects to build and maintain libraries that are used in your main project file.

Subprojects are listed in the Files view of the project window with the other components of your project. They can be assigned to any build target in the main project. When you add a subproject, you can select the build targets to which it belongs.

You can configure your main project so that a Make command builds one or more build targets in a subproject when it builds the containing build target in the main project. You can also configure the main project so that it links the output from any build target in a subproject to any build target in the main project. This means, for example, that you can link the Release build target output from a subproject in which the code is well tested, with the Debug build target of your main development project.

This section includes:

- *Main steps required for compiling and linking a subproject*
- *Subprojects with multiple build targets* on page 4-42
- *Example project with a subproject* on page 4-43.

### 4.9.1 Main steps required for compiling and linking a subproject

The following steps are required to compile and link the output from a subproject with your main project:

1. Add the subproject to one or more build targets in the main project.
2. Specify which, if any, build target in the subproject is to be built when the main project is built. By default none of the build targets is built when the subproject is first added.
3. Specify which, if any, output objects are to be linked with the output from the main project. By default, none of the build target output objects are linked when the subproject is first added.

Each of these steps is independent of the others. For example, you can specify that:

- a subproject build target is built when the main project is built, but not linked with the main project
- the output from a subproject build target is linked, but the subproject build target is not built when the main project is built.

4.9.2 Subprojects with multiple build targets

If you have multiple build targets in a subproject, you can create link dependencies between the main project and any of the build targets in that subproject (see Figure 4-6).

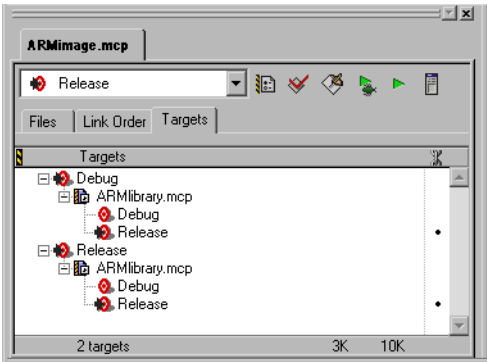


Figure 4-6 Multiple build targets in subprojects

The subprojects are linked in the order given in the Link Order view (see Figure 4-7).

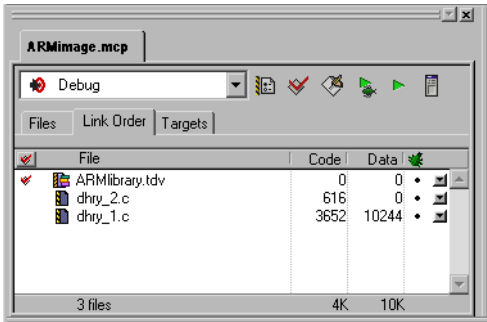


Figure 4-7 Link Order view

This means that, if you are using the ARM project stationery supplied with RVDS (see *Overview of the ARM project stationery supplied with RVDS* on page 4-10), you can create a link dependency to any, or all, of the Debug and ReLease build targets in the subproject, all of which might contain the same code, but built with different optimization and debug options.

When you build your project, the ARM linker selects the first available object file that resolves the unresolved symbol it is processing. However, in the Link Order view, the output filenames for each of the build targets are identical. Therefore, select the output file in the Files view and use the Project Inspector to determine which output object is being linked.

### 4.9.3 Example project with a subproject

The following example shows how to modify the project created in *Creating a project that builds an object library and ROM images* on page 4-34. To do this, you must:

1. Create a new library project to build the `vfp_support.a` object library (see *Creating the library project*).
2. Set up the build target for the library project (see *Setting up the build target for the library*).
3. Add the sources to the build target for the new library project (see *Adding the sources to the build target* on page 4-44).
4. Remove the `vfp_support` build target, and associated source files (see Table 4-3 on page 4-39), from the `vfpsupport` project (see *Removing the vfp\_support build target from the main project* on page 4-45).
5. Add the library project as a subproject to the `vfpsupport` ARM executable image project (see *Adding a project as a subproject* on page 4-45).

The example also gives details on how to specify link and build dependencies.

#### Creating the library project

To create the library project:

1. Start CodeWarrior for RVDS, and close any other projects that are open.
2. Select **File** → **New** from the main menu.
3. Select the **ARM Object Library** stationery.
4. Click **Set...** to display the Create New Project dialog box.
5. Locate your `backup_location\vfpsupport\vfp_support` folder.
6. Enter **vfp\_support**, and deselect **Create Folder**.
7. Click **Save**.
8. Click **OK**. The `vfp_support.mcp` project file and the `vfp_support_Data` directory are created.

#### Setting up the build target for the library

To set up the build target for the object library project:

1. Click the **Targets** tab to display the Targets view for the project.

2. Double-click on the Release build target to display the Release Settings dialog box.
3. Select **Target Settings** in the Target Settings Panels list:
  - a. Change the Target Name to **vfp\_support**.
  - b. Change the Linker to **ARM RealView Librarian**.
  - c. Make sure Post-linker is set to **None**.
  - d. Click **Apply** to apply the settings.The build target name changes to vfp\_support.
4. Select **RealView Assembler** in the Target Settings Panels list:
  - a. Set the Architecture or processor to **ARM1020E**.
  - b. Set the Floating Point to **VFPv2**.
  - c. Enter **--fpmode ieee\_full** in the Equivalent Command Line after any other command line options.
  - d. Click **Apply** to apply the settings.
  - e. Click **OK** when CodeWarrior informs you that you must rebuild the target.
5. Select **RealView Compiler** in the Target Settings Panels list:
  - a. Set the Architecture or processor to **ARM1020E**.
  - b. Set the Floating Point to **VFPv2**.
  - c. Click the **Debug/Opt** tab.
  - d. Set the Optimization Level to **2 (poor debug view, better code)**.
  - e. Set the Floating Point Models to **Full IEEE**.
  - f. Click **Apply** to apply the settings.
  - g. Click **OK** when CodeWarrior informs you that you must rebuild the target.
6. Click **OK** to save the settings and close the dialog box.
7. Right-click on the Debug build target, and select **Delete** from the context menu.

### Adding the sources to the build target

You must now add the source files to the build target as follows:

1. Select **Project** → **Add Files...** from the main menu.
2. Locate your *backup\_location*\vfp\_support\vfp\_support directory.
3. Press the Ctrl key, and select all the \*.c and \*.s source files.

4. Click **Open**.

Because there is only one build target for the project, CodeWarrior automatically adds the sources to it.

### Removing the vfp\_support build target from the main project

To remove the vfp\_support build target from the vfpsupport project:

1. Close the new vfp\_support library project.
2. Open the main vfpsupport project.
3. Click the **Files** tab to display the Files view for the project.
4. Change the current build target to be vfp\_support. Those files that do not have n/a in the Code and Data columns are part of the vfp\_support build target.
5. Select all the source files for the vfp\_support build target. Use the Shift key or Ctrl key if necessary.
6. Right-click on the selected source files, and select **Remove** from the context menu.
7. Click the **Targets** tab to display the Targets view for the project.
8. Right-click on the vfp\_support subtarget, and select **Delete** from the context menu to delete it.
9. Click **OK** when prompted.  
The build target is removed from the project.

### Adding a project as a subproject

To add the vfp\_support library project as a subproject to the vfpsupport project:

1. Add the subproject to the main project:
  - a. Select **Project** → **Add Files...** from the main menu to display the Select files to add... dialog box, shown in Figure 4-8 on page 4-46.

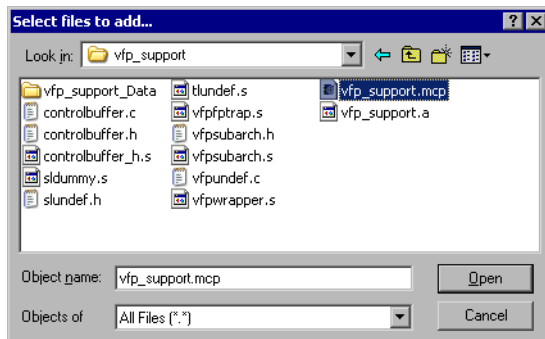


Figure 4-8 Adding a subproject

- b. Locate the *backup\_location*\vfp\_support\vfp\_support directory.
- c. Select the vfp\_support.mcp project file.
- d. Click **Open**.

Because there is only one build target for the project, CodeWarrior automatically adds the library subproject to the application build target. Figure 4-9 shows an example of the Files view for the project.

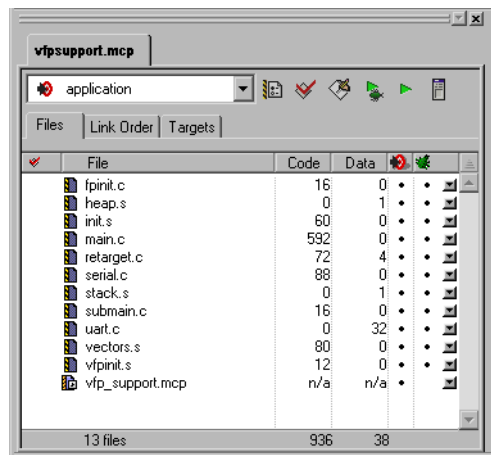


Figure 4-9 Project with subproject Files view

2. Click the **Targets** tab to display the Targets view for the project and click the plus sign next to a build target containing the subproject to expand the hierarchy. Each build target in the subproject is listed in the hierarchy. Figure 4-10 on page 4-47 shows an example.

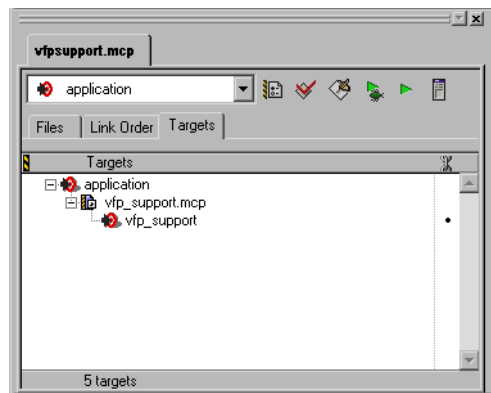


**Figure 4-10 Subproject build target view**

3. Click on the Target icon next to the subproject build target. This ensures that this subtarget is built when the main project is built (Figure 4-11). CodeWarrior for RVDS displays an arrow and target icon for build targets that are selected. When the main project is built, selected subtargets are built first if they have changed, or have been touched.

For example, in Figure 4-11 the vfp\_support subtarget in the example subproject is built before the application build target in the main project.

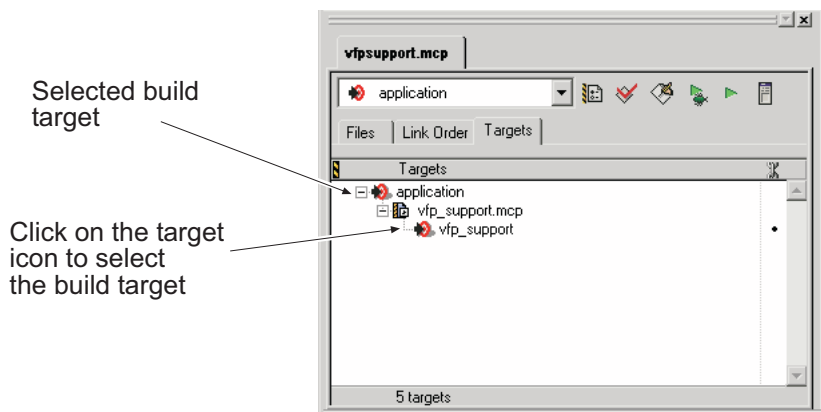


**Figure 4-11 Selecting subproject build targets for building**

4. Click in the link column next to the subproject build targets you want to link with the main project output (Figure 4-12 on page 4-48).

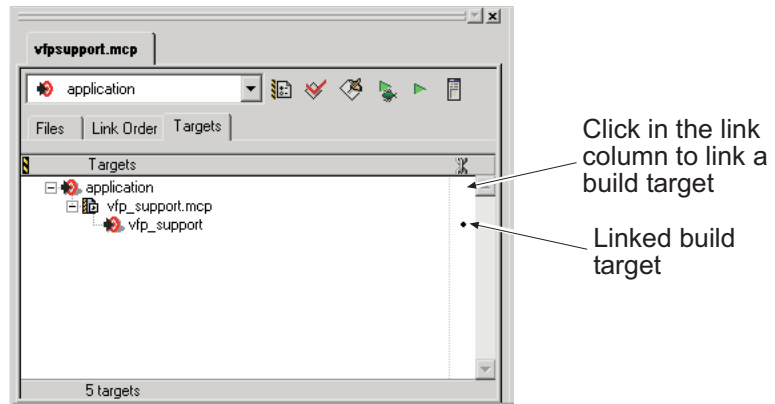


**Figure 4-12 Selecting subproject build targets for linking**

5. You can now rebuild the project.

## 4.10 Running batch files with the batch runner

CodeWarrior for RVDS provides a batch runner utility that you can use to run a DOS batch file from your project. To use the batch runner you must:

- Configure the file mappings for your build target to recognize .bat files. By default, the ARM project stationery supplied with RVDS is configured to recognize batch files.
- Configure the batch runner as the postlinker in the Target Settings panel.

### ———— **Note** ————

You can configure only one postlinker. This means that you cannot use the batch runner and fromELF in the same build target. However, you can run fromELF from the batch file.

- Add the batch file to the build target and make the build target. The batch runner is run only after a successful link operation.

This section includes:

- *Configuring file mappings to recognize batch files*
- *Configuring the batch runner as the postlinker on page 4-50*
- *Adding batch files and making the build target on page 4-51.*

### 4.10.1 Configuring file mappings to recognize batch files

If you use the ARM project stationery supplied with RVDS to create a new project, then the .bat file mapping is already configured.

If you have a legacy project, then you must configure the file mappings for each build target to which you want to add batch files. Do this as follows:

1. Display the Target Settings panel for the build target you want to configure (see *Displaying the build target configuration panels* on page 4-14).
2. Click **File Mappings** in the Target Settings Panels list to display the configuration panel (Figure 4-13 on page 4-50).

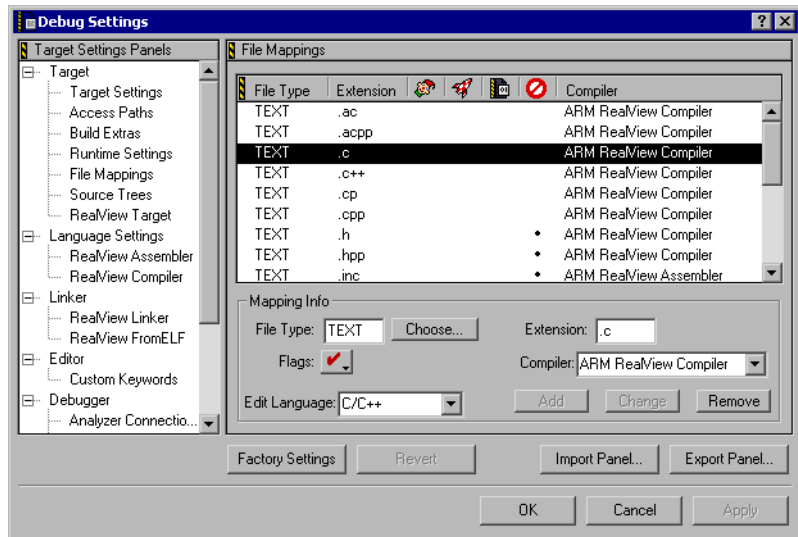


Figure 4-13 File Mappings panel

- Click the entry for the .c mapping in the File Mappings list to select it.
- Change the filename extension, from .c to .bat.
- Click the **Compiler** drop-down list and select **None**. Figure 4-14 shows an example.

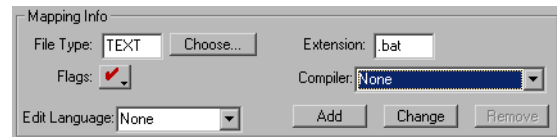


Figure 4-14 Specifying batch file mappings

- Click **Add** to add a new file mapping.
- Click **Save** to save your changes.

#### 4.10.2 Configuring the batch runner as the postlinker

To run batch files from your project, you must configure the batch runner to be run as a postlinker in the Target Settings configuration panel.

To configure the batch runner:

- Display the Target Settings panel for the build target you want to configure (see *Displaying the build target configuration panels* on page 4-14).

2. Click **Target Settings** in the Target Settings Panels list to display the configuration panel (Figure 4-15).

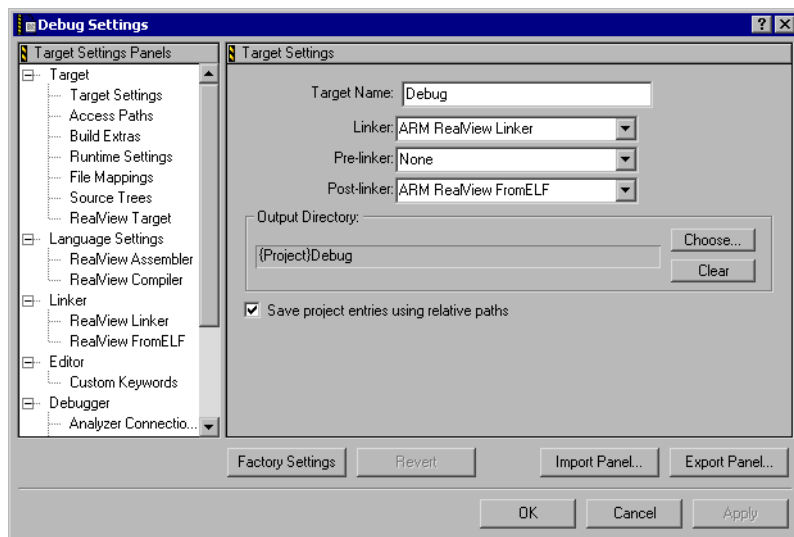


**Figure 4-15 Target Settings panel**

3. Click the **Post-linker** drop-down list and select **Batch File Runner**. See *Summary of the build target configuration panels* on page 4-14 for more information on the other options on this panel.
4. Click **Save** to save your changes.

### 4.10.3 Adding batch files and making the build target

You can add one or more batch files to any build target you have configured to accept files with a .bat extension. You can add any number of batch files to the build target, however the batch runner will only run the batch file listed first in the Link Order view of the project window.

#### **Note**

If you try to add batch files to build targets in the current project that you have not configured to accept .bat file extensions, the CodeWarrior for RVDS displays a warning message and adds the batch files only to the properly configured targets.

To add the batch files and make your build target:

1. Either:
  - select **Add Files...** from the **Project** menu
  - drag and drop the batch files onto the project window.

See the example in *Adding source files to the build targets* on page 4-30 for more information.

2. Touch your project files, if required, to ensure that the project is rebuilt. The batch file runner is executed only after a link step. If your project is up to date, the linker is not executed and the batch file runner is not run.
3. Click the **Link Order** tab to specify which batch file is to be run, if you have added more than one batch file to the build target. Drag the batch file you want to execute so that it is displayed before any other batch file (Figure 4-16).

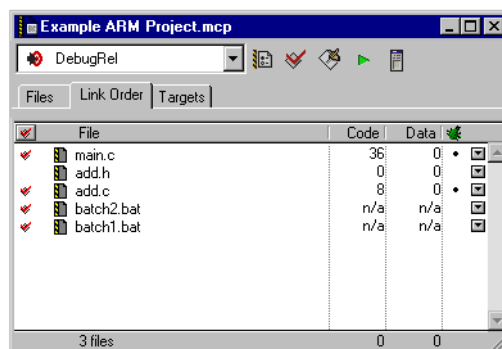


**Figure 4-16 Setting the link order**

4. Select **Make** from the **Project** menu, or click the **Make** button to build your project. The first batch file in the link order list is executed after the linker has completed, regardless of the order of the batch files in the Files view. For example, batch2.bat in Figure 4-16 is executed first.

## 4.11 Creating your own project stationery

You can create your own CodeWarrior for RVDS stationery project file that includes:

- preconfigured build target settings for the project
- predefined build targets, subprojects, and build dependencies
- all files included in the stationery project.

You might want to do this if, for example, you have multiple developers working on different parts of a project, and you want to have them use common configuration settings.

The easiest way to create custom stationery is to modify a project based on existing stationery and save it under a new name in the stationery folder. CodeWarrior for RVDS duplicates the stationery project file and its source files when you create a new project and select your stationery project in the New dialog box.

Before you create your own project stationery, it is recommended that you familiarize yourself with the project stationery supplied by ARM Limited. See *Project stationery supplied with RVDS* on page 4-10 for more information.

---

### Note

You must not create a project directly in the stationery folder. Instead, create a new project in a user defined folder and then save it under a new name in the stationery folder.

---

### 4.11.1 Rules for creating project stationery

When creating project stationery, follow these rules:

- You must create a new folder for your project stationery, and give it a suitable name.
- The project file for your new stationery must be placed in the top-level folder of that stationery.
- If your sources are located in different subdirectories, make sure:
  - your project is created with the project file in the top-level directory of the project
  - you duplicate the project directory structure in the project stationery folder.
- You do not have to copy or duplicate any *project\_Data* directories in the stationery folder.

### 4.11.2 Creating project stationery

This example uses the vfpsupport project described in *Creating a project that builds an object library and ROM images* on page 4-34.

To create your own custom stationery:

1. Open an existing project, or create a new project (see *Creating a new project* on page 4-3), for example, vfpsupport.mcp.
2. Modify the project settings to suit your requirements. You can add and remove files as necessary to create the base project you want.
3. Select **Save A Copy As...** from the **File** menu. CodeWarrior for RVDS displays a Save a copy of project as dialog box (Figure 4-17).

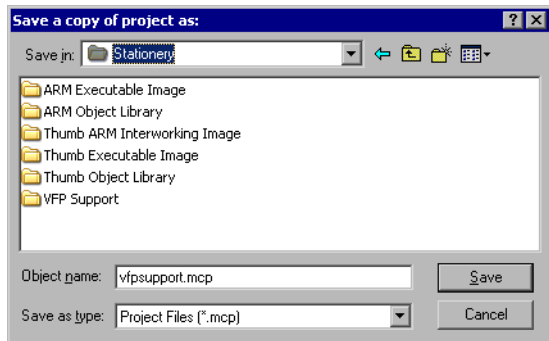


**Figure 4-17 Save a copy of project as dialog**

4. Locate the CodeWarrior Stationery folder (see *The project stationery folder* on page 4-10):
5. Create a new folder for your stationery:
  - a. Click the **Create New Folder** button on the dialog box.
  - b. Enter a name for your new stationery folder, for example, **VFP Support**.
  - c. Double-click on the new folder to make it the current folder. The folder name appears in the Save in field.
6. Enter the name you want to use for the project stationery file in the Object name field, for example **vfpsupport.mcp**. This does not have to be the same name as the stationery folder.
7. Click **Save** to save the file.
8. Copy the project source files to the project stationery folder so that they are copied to new projects created with the project stationery.

---

**Note**

---

If your sources are located in different subdirectories of your project, you must duplicate the directory structure (see *Rules for creating project stationery* on page 4-53).

---

For the vfpsupport project:

- a. Copy the following directories from the vfpsupport directory to the new VFP Support stationery folder:
  - build4
  - vfp\_init
  - vfp\_support.

---

**Note**

---

You do not have to copy the Debug and Release directories, because these are automatically created when you create a new project using this stationery. This is because the directories are specified in the Output Directory setting on the Target Settings panel of the build targets.

---

- b. Copy the main.c file from the vfpsupport directory to the new VFP Support stationery folder.
- c. Remove the following files from the VFP Support\build4 stationery folder:
  - build.bat
  - makefile.
- d. Remove the following files from the VFP Support\vfp\_support stationery folder:
  - build.bat
  - makefile
  - vfp\_support.a.

Your new project stationery is now ready to use. You can select your custom project stationery when you create a new project. The project settings, source files, and any subdirectories in your project stationery are used to create the new project.

## 4.12 Upgrading CodeWarrior for ADS projects

If you have CodeWarrior for ADS projects, you can upgrade them to CodeWarrior for RVDS. To do this:

1. Start CodeWarrior for RVDS as described in *Starting to use CodeWarrior for RVDS* on page 4-3.
2. Select **File** → **Open...** from the main menu to display the Open dialog box.
3. Locate your CodeWarrior for ADS project file (.mcp), and click **Open**.  
You can select more than one file if required. For example, if you want to convert both the interwork project files ARM\_to\_Thumb.mcp and Thumb\_to\_ARM.mcp, press the Ctrl key then select each project file.  
A Convert Project dialog box is displayed, that asks if you want to update the target settings panels to the new CodeWarrior for RVDS format.
4. If you have selected multiple project files to convert, select **Use For All Remaining Projects** to update the target settings panels for all selected CodeWarrior for ADS project files. This also prevents the Convert Project dialog box being displayed when CodeWarrior for ADS converts the remaining project files.
5. Click **OK** to convert the project. CodeWarrior for RVDS:
  - converts the project
  - renames the original project file to *filename.old.mcp*
  - if necessary, creates a subdirectory called *projectname\_Data*, which also contains subdirectories for each build target defined in the original project
  - creates a text file with the filename *ProjectName TargetName compiler command lines.txt*, for each build target containing the build options used.

---

### Note

CodeWarrior for ADS has four compiler panels, because the ARM and Thumb, C and C++ compilers were separate compilers. However, in RVDS the compilers are combined into a single executable. Therefore, in CodeWarrior for RVDS there is only one compiler panel.

Only the settings from the first non-empty command line for the ARM C, ARM C++, Thumb C, and Thumb C++ compilers are used for the converted CodeWarrior for RVDS project. Where multiple distinct compilers are used, the settings for all non-empty ADS panes are stored in the *ProjectName TargetName compiler command lines.txt* files. You can view the contents of these files to determine any missing settings that you might want to re-apply to your project.

---

If you click **Cancel** for a project, then that project is not converted. If you selected multiple projects to convert, then the dialog is displayed again for the next project.

---

**Note**

---

ADS projects contain the Release, DebugRel, and Debug build targets. These build targets are included in your converted project. However, the ARM project stationery provided with RVDS provides only the Release and Debug build targets (see *Predefined build targets in ARM project stationery* on page 4-13).

---

6. Some build tool options cannot be imported from the ADS project. If your ADS project uses options that cannot be imported, they are highlighted in red in the equivalent command line for the associated panel. To fix these, follow the instructions in *Fixing options that cannot be imported from ADS*.

#### 4.12.1 Fixing options that cannot be imported from ADS

If your ADS project uses options that cannot be imported, they are highlighted in red in the equivalent command line for the associated panel. To fix these, do the following for each of the build targets Release, DebugRel, and Debug:

1. Select **Edit** → **TargetName Settings...** from the main menu.
2. Click **RealView Assembler** in the Target Settings Panels list. CodeWarrior for RVDS displays the RealView Assembler panel.
3. Remove any options from the Equivalent Command Line that are highlighted in red.
4. Click **Apply** to save your changes.

---

**Note**

---

If the **Apply** button is not disabled on any pane, you must click **Apply** on that pane to make sure that the changes you make are saved before moving to another pane.

---

5. Click **RealView Compiler** in the Target Settings Panels list. CodeWarrior for RVDS displays the RealView Compiler panel.
6. Remove any options from the Equivalent Command Line that are highlighted in red.

7. Review the remaining compiler options in the equivalent command line and check that they are correct. In particular, if your ADS project is an ARM-Thumb interworking project, only the settings for the ARM C compiler are imported. For a Thumb project only:
  - a. Change the ARM/Thumb State to **Thumb**.  
This adds the `--thumb` compiler option.
  - b. Click **Apply** to save your changes.
8. Click **RealView FromELF** in the Target Settings Panels list. CodeWarrior for RVDS displays the RealView FromELF panel.
9. Remove any options from the Equivalent Command Line that are highlighted in red.
10. Review the remaining fromELF options in the equivalent command line and check that they are correct.
11. Click **Apply** to save your changes.
12. Click **OK** to save your changes and close the *TargetName* Settings dialog box.  
When you have made the appropriate changes to all the build targets, you can build your project.

## 4.13 Creating projects from XML formatted files

You can import project files that are coded in CodeWarrior for RVDS XML format (see *Exporting CodeWarrior for RVDS projects in XML format* on page 4-63). If you want to import a RealView Debugger project then you must first convert it to XML.

This section includes:

- *Importing XML formatted project files.*

### 4.13.1 Importing XML formatted project files

To import an XML formatted project file:

1. Select **File** → **Import Project...** from the main menu. An Open dialog box is displayed.
2. Locate the required XML formatted project file.
3. Click **Open**. A Name new project as dialog box is displayed.
4. Enter a filename for the new project.
5. Click **Save**. The XML project file is imported, and converted to a CodeWarrior for RVDS project file. The project contains the build targets, source files, and settings specified in the XML file.

## 4.14 Importing makefiles

CodeWarrior for RVDS can import simple make files into CodeWarrior project files. CodeWarrior for RVDS provides a Makefile Importer wizard to process the files. The wizard performs the following tasks:

- parses the makefile
- creates a CodeWarrior project
- creates build targets
- adds source files as specified in the makefile
- matches the information specified in the makefile to the output name, output directory, and access paths of the created build targets
- selects a linker to use with the project.

### 4.14.1 Using the Makefile Importer wizard

To create a new project from a makefile:

1. Select **File** → **New...** from the main menu. CodeWarrior for RVDS displays the New dialog box (see Figure 4-1 on page 4-3).
2. Click the **Project** tab and select the **Makefile Importer Wizard** from the list of project stationery.
3. Enter a name and location for the project to be created from the imported makefile (see *Creating a new project* on page 4-3 for naming conventions) and click **OK**. For example, C:\work\Examples\Import.

CodeWarrior for RVDS displays the Makefile Importer Wizard (Figure 4-18 on page 4-61).

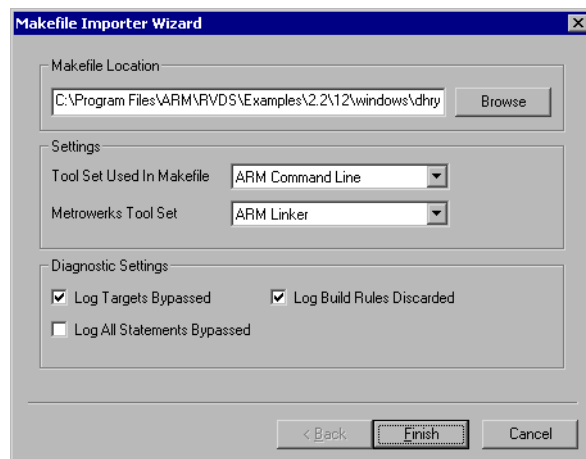


Figure 4-18 Makefile Importer Wizard

4. Enter the location of the makefile in the Makefile Location text field, or click **Browse** to select the makefile from the standard file dialog. The example shown in Figure 4-18 uses the dhry.mk file in the example Dhrystone project supplied with RVDS.

5. Select Settings options:

#### **Tool Set Used in Makefile**

This is set to **ARM Command Line**, which is the only option available.

#### **Metrowerks Tool Set**

This is set to **ARM Linker**, which is the only option available.

6. Select Diagnostic Settings as required (see Figure 4-18):

#### **Log Targets Bypassed**

Select this option to log information about the build targets parsed in the makefile that are not converted to CodeWarrior build targets.

#### **Log Build Rules Discarded**

Select this option to log information about the build rules in the makefile that are discarded in the conversion to a CodeWarrior project.

#### **Log All Statements Bypassed**

Select this option to log the same information as both the **Log Targets Bypassed** and **Log Build Rules Discarded** options, and information about other items in the makefile that are not understood during the parsing process.

Diagnostic messages are displayed in a project message window. The project message window is similar to the message window.

7. Click **Finish**. If the Makefile Importer wizard can successfully process the makefile, it displays a summary window showing the current conversion settings (Figure 4-19).

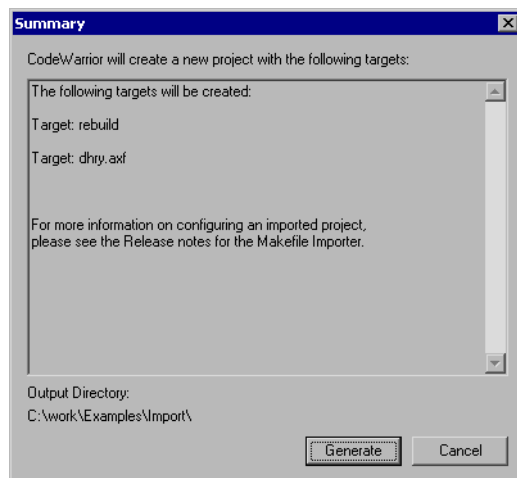


**Figure 4-19 Makefile importer summary**

8. Click **Generate** to import the makefile. CodeWarrior for RVDS generates a new project based on the makefile.

## 4.15 Exporting CodeWarrior for RVDS projects in XML format

To export a CodeWarrior for RVDS project in XML format:

1. Open the CodeWarrior for RVDS project.
2. Select **File** → **Export Project...** from the main menu to display the Export project as dialog box.
3. Click **Save** to save the project file, and close the dialog box. The project file is save with the .xml extension, for example, testproject.mcp.xml.

You can subsequently import the XML formatted project file if required. See *Importing XML formatted project files* on page 4-59 for details.

## 4.16 Using CodeWarrior for RVDS from the command line

CMDIDE.EXE is a console window program that you can start from the command line to build project files that have been created and edited with CodeWarrior for RVDS. CMDIDE.EXE invokes CodeWarrior for RVDS, passes the proper parameters to produce a build, and waits for CodeWarrior to finish its operation.

The command-line arguments are:

|                           |                                                                                                                                                           |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Projectname</i>        | Specifies the project to use.                                                                                                                             |
| <i>/tTargetname</i>       | Specifies a target to become the current target.                                                                                                          |
| <i>/r</i>                 | Removes the objects of the current target before building.                                                                                                |
| <i>/b</i>                 | Builds the current target.                                                                                                                                |
| <i>/c</i>                 | Closes the project after building.                                                                                                                        |
| <i>/f [y n]</i>           | Makes the IDE the front window:                                                                                                                           |
| y                         | Set the focus to the IDE (default).                                                                                                                       |
| n                         | Start minimized.                                                                                                                                          |
| <i>/q</i>                 | Quits CodeWarrior after building.                                                                                                                         |
| <i>/v[y n a]</i>          | Options for converting projects on open:                                                                                                                  |
| y                         | Convert without asking.                                                                                                                                   |
| n                         | Do not convert.                                                                                                                                           |
| a                         | Ask whether to convert.                                                                                                                                   |
| <i>/s</i>                 | Forces the command line to be processed in a new instance of CodeWarrior (rather than using a current instance).                                          |
| <i>/x Projectname.xml</i> | Specifies an XML project file to import.                                                                                                                  |
|                           | A project file argument must also be specified for the name of the imported project. When the project has been imported, build operations might commence. |
| <i>/w n</i>               | Starts with no workspace.                                                                                                                                 |

If more than one project document is specified to be opened in the command line, the */t* target and */b* build command flags apply to the first project document found in the list of documents. If no project is specified in the command line, it uses the usual logic in CodeWarrior (front project or default project) to select the project to build.

For example, to build the dhrystone project, change directory to the dhrystone example directory and type:

```
cmdide dhrystone.mcp /t Debug /c /b
```

If no target is specified, cmdide uses whatever the current target is for that project.

All build commands are executed using a different process from the one launched from the command line. The original process returns when the command line has been completely processed and the build has completed.

The following codes are returned and can be tested using the IF ERRORLEVEL instruction in a batch file:

|    |                                   |
|----|-----------------------------------|
| 0  | No error                          |
| 1  | Error opening file                |
| 2  | Project is not open               |
| 3  | IDE is already building           |
| 4  | Invalid target name (for /t flag) |
| 5  | Error changing current target     |
| 6  | Error removing objects            |
| 7  | Build was canceled                |
| 8  | Build failed                      |
| 9  | Process aborted.                  |
| 10 | Error importing project.          |

---

**Note**

---

Though IDE.EXE understands the same parameters as CMDIDE.EXE, you must use CMDIDE.EXE to ensure that builds are correctly serialized rather than executed all at once.

---



# Appendix A

## Using the armenv Tool

This appendix describes the armenv tool that you can use to manage your ARM RealView product installations. It includes the following sections:

- *About the armenv tool* on page A-2
- *Using the armenv tool* on page A-3.

## A.1 About the armenv tool

The armenv tool enables you to:

- set up and remove the environment variables for ARM RealView products
- check for clashes between the ARM RealView products you have installed
- set up different varieties of the same product.

---

**Note**

---

You cannot use the armenv tool for Custom installations in this release of RVDS.

---

You can find the armenv tool at:

*install\_directory/bin/platform*

## A.2 Using the armenv tool

This section describes the syntax of the armenv command, and shows some examples of how it can be used. It includes:

- *armenv command syntax*
- *armenv command-line arguments*
- *Examples* on page A-5

### A.2.1 armenv command syntax

The command syntax of the armenv tool is:

```
armenv [-r root] [-u] -p product [--and] -p product... [--user|--sys|--proc]
[--bat|--sh|--csh|--posh|--exec program [args]]
```

The arguments are described in *armenv command-line arguments*.

### A.2.2 armenv command-line arguments

Table A-1 shows the command-line arguments that are available on all platforms.

**Table A-1 Generic armenv arguments**

| Argument   | Description                                                                                                                                                    |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| --help     | Displays help on the command-line arguments.                                                                                                                   |
| -r root    | The absolute path to the root of the product installation, <i>install_directory</i> . For example, on Windows the default root is:<br>C:\Program Files\ARM     |
| -p product | The ARM RealView product. See <i>Product syntax</i> on page A-4 for more details.                                                                              |
| --and      | Compute settings for all products before this argument, then do the same for those following it. The settings in the second group override those in the first. |
| --proc     | Change the environment for the current process only. You cannot use this argument with --system or --user on Windows.                                          |
| --exec     | Execute a program in the new environment. You cannot use this argument with --bat on Windows, or with --sh, --csh, or --posh on Sun Solaris and Red Hat Linux. |
| -u         | Attempts to undo the changes to the environment that were made when setting up the product.                                                                    |

Table A-2 shows the command-line arguments that are specific to Windows systems.

**Table A-2 armenv arguments specific to Windows**

| Argument | Description                                                                         |
|----------|-------------------------------------------------------------------------------------|
| --system | Update the Windows SYSTEM area of the registry. This is the default.                |
| --user   | Update the Windows USER area of the registry.                                       |
| --bat    | Changes the environment for the current command prompt window. This is the default. |

Table A-3 shows the command-line arguments that are specific to Sun Solaris and Red Hat Linux systems. You can specify only one of these.

**Table A-3 armenv arguments specific to Sun Solaris and Red Hat Linux**

| Argument | Description                                            |
|----------|--------------------------------------------------------|
| --csh    | Generate a csh syntax shell script.                    |
| --sh     | Generate a sh syntax shell script.                     |
| --posh   | Generate a portable shell script. This is the default. |

**Product syntax**

The syntax for specifying the product is:

`-p category [name] [version [rev]] [-v name value]...`

where:

- category*      The product identifier, for example, RVDS or RVCT.
- name*          Do not use this argument (armenv uses the default name Contents).
- version*       The version number of the product, for example, 2.2. If you do not specify a version, the most recent version of the installed product is used.
- rev*           A specific build number for the product. If you do not specify a build number, the most recent build of the installed product is used.
- v name value*
  - Identifies a variant of the same product.
  - name*          The type of the variant, for example, platform. It is suggested that you use only the platform variant.
  - value*        The specific variant, for example, solaris-sparc.

For example, you might have both Sun Solaris and Red Hat Linux variants of RVDS v2.2 installed. See *Examples* for an example of how to use this argument.

### A.2.3 Examples

To set up the Sun Solaris environment variables for the csh shell, and for the most recent build of RVDS v2.2, enter:

```
armenv -r ~/ -p RVDS 2.2 -v platform solaris-sparc --csh
```

To check for clashes between RVCT v2.2 and RVCT v2.1, enter:

```
armenv -p RVCT 2.2 -p RVCT 2.1
```

To override the RVCT v2.1 settings with the RVCT v2.2 settings, enter:

```
armenv -p RVCT 2.1 --and -p RVCT 2.2
```



## Appendix B

# Using the cwfileassoc Tool

This appendix describes the `cwfileassoc` tool that you can use to modify the CodeWarrior file associations. It includes the following sections:

- *Using the `cwfileassoc` tool* on page B-2.

## B.1 Using the *cwfileassoc* tool

If you chose to install CodeWarrior during the installation of RVDS, you were prompted to specify which file types to associate with CodeWarrior. After installation, you can change the file associations using the *cwfileassoc* tool.

This section describes the syntax of the *cwfileassoc* tool, and shows an example of how to use the tool. It includes:

- *cwfileassoc* command syntax
- Example

You can find the *cwfileassoc* tool at:

*install\_directory*\IDEs\CodeWarrior\CodeWarrior\5.6.1\build\_num

### B.1.1 *cwfileassoc* command syntax

The command syntax of the *cwfileassoc* tool is:

*cwfileassoc* "*install\_directory*"

If your installation directory has spaces, then you must enclose it in double quotes.

### B.1.2 Example

To modify the CodeWarrior file associations with RVDS installed in the default location, enter:

*cwfileassoc* "C:\Program Files\ARM"

You are prompted to modify each of the following file types in turn:

- .mcp
- .c
- .h
- .cpp
- .hpp
- .scat.

At each file type prompt, enter:

**S** To associate the file type with CodeWarrior.

**R** To remove the association of the file type from CodeWarrior.

**A** To leave the association for the file type unchanged.

# Glossary

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

**AAPCS** *See* Procedure Call Standard for the ARM Architecture.

**ABI for the ARM Architecture (Base Standard) (BSABI)**

The ABI for the ARM Architecture is a collection of specifications, some open and some specific to ARM architecture, that regulate the inter-operation of binary code in a range of ARM architecture-based execution environments. The base standard specifies those aspects of code generation that must be standardized to support inter-operation and is aimed at authors and vendors of C and C++ compilers, linkers, and runtime libraries.

**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 legacy ARM architecture-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. ADS is superseded by RealView Developer Suite (RVDS).

*See also* RealView Developer Suite.

**ARM eXtended Debugger (AXD)**

The *ARM eXtended Debugger* (AXD) is a single-processor debugger that runs on Windows platforms. AXD supports Multi-ICE and RealView ARMulator ISS debug targets with legacy ARM7 and ARM9 core modules.

*See also* ARM Symbolic Debugger, Multi-ICE, RealView ARMulator ISS, and RealView Debugger.

**ARM Symbolic Debugger (armsd)**

*ARM Symbolic Debugger* (armsd) is a command-line debugger that runs on all supported platforms. armsd supports only RealView ARMulator ISS debug targets with legacy ARM7 and ARM9 core modules.

*See also* ARM eXtended Debugger, RealView ARMulator ISS, and RealView Debugger.

**armasm** The ARM assembler.

**armcc** The ARM C compiler.

**armlink** The ARM C linker.

**armsd** *See* ARM Symbolic Debugger.

**AXD** *See* ARM eXtended Debugger.

**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.

**CodeWarrior IDE** A development environment for the RealView Developer Suite.

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

**Debug With Arbitrary Record Format (DWARF)**

ARM code generation tools generate debug information in DWARF2 format by default. From RVCT v2.2, you can optionally generate DWARF3 format (Draft Standard 9).

**Digital Signal Processor (DSP)**

DSPs are special processors designed to execute repetitive, maths-intensive algorithms. Embedded applications might use both ARM processor cores and DSPs.

**DSP** *See* Digital Signal Processor.

**DWARF** *See* Debug With Arbitrary Record Format.

**ELF** *See* 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.

**EmbeddedICE logic** The EmbeddedICE logic is an on-chip logic block that provides TAP-based debug support for ARM processor cores. It is accessed through the TAP controller on the ARM core using the JTAG interface.

*See also* IEEE1149.1.

**ETM** *See* Embedded Trace Macrocell.

**ETV** *See* Extended Target Visibility.

**Executable and Linking Format (ELF)**

Executable and Linking Format (ELF) is a format used by the ARM code generation tools to produce objects and executable images.

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

**Extended Target Visibility (ETV)**

Extended Target Visibility enables RealView Developer Suite to access features of the underlying target, such as chip-level details provided by the hardware manufacturer or SoC designer.

**IDE** *See* Integrated Development Environment.

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

**Integrated Development Environment (IDE)**

A development environment offering facilities for automating image-building and file-management processes. The CodeWarrior IDE is an example.

|                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Integrator</b>                                               | A range of ARM hardware development platforms. <i>Core modules</i> are available that contain the processor and local memory.                                                                                                                                                                                                                                                                                         |
| <b>Joint Test Action Group (JTAG)</b>                           | An IEEE group focussed on silicon chip testing methods. Many debug and programming tools use a <i>Joint Test Action Group</i> (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).                                                                                                      |
| <b>JTAG</b>                                                     | <i>See</i> Joint Test Action Group.                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>JTAG interface unit</b>                                      | A protocol converter that converts low-level commands from RealView Developer Suite into JTAG signals to the EmbeddedICE logic and the ETM.                                                                                                                                                                                                                                                                           |
| <b>Little-endian</b>                                            | 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.<br><br><i>See also</i> Big-endian.                                                                                                                                                                                                                       |
| <b>Multi-ICE</b>                                                | A JTAG-based tool for debugging embedded systems.                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Procedure Call Standard for the ARM Architecture (AAPCS)</b> | The Procedure Call Standard for the ARM Architecture specifies a family of <i>Procedure Call Standard</i> (PCS) variants, to define how separately compiled and assembled routines can work together. The standard provides equal support for both ARM state and Thumb state to enable interworking. It favors small code-size, and provides functionality appropriate to embedded applications and high performance. |
| <b>Profiling</b>                                                | Accumulation of statistics during execution of a program being debugged, to measure performance or to determine critical areas of code.                                                                                                                                                                                                                                                                               |
| <b>RDI</b>                                                      | <i>See</i> Remote Debug Interface.                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>RealView ARMulator ISS (RVISS)</b>                           | The most recent version of the ARM simulator, RealView ARMulator ISS is supplied with RealView Developer Suite. It communicates with a debug target using RV-msg, through the RealView Connection Broker interface, and RDI.<br><br><i>See also</i> RDI and RealView Connection Broker.                                                                                                                               |
| <b>RealView Compilation Tools (RVCT)</b>                        | 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 <i>RISC</i> processors.                                                                                                                                                                                                                   |

**RealView Connection Broker**

RealView Connection Broker is an execution vehicle that enables you to connect to simulator targets on your local system, or on a remote system. It also enables you to make multiple connections to the simulator.

*See also* RealView ARMulator ISS.

**RealView Debugger**

RealView Debugger is a multiprocessor debugger that runs on all supported platforms. You can connect to debug targets using Multi-ICE, RVISS, and RealView ICE.

*See also* ARM Symbolic Debugger, Multi-ICE, RealView ARMulator ISS, and RealView ICE.

**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, RVISS)
- 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)**

A JTAG-based debug solution to debug software running on ARM processors.

**RISC**

Reduced Instruction Set Computer.

**RTOS**

Real Time Operating System.

**RVCT**

*See* RealView Compilation Tools.

**RVISS**

*See* RealView ARMulator ISS.

**Simulator**

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

*See also* RealView ARMulator ISS.

|                                    |                                                                                                                                                                                                                                                                                                                                                                                                       |
|------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Target</b>                      | The target hardware, including processor, memory, and peripherals, real or simulated, on which the target application is running.                                                                                                                                                                                                                                                                     |
| <b>Target Vehicle Server (TVS)</b> | 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.                                                                  |
| <b>Tracing</b>                     | 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. |
| <b>TVS</b>                         | <i>See</i> 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

- About this book vi
- Access Paths panel 4-17
- Adding
  - batch files 4-51
- Analyzer Connections panel 4-25
- ARM eXtended Debugger
  - introduction to 1-7
- ARM Symbolic Debugger
  - introduction to 1-8
- armenv
  - command syntax A-3, B-2
- armenv tool A-2
  - arguments (Red Hat Linux) A-4
  - arguments (Sun Solaris) A-4
  - arguments (Windows) A-4
  - arguments, generic A-3
  - examples of use A-5, B-2
- armsd
  - see* ARM Symbolic Debugger
- AXD
  - see* ARM eXtended Debugger

## B

- Batch file runner 4-49
- Book, about this vi
- Browser
  - generating symbolics information 4-19
  - including subprojects 4-18
- Build Extras panel 4-18
- Build target configuration
  - Debugger panels 4-25
  - Editor panels 4-25
  - Language Settings panels 4-23
  - Linker panels 4-24
  - Miscellaneous panels 4-26
  - Target panels 4-15
- Build targets
  - configuring 4-14
  - overview 4-5

## C

- Cache subprojects option 4-18
- CodeWarrior
  - changing file associations B-2
  - command line 4-64
  - converting makefiles to projects 4-60, 4-62
- CodeWarrior IDE
  - configuration panels 4-14
  - overview 1-3
  - starting 4-3
- Commands
  - armenv A-3, B-2
  - cwfileassoc B-2
- Comments
  - on documentation x
  - on RealView Developer Suite x
- Creating
  - interworking projects 4-27
  - project stationery 4-53
- Custom Keywords panel 4-25
- cwfileassoc tool B-2

**D**

Debug target  
   support in ADS 1-14  
   support in RVDS 1-14  
 Debugger panels 4-25  
 Debugger Settings panel 4-25  
 Definition  
   of subproject 4-41  
 Directories  
   documentation 1-2  
   examples 1-2  
   installation 1-2  
 Documentation  
   list of 1-10  
   online 1-10  
   structure of this book vi  
 Documentation feedback x

**E**

Editor panels 4-25  
 Enquiries x  
 Environment variables  
   RVCT22BIN 4-2  
 Example code  
   location of 1-11

**F**

Factory settings  
   panels specific to ARM tools 4-14  
   reverting to 4-14  
 Feedback  
   on documentation x  
   on RealView Debugger x  
 File Mappings panel 4-20  
 FLEXlm  
   licenses 1-9

**G**

Generating symbolics information  
   4-19  
 Glossary Glossary-1

**I**

IDE  
   *see* CodeWarrior IDE  
 Interworking projects, creating 4-27

**L**

Language Settings panels 4-23  
 Licenses 1-9  
 Linker panels 4-24

**M**

Makefile Importer 4-62  
 Makefile Importer Wizard 4-60  
 Makefiles  
   converting to projects 4-60, 4-62  
   importing 4-60  
 Miscellaneous panels 4-26

**O**

Online documentation 1-10  
 Other Executables panel 4-25

**P**

Panel groups  
   Custom panels 4-25  
   Debugger panels 4-25  
   Language Settings panels 4-23  
   Linker panels 4-24  
   Miscellaneous panels 4-26  
   Target panels 4-15  
 Panels  
   Access Paths panel 4-17  
   Analyzer Connections panel 4-25  
   Build Extras panel 4-18  
   Custom Keywords panel 4-25  
   Debugger Settings panel 4-25  
   displaying 4-14  
   File Mappings panel 4-20  
   getting help on 4-14  
   Other Executables panel 4-25

RealView Assembler panel 4-23  
 RealView Compiler panel 4-23  
 RealView Debugger panel 4-25  
 RealView Disassembly panel 4-26  
 RealView FromELF panel 4-25  
 RealView Linker panel 4-24  
 RealView Runner panel 4-26  
 RealView Target panel 4-21  
 Remote Debugging panel 4-25  
 Runtime Settings panel 4-20  
 Source Trees panel 4-21  
 Target Settings panel 4-15  
 Performance reducing build time  
   Cache subprojects option 4-18  
 Problem solving x  
 Product feedback x  
 Project stationery 4-10  
   creating 4-53  
 Projects  
   about project stationery 4-10  
   converting makefiles 4-60, 4-62  
   defined 1-3  
   exporting to XML 4-63  
   subproject definition 4-41

**Q**

Queries x

**R**

RealView ARMulator ISS  
   introduction to 1-7  
 RealView Assembler panel 4-23  
 RealView Compilation Tools  
   introduction to 1-6  
 RealView Compiler panel 4-23  
 RealView Debugger  
   introduction to 1-4  
   Software Problem Report x  
   starting 4-3  
 RealView Debugger panel 4-25  
 RealView Developer Suite  
   components 1-2  
   directories 1-2  
   documentation 1-10  
   licensing 1-9

- list of documents 1-10
- RealView Disassembly panel 4-26
- RealView FromELF panel 4-25
- RealView Linker panel 4-24
- RealView Runner panel 4-26
- RealView Target panel 4-21
- Remote Debugging panel 4-25
- RTOS
  - plugin 1-5
  - support package 1-5
- Running batch files 4-16
- Runtime Settings panel 4-20
- RVCT
  - see* RealView Compilation Tools
- RVCT22BIN environment variable 4-2

## S

- Software Problem Report
  - RealView Debugger x
- Source Trees panel 4-21
- Starting
  - CodeWarrior IDE 4-3
  - RealView Debugger 4-3
- Stationery 4-10
  - creating 4-53
- Structure of this book vi
- Subproject
  - definition 4-41
- Support for RTOS 1-5

## T

- Target panels 4-15
- Target Settings panel 4-15
- Terminology Glossary-1
- Thumb ARM interworking stationery,
  - using 4-27

## U

- Use modification date caching option
  - 4-18
- Using the Thumb ARM interworking
  - stationery 4-27

