RealView Debugger
Essentials Guide

Copyright © 2002-2004 ARM Limited. All rights reserved.

Release Information

The following changes have been made to this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Issue</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 2002</td>
<td>A</td>
<td>RealView Debugger v1.5 release</td>
</tr>
<tr>
<td>September 2002</td>
<td>B</td>
<td>RealView Debugger v1.6 release</td>
</tr>
<tr>
<td>February 2003</td>
<td>C</td>
<td>RealView Debugger v1.6.1 release</td>
</tr>
<tr>
<td>September 2003</td>
<td>D</td>
<td>RealView Debugger v1.6.1 release for RVDS v2.0</td>
</tr>
<tr>
<td>January 2004</td>
<td>E</td>
<td>RealView Debugger v1.7 release for RVDS v2.1</td>
</tr>
</tbody>
</table>

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 Open Access. This document has no restriction on distribution.

Product Status

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

Web Address

http://www.arm.com
### Contents

**RealView Debugger Essentials Guide**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td></td>
<td>vi</td>
</tr>
<tr>
<td>Feedback</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td><strong>Chapter 1</strong></td>
<td>Using the Essentials Guide</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>Using this book</td>
<td>1-2</td>
</tr>
<tr>
<td>1.2</td>
<td>Using the documentation suite</td>
<td>1-3</td>
</tr>
<tr>
<td><strong>Chapter 2</strong></td>
<td>About RealView Debugger</td>
<td></td>
</tr>
<tr>
<td>2.1</td>
<td>RealView Debugger</td>
<td>2-2</td>
</tr>
<tr>
<td>2.2</td>
<td>About the debugging environment</td>
<td>2-4</td>
</tr>
<tr>
<td>2.3</td>
<td>Debugging mode</td>
<td>2-7</td>
</tr>
<tr>
<td><strong>Chapter 3</strong></td>
<td>Features of RealView Debugger</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>RealView Debugger v1.7</td>
<td>3-2</td>
</tr>
<tr>
<td>3.2</td>
<td>Changes between RealView Debugger v1.6.1 and v1.6</td>
<td>3-5</td>
</tr>
<tr>
<td>3.3</td>
<td>RealView Debugger v1.6</td>
<td>3-6</td>
</tr>
<tr>
<td>3.4</td>
<td>Getting more information online</td>
<td>3-9</td>
</tr>
<tr>
<td><strong>Chapter 4</strong></td>
<td>Getting Started with RealView Debugger</td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td>Starting RealView Debugger</td>
<td>4-2</td>
</tr>
<tr>
<td>4.2</td>
<td>Connecting to a target</td>
<td>4-4</td>
</tr>
</tbody>
</table>
4.3 Working with memory ................................................................. 4-8
4.4 Loading an image ................................................................. 4-12
4.5 Debugging an image ................................................................. 4-16

Chapter 5 Quick-start Tutorial
5.1 How to use the tutorial ................................................................. 5-2
5.2 Creating your first project ................................................................. 5-3
5.3 Debugging with RealView Debugger ................................................................. 5-15
5.4 Working with custom panes ................................................................. 5-33
5.5 More about projects ................................................................. 5-35
5.6 Completing the tutorial ................................................................. 5-42

Chapter 6 Ending your RealView Debugger Session
6.1 Saving the session ................................................................. 6-2
6.2 Disconnecting from a target ................................................................. 6-5
6.3 Exiting RealView Debugger ................................................................. 6-8

Chapter 7 RealView Debugger Desktop
7.1 About the desktop ................................................................. 7-2
7.2 Finding options on the main menu ................................................................. 7-12
7.3 Working with button toolbars ................................................................. 7-16
7.4 Working in the Code window ................................................................. 7-20
7.5 Editor window ................................................................. 7-23
7.6 Resource Viewer window ................................................................. 7-24
7.7 Analysis window ................................................................. 7-28

Appendix A Configuration Files Reference
A.1 Overview ................................................................. A-2
A.2 Files in the etc directory ................................................................. A-3
A.3 Files in the home directory ................................................................. A-5

Appendix B Moving from AXD to RealView Debugger
B.1 RealView Debugger configuration ................................................................. B-2
B.2 RealView Debugger operations ................................................................. B-6
B.3 Project management in RealView Debugger ................................................................. B-9

Glossary
Preface

This preface introduces the *RealView Debugger v1.7 Essentials Guide*, that shows you how to start using RealView® Debugger 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 Debugger provides a powerful tool for debugging and managing software projects. This book contains:

- an introduction to the software components that make up RealView Debugger
- a tutorial to create a project and build an executable image
- a step-by-step guide to getting started, making a connection to a target, and loading an image to start a debugging session
- details about ending a debugging session
- a description of the RealView Debugger desktop
- a glossary of terms for users new to RealView Debugger.

Intended audience

This book has been written for developers who are using RealView Debugger to manage ARM® targeted development projects. It assumes that you are an experienced software developer, and that you are familiar with the ARM development tools. It does not assume that you are familiar with RealView Debugger.

This book includes an appendix that contains information for developers moving from ARM eXtended Debugger (AXD) to RealView Debugger.

Examples

The examples given in this book have all been tested and shown to work as described. Your hardware and software might not be the same as that used for testing these examples, so it is possible that certain addresses or values might vary slightly from those shown, and some of the examples might not apply to you. In these cases you might have to modify the instructions to suit your own circumstances.

The examples in this book use the ARM-targeted programs stored in the `\Examples` directory in your RealView Developer Suite root installation.

In general, examples use RealView ARMulator® ISS to simulate an ARM-based debug target. In some cases, examples are given for other debug target systems.
Using this book

This book is organized into the following chapters:

Chapter 1 Using the Essentials Guide
Read this chapter for an introduction to using this book. It describes how this book is organized, where to find specific features, and how to use the tutorial and the rest of the documentation suite.

Chapter 2 About RealView Debugger
Read this chapter for an introduction to RealView Debugger. This chapter describes the underlying debugger concepts and explains terminology used in the rest of this book.

Chapter 3 Features of RealView Debugger
Read this chapter for a history of RealView Debugger releases, including a summary of new functionality in RealView Debugger v1.7.

Chapter 4 Getting Started with RealView Debugger
This chapter explains how to begin using RealView Debugger for the first time. This describes how to start RealView Debugger, make a connection, and load an image ready to start debugging.

Chapter 5 Quick-start Tutorial
Read this chapter when you have access to a workstation. Follow the step-by-step instructions to gain some experience of using RealView Debugger to create a project and debug an application.

Chapter 6 Ending your RealView Debugger Session
This chapter describes how to end your RealView Debugger session and exit the debugger.

Chapter 7 RealView Debugger Desktop
Read this chapter for a detailed description of the contents of the RealView Debugger desktop.

Appendixes and Glossary

Appendix A Configuration Files Reference
Read this appendix for details of the files created when you install RealView Debugger v1.7. It describes where files are stored and what information each file contains.
Appendix B Moving from AXD to RealView Debugger

This appendix is aimed at developers moving from AXD, released as part of ARM Developer Suite™ (ADS) v1.2 and v1.1, to RealView Debugger. Read this appendix for pointers on how to carry out specific tasks in RealView Debugger to make this transition.

Glossary

An alphabetically arranged glossary defines the special terms used in this book.

Typographical conventions

The following typographical conventions are used in this book:

*italic* Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.

*bold* Highlights interface elements, such as menu names. Denotes ARM processor signal names. Also used for terms in descriptive lists, where appropriate.

*monospace* Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.

*monospace italic* Denotes arguments to commands and functions where the argument is to be replaced by a specific value.

*monospace bold* Denotes language keywords when used outside example code.

Further reading

This section lists publications from both ARM Limited and third parties that provide additional information.

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

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

- *RealView Debugger v1.7 User Guide* (ARM DUI 0153)
- *RealView Debugger v1.7 Project Management User Guide* (ARM DUI 0227)
- *RealView Debugger v1.7 Target Configuration Guide* (ARM DUI 0182)
- *RealView Debugger v1.7 Command Line Reference Guide* (ARM DUI 0175)

For details on using the *RealView Compilation Tools* (RVCT), see the books in the RVCT documentation suite.

For details on using RealView ARMulator ISS, see the following documentation:


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

Refer to the datasheet or Technical Reference Manual for information relating to your hardware.

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

- *RealView® ICE User Guide* (ARM DUI 0155)
- *Multi-ICE® User Guide* (ARM DUI 0048)

Other publications

For a comprehensive introduction to ARM architecture see:


For the definitive guide to the C programming language, on which the RealView Debugger macro and expression language is based, see:


For more information about Oak and TeakLite processors from the DSP Group see:

Feedback

ARM Limited welcomes feedback on both RealView Debugger and its documentation.

Feedback on RealView Debugger

If you have any problems with RealView Debugger, submit a Software Problem Report:

1. Select Help → Send a Problem Report... from the RealView Debugger main menu.
2. Complete all sections of the Software Problem Report.
3. To get a rapid and useful response, give:
   • a small standalone sample of code that reproduces the problem, if applicable
   • 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.
4. Email the report to your supplier.

Feedback on this book

If you have any comments on this book, send email to errata@arm.com giving:
• the document title
• the document number
• the page number(s) to which your comments apply
• a concise explanation of your comments.

General suggestions for additions and improvements are also welcome.
Chapter 1
Using the Essentials Guide

This chapter introduces the RealView Debugger Essentials Guide and explains how to use this book to start working with RealView® Debugger. It contains the following sections:

- Using this book on page 1-2
- Using the documentation suite on page 1-3.
1.1 Using this book

You are recommended to read the chapters in this book to start debugging your images and to learn how to use RealView Debugger quickly. This book describes the minimum needed for the new user.

The main part of this book is organized as a tutorial to introduce you to the features of RealView Debugger so that you can start debugging your own images. It describes:

- terminology used throughout the RealView Debugger documentation suite to describe debugging concepts, see Chapter 2 About RealView Debugger
- how to start RealView Debugger and connect to a debug target, see Chapter 4 Getting Started with RealView Debugger
- the steps needed to create your first project to build an image, see Chapter 5 Quick-start Tutorial
- how to start using RealView Debugger to debug that image, see Chapter 5 Quick-start Tutorial
- ending your debugging session and closing down RealView Debugger, see Chapter 6 Ending your RealView Debugger Session
- the RealView Debugger desktop, see Chapter 7 RealView Debugger Desktop.

Read Chapter 3 Features of RealView Debugger for details of the new features in RealView Debugger 1.7 and information about previous releases.

Note

If you are upgrading to RealView Debugger from ARM eXtended Debugger (AXD), use Appendix B Moving from AXD to RealView Debugger for a checklist of the tasks you might have to complete to get started, or for questions about moving to the new debugger.
1.2 Using the documentation suite

The other books that make up the RealView Debugger documentation suite are:

- RealView Debugger User Guide
- RealView Debugger Project Management Guide
- RealView Debugger Target Configuration Guide
- RealView Debugger Command Line Reference Guide
- RealView Debugger Extensions User Guide.

At the front of each book is a Preface describing how the contents are organized and how information is presented in the chapters. The following description explains how you might use the books:

1. For a comprehensive description of the features available in RealView Debugger, see RealView Debugger v1.7 User Guide. This describes, in detail, how to debug your images and how to configure RealView Debugger to customize your working environment. This book also contains examples of debugging software and details shortcuts, and tips, for the developer.

   **Note**
   The RealView Debugger v1.7 User Guide also contains information for developers using RealView Debugger on non-Windows platforms. See Appendix B RealView Debugger for Sun Solaris and Red Hat Linux for details.

2. RealView Debugger uses projects to save your list of files, understand your build model, and maintain a record of your project-level preferences. A project also enables RealView Debugger to save, and load automatically, specified debugging states, for example breakpoints. See RealView Debugger v1.7 Project Management User Guide for a full description of project management in RealView Debugger.

3. RealView Debugger v1.7 Target Configuration Guide describes how to connect to targets, how to amend existing targets that are set up in the base product, and how to customize your own targets.

4. If you want to use the RealView Debugger Command Line Interface (CLI) to control your debugging tasks, RealView Debugger v1.7 Command Line Reference Guide provides a detailed description of every CLI command and includes examples of their use.

5. If you have the appropriate licenses, you can access RealView Debugger extensions, for example multiprocessor debugging mode. These features are described in RealView Debugger v1.7 Extensions User Guide.
See the installation notes delivered with your product for details on installing RealView Debugger.
Chapter 2
About RealView Debugger

This chapter introduces RealView® Debugger. It explains how the debugger provides a development environment for embedded systems applications using the ARM® family of processors.

This chapter contains the following sections:
• RealView Debugger on page 2-2
• About the debugging environment on page 2-4
• Debugging mode on page 2-7.
2.1 RealView Debugger

RealView Debugger enables you to debug your embedded application programs and have complete control over the flow of the program execution so that you can quickly isolate and correct errors.

2.1.1 RealView Debugger concepts and terminology

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

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

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

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

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

**DSP**
RealView Debugger has been developed to provide full debugging functions when working with a range of debug target systems including Digital Signal Processors (DSPs). DSP-based debugging is a separately licensed feature of RealView Debugger.

**RTOS**
Operating systems provide software support for application programs running on a target. Real Time Operating Systems (RTOSs) are operating systems that are designed for systems that interact with real-world activities where time is critical.
Multithreaded operation

RTOS jobs can share the memory of the processor so that each can share all the data and code of the others. These are called threads. RealView Debugger enables you to:

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

RealView Debugger uses a three-tier environment to debug application programs:

- debugger software
- a debug interface layer, incorporating the execution vehicles
- the debug target.

RealView Debugger uses connection information to describe:

- how the debugger connects to the debug target
- information required to use that connection
- what kind of processor the target is using.

Connection information might also include cached copies of processor registers or memory.

This approach means that you can switch between debug targets without having to start a second or third instance of the debugger program.

This section describes the RealView Debugger debugging environment:

- Components of RealView Debugger
- Debug target interface on page 2-5
- Persistence information on page 2-6.

2.2.1 Components of RealView Debugger

RealView Debugger comprises:

**Graphical User Interface (GUI)**

This gives access to the main features of the debugger, command processing, and the Code windows.

**Target Vehicle Server (TVS)**

RealView Debugger maintains connections through the Target Vehicle Server (TVS) and plugins that support each combination of target processor and execution vehicle. Using plugins, for example a board file (*.brd), and Board/Chip definition files (*.bcd), enables RealView Debugger to enumerate advanced information about your target environment, hardware, or processor.

The TVS contains the basic debugging functionality and forms most of the software making up RealView Debugger, for example, it supports multithreaded operation under an RTOS, and enables tracing and performance profiling. If you have the appropriate license, the TVS provides multiprocessor debugging.
RealView Connection Broker

RealView Connection Broker, rvbroker.exe, handles connections to vehicles that reside on the same workstation as RealView Debugger or any other workstation on your network.

RealView Connection Broker operates in two modes:

- **Local**  Operating as RealView Simulator Broker, this runs on your local workstation and enables you to access local targets.
- **Remote** Operating as RealView Network Broker, this runs on a remote workstation and makes specified targets on that workstation available to other workstations connected to the same network.

### 2.2.2 Debug target interface

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

The debug target interface contains the execution vehicles that process requests from the client tools to the target. A debug interface might be a JTAG interface, a simulator, or a ROM monitor.

**RealView ARMulator ISS**

*RealView ARMulator ISS* (RVISS) is an *Instruction Set Simulator* (ISS). It simulates the instruction sets and architecture of ARM processors, together with a memory system and peripherals. You can extend it to simulate other peripherals and custom memory systems.

RVISS runs on the same host computer as the debugger, and includes facilities for communicating with the debugger.

**Note**

RVISS is not the same as the *ARM Developer Suite™* (ADS) ARMulator supplied with previous releases of RealView Debugger.


# 2.2.3 Persistence information

RealView Debugger maintains persistence information to enable you to halt a debugging session and resume at a later date. This means that the debugger can remember your working environment including:

- current target connections
- loaded images
- open projects
- desktop settings, for example pane selections and window positions.
2.3 Debugging mode

The RealView Debugger base product enables you to debug your images in single connection mode, that is, where there is only one connection.

If you have the appropriate license, you can also debug multiprocessor applications. RealView Debugger supports such multiprocessor debugging by maintaining connections to multiple debug targets through one or more Code windows. When working in multiprocessor debugging mode, you can use one Code window to cycle through the connected targets, or multiple Code windows to view different targets.

Multiprocessor debugging mode is a separately licensed feature of RealView Debugger and is described in detail in RealView Debugger v1.7 Extensions User Guide.
Chapter 3
Features of RealView Debugger

This chapter describes the features of RealView® Debugger v1.7 and highlights changes in functionality between RealView Debugger v1.7 and previous releases, that is RealView Debugger v1.6.1, and v1.6. It contains the following sections:

- RealView Debugger v1.7 on page 3-2
- Changes between RealView Debugger v1.6.1 and v1.6 on page 3-5
- RealView Debugger v1.6 on page 3-6
- Getting more information online on page 3-9.
3.1 RealView Debugger v1.7

This section describes the changes between RealView Debugger v1.7 and the previous release RealView Debugger v1.6.1. It contains the following sections:

- Updated documentation
- Advanced debugging facilities
- RealView ARMulator ISS
- Trace, Analysis, and Profiling on page 3-5
- Enhanced RTOS support on page 3-3
- New GUI elements on page 3-4.

3.1.1 Updated documentation

The documentation for developers using RealView Debugger on Windows has been updated to include enhancements and new features in RealView Debugger v1.7. See:

- RealView Debugger Essentials Guide for updated information for developers moving to RealView Debugger from ARM® eXtended Debugger (AXD). See Appendix B Moving from AXD to RealView Debugger for details.

- The detailed description of project management in RealView Debugger has been moved from RealView Debugger User Guide to a new book called RealView Debugger Project Management User Guide.


3.1.2 Advanced debugging facilities

RealView Debugger v1.7 enables you to connect to a RealView ICE® target using RealView ICE.

3.1.3 RealView ARMulator ISS

In RealView Developer Suite (RVDS), RealView ARMulator® ISS (RVISS) replaces ARM Developer Suite™ (ADS) ARMulator.

RVISS enables your debugger to connect using the Remote Debug Interface (RDI) and RealView Connection Broker interface. With RealView Connection Broker you connect to multiple instance of RVISS, and you can also connect to a remote RVISS that is on a different system to your debugger. For more details, see the RealView ARMulator ISS User Guide.
3.1.4 Trace, Analysis, and Profiling

RealView Debugger v1.7 includes enhancements to the Trace and profiling features, including changes to the:

- ETM configuration dialog
- way that trace information is displayed so that interleaved source can be viewed
- method for setting tracepoints
- Set Address/Data Break/Tracepoint dialog
- Analysis window (menu changes).

--- Note ---

RealView Debugger v1.7 enables you to access Trace and profiling features without having to purchase a separate license. These features are part of the core product.

3.1.5 Enhanced RTOS support

RealView Debugger v1.7 includes enhancements to RTOS awareness and visualization.

Running System Debug

*Running System Debug (RSD)* means that you can debug a target when it is running. This means that you do not have to stop your debug target before carrying out any analysis of your system. Where supported by your RTOS, RSD enables you to debug threads individually or in groups.

Thread-based breakpoints

RealView Debugger v1.7 enables you to use the Set Address/Data Break/Tracepoint dialog box and the Break/Tracepoints pane to set thread-based breakpoints when running in RSD mode.

RTOS visualization

This release sees new RTOS visualization features. This gives users an improved threads view in the Process Control pane and provides new menus and tabs in the Resource Viewer window.
RTOS CLI commands

RealView Debugger v1.7 includes new RTOS resource commands that enable you to control RTOS awareness, manage breakpoints and resources, and perform operations on RTOS objects.

Note
RealView Debugger v1.7 does not support RTOS resource CLI commands of the form:

\[D<resource-list>=expression\]

Use \[dos_<resource-list>\] commands instead.

See the chapter describing RTOS support in RealView Debugger v1.7 Extensions User Guide for full details of all the RTOS support provided by RealView Debugger v1.7.

3.1.6 New GUI elements

New toolbar buttons and menu changes mean that RealView Debugger v1.7 users now have quick access to commonly used features. The changes include:

- a new **Thread** menu, available from the main **File** menu, that replicates the drop-down menu from the **Thread** button
- a new Actions toolbar button to hide or display the Connection Control window
- a new Actions toolbar button to disconnect from a target.
- an addition to the Execution group, on the Actions toolbar, to execute a **Go to Cursor** operation.

RealView Debugger v1.7 also includes:

- the ability to choose which register is used as a stack pointer, indicated by a new Expression Pointer (EP).
- user-specified data display in the Stack pane
- type ahead for navigating sources and images in the Process Control pane
- persistence of source search paths and path mappings through project settings
- a new **Help** button on the Project Control dialog box
- improved error messages.
3.2 Changes between RealView Debugger v1.6.1 and v1.6

This section describes the changes between RealView Debugger v1.6.1 and the previous release, that is RealView Debugger v1.6. It contains the following sections:

- Updated documentation
- Advanced debugging facilities on page 3-7
- Trace, Analysis, and Profiling on page 3-7
- RealView Debugger downloads on page 3-8.

3.2.1 Updated documentation

The documentation for developers using RealView Debugger on Windows has been updated to include new features in RealView Debugger v1.6.1. To aid readability, this release is referred to as RealView Debugger v1.6 throughout the rest of the documentation suite.

--- Note ---

The RealView Debugger User Guide also contains information for developers using RealView Debugger on non-Windows platforms. See Appendix C RealView Debugger for Sun Solaris and Red Hat Linux for details.

3.2.2 Advanced debugging facilities

RealView Debugger v1.6.1 enables you to use the debug communications channel (DCC) for semihosting where you are connected to a Multi-ICE® target.

3.2.3 Trace, Analysis, and Profiling

New in RealView Debugger v1.6.1, call-graph profiling enables you to view analysis information about a chosen function in your program, for example which functions call this function and which functions are called by it. You can view this information as a graphical display.

3.2.4 RealView Debugger downloads

The Help menu now contains a new option to give access to the new Updates and Utilities area. See http://www.arm.com to access these resources.
3.3 RealView Debugger v1.6

This section describes the features available in RealView Debugger v1.6. It contains the following sections:

- Multi-core debugging
- OS awareness
- Extended Target Visibility (ETV)
- Advanced debugging facilities on page 3-7
- Trace, Analysis, and Profiling on page 3-7
- Project manager on page 3-8
- RealView Debugger downloads on page 3-8.

3.3.1 Multi-core debugging

RealView Debugger v1.6 provides a single debug kernel for mixed ARM® and DSP debugging. The debugger provides full support for synchronized start and stop, stepping, and cross triggering of breakpoints.

3.3.2 OS awareness

RealView Debugger v1.6 enables you to:

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

3.3.3 Extended Target Visibility (ETV)

RealView Debugger v1.6 provides visibility of targets such as boards and System-on-Chip (SoC). Users can configure targets using Board/Chip definition files and preconfigured files are available:

- ARM family files provided as part of the installation
- customer/partner board files provided through www.arm.com.
3.3.4 Advanced debugging facilities

RealView Debugger v1.6 provides standard debug views and advanced debugging features:

- RealView Debugger supports variables of 64-bit type ‘long long’ throughout the user interface.
- There is now support for module statics, that is static variables of non-local scope, in the Call Stack pane.
- RealView Debugger offers a powerful command-line interface and scripting capability that includes macros support, conversion from ARM eXtended Debugger (AXD) and armd, and history lists to record previous actions.
- Users can access a console (headless debugger) driven from the command line or from scripts.
- RealView Debugger includes an editing control called Tooltip Evaluation that provides hover-style evaluation in different code views.
- RealView Debugger enables you to position a Memory pane to display a memory region based on the contents of a variable or register in the Register or Watch panes, or in the Src tab.
- Users now have greater control over panes in the Code window and the debug views displayed. RealView Debugger provides the option of using a single Code window to display a wide range of data views during debugging (new in v1.6).
- Programming Flash modules are available as standard.
- Memory mapping is enabled if required.
- Colored memory views indicate the type of memory according to memory map settings.

3.3.5 Trace, Analysis, and Profiling

New in RealView Debugger v1.6, Trace, Analysis, and Profiling is enabled by a Trace debug license. Trace support is available for:

- ARM Embedded Trace Macrocell™ (ETM) v1.0 (ETM7™ and ETM9™), including On-Chip Trace
- ARM ETM v2.0 (ETM10™) (beta)
- RVISS ETM simulator
Features of RealView Debugger

- DSP simulators (Oak and TeakLite)
- DSP Group On-Chip Trace (Oak and TeakLite)
- Motorola M56621 On-Chip Trace
- Intel XScale On-Chip Trace.

Trace and Profiling provides full trace support including simple and complex tracepoints and data filtering:
- viewing raw trace
- viewing code trace
- viewing data trace
- viewing disassembly trace
- tracing of function calls
- the profiling of time spent in each function
- the ability to filter captured trace data by field
- the ability to sort captured trace data by field.

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

3.3.6 Project manager

RealView Debugger v1.6 is a fully-featured Integrated Development Environment (IDE) including a project manager and build system.

The project manager includes a Configuration Summary window to display the switch string passed to the compiler tools for build target configurations in the current project.

3.3.7 RealView Debugger downloads

ARM provides a range of services to support developers using RealView Debugger.

The Help menu gives access to the new Updates and Utilities area. See http://www.arm.com to access these resources.

Use this to download RTOS awareness modules, from a range of vendors, to support RTOS developers and provide enhanced support for different hardware platforms through technical information and board description files.
3.4 Getting more information online

The full documentation suite is available online as DynaText and PDF files.

Depending on your installation, select Programs → ARM from the Windows Start menu. From here select:

- DynaText Documentation to view the DynaText files.
- RealView Developer Suite v2.1 → PDF Documentation to view the PDF files.

You can also access the DynaText files from the Help menu when RealView Debugger is running.

Note

The DynaText and PDF files contain the same information.

The RealView Debugger User Guide contains information for developers using RealView Debugger on non-Windows platforms. See Appendix B RealView Debugger for Sun Solaris and Red Hat Linux for more details.
Chapter 4
Getting Started with RealView Debugger

This chapter gives step-by-step instructions to get started with RealView® Debugger, including making a connection and loading an image for debugging. It also covers the main tasks that you might carry out in a debugging session.

It contains the following sections:

- Starting RealView Debugger on page 4-2
- Connecting to a target on page 4-4
- Working with memory on page 4-8
- Loading an image on page 4-12
- Debugging an image on page 4-16.
4.1 Starting RealView Debugger

To start your debugging session, you must complete the following steps:

1. Start RealView Debugger.
2. Connect to your chosen debug target.
3. Load an image for debugging.

This section describes how to start RealView Debugger and display the default Code window. It contains the following sections:

- Starting RealView Debugger
- The Code window

4.1.1 Starting RealView Debugger

To start RealView Debugger:

1. Select Programs → ARM from the Windows Start menu.
2. Select RealView Developer Suite v2.1 → RealView Debugger v1.7 from the menu.

The first time you run RealView Debugger after installation, it creates a unique working directory, in your RealView Debugger home directory, for you to store your personal files, debugger settings, and target configuration files. RealView Debugger then creates files in, or copies files into, this directory ready for your first debugging session.

If a user ID is not specified then RealView Debugger creates a general-purpose working directory called \owner in the default \home directory.

4.1.2 The Code window

Starting RealView Debugger immediately after installation displays the default Code window to provide a starting point for all debugging tasks. The Code window is your main debugging and editing window. This is shown in Figure 4-1 on page 4-3.
The appearance of the Code window depends on your licenses. For example, the base product enables you to debug your images in single connection mode, that is where there is only one connection. If you are licensed to work in multiprocessor debugging mode, the title bar shows your current windows attachment. When you first display the default Code window in multiprocessor mode, this shows [Unattached].

--- Note ---

If you are licensed to use the RealView Debugger multiprocessor extension, your Code window includes other buttons not shown here, for example for making connections in multiprocessor mode, or for viewing threads with RTOS support.

For a full description of the contents of this window, see Chapter 7 RealView Debugger Desktop.
4.2 Connecting to a target

The next stage in your debugging session is to connect to your debug target. The RealView Debugger base product includes built-in configuration files to enable you to make a connection without having to modify any configuration details.

This section introduces target configuration and how to make a connection:

- Target configuration
- Working with connections
- Making a connection on page 4-5
- Setting connect mode on page 4-7.

4.2.1 Target configuration

RealView Debugger uses a **board file** to access information about the debugging environment and the debug targets available to you, for example how memory is mapped. The board file describes your debug target to the debugger. This is called target configuration.

You can start to use RealView Debugger with the default board file installed as part of the base product without making any further changes.

---

**Note**

See *RealView Debugger v1.7 Target Configuration Guide* for details of how to customize your targets.

---

4.2.2 Working with connections

RealView Debugger makes a distinction between target configuration, and how a target is accessed, that is the connection.

Select **File → Connection → Connect to Target...** from the main menu to display the Connection Control window ready to make your first connection. This is shown in Figure 4-2 on page 4-5.

---

**Note**

You can also click the **Connection Control** button, in the Connection group on the Actions toolbar, to display the Connection Control window quickly. If the window is hidden by other windows, click the button twice.
The Connection Control window dynamically details all your connections during a debugging session, gives access to your target configuration details, and enables you to connect to (or disconnect from) targets.

--- Note ---
If you are licensed to use RealView Debugger extensions, the Connection Control window includes tabs (not shown here). For example, in multiprocessor debugging mode, the window includes a **Connect** tab and a **Synch** tab. See the chapter describing connecting to (and disconnecting from) targets in *RealView Debugger v1.7 Target Configuration Guide* for full details on how to use this window.

### 4.2.3 Making a connection

The base product configuration files include the top-level entry **Server**. This is the target vehicle that supports RealView Connection Broker connections to local or remote targets, shown in Figure 4-2. By default, **localhost** targets are defined. For details on how to create connections to remote targets, see the chapter on working with remote targets in *RealView Debugger v1.7 Target Configuration Guide*. 
Expand localhost to show the Simulator Broker targets available on your local machine. The new_ARM target connection uses the RealView Connection Broker interface of RealView ARMulator® ISS (RVISS).

The default configuration files installed as part of the base product enable you to connect to an ARM7TDMI® core using RVISS on your local workstation.

Select the new_ARM check box so that it is checked to make the connection.

With the connection established, your Code window is updated:

- the Code window title bar is updated with the name of the current connection
- the hyperlink in the File Editor pane changes to enable you to load an image
- the Output pane displays details of the connection software
- panes are updated with debug information, for example the Register pane shows the core registers for the connected target.

**Note**

If you are licensed to work in multiprocessor debugging mode, a second new_ARM entry also appears ready to make further connections using RVISS.

If you are always debugging code on the same target you can configure RealView Debugger to make the same connection automatically each time it starts. See the chapter describing connecting to targets in RealView Debugger v1.7 Target Configuration Guide for details of how to set this option.

**RDI connection details**

The top-level entry ARM-A-RR is the target vehicle that supports connections to ARM RDI targets, shown in Figure 4-2 on page 4-5. Expand this to show the ARMulator connection that uses the RDI interface of the local RVISS simulator.

RealView Debugger displays RDI connection details in different tabs depending on the startup conditions and the Code windows you are using. If this is the first time you connect to an RDI target from the default Code window, the startup connection details are displayed in the Log tab and the Cmd tab of the Output pane. In future debugging sessions, this information is displayed in the Cmd tab.
4.2.4 Setting connect mode

You can control the way a target processor starts when you connect. This is particularly useful when debugging multiprocessor debug targets and working with multiple threads. In single processor debugging mode, you might want to leave an image executing while RealView Debugger closes down and then restart at a later date.

If you are not connected, you can set connect mode when you make a new connection:

1. Select File → Connection → Connect to Target..., or click the Connection Control button, to display the Connection Control window.

2. Right-click on the connection entry, for example new_ARM, and select Connect (Defining Mode)... from the Connection context menu.

3. Select the required state from the selection box. The options listed depend on your target vehicle. For example, when connecting to RVISS, the following options are listed:
   - for connections through the RDI interface:
     - No Reset and Stop (the default)
     - No Reset and No Stop (this is not supported in RVISS)
   - for connections through the RealView Connection Broker interface:
     - No Reset and Stop.

4. Click OK to make the connection with the processor in the required state.

Note

If you click Cancel, the connection continues using the default connect mode as defined by the target vehicle.

If you set connect mode from the Connection Control window, this temporarily overrides any setting in your target configuration file.

See the chapter describing connecting to targets in RealView Debugger v1.7 Target Configuration Guide for full details of connect mode and how to specify this setting for your debug target.
4.3 Working with memory

Before you load an image, you might have to define memory settings. This depends on the debug target you are using to run your image. For example, if you are using the default RVISS to simulate an ARM processor, setting the value of top of memory is not appropriate and a default value is used (0x80000).

Where appropriate, defining memory gives you full access to all the memory on your debug target. RealView Debugger enables you to do this in different ways, for example using an include file, or defining the memory map as part of your target configuration settings. For full details of these options, see the chapter describing memory mapping in RealView Debugger v1.7 User Guide.

--- Note ---
In the example in this section, you set up memory manually for the current session. Target memory settings defined in this way are only temporary and are lost when you exit RealView Debugger.

This section describes how to set up memory:
- Setting top of memory and stack values
- Setting top of memory for a session on page 4-9.

4.3.1 Setting top of memory and stack values

The top of memory variable is used to enable the semihosting mechanism to return the top of stack and heap. If you are not using an ARM-based target, or if your target does not use semihosting, this is ignored.

If you do not set these values, RealView Debugger uses default settings that are dependent on the debug target. For ARM processors the default value used for top of memory is 0x20000. If you are using RVISS to simulate an ARM target, the default setting for top of memory is 0x80000.

When you first connect to an ARM-based target, RealView Debugger might display a warning message in the Cmd tab, for example:

Warning: No stack/heap or top of memory defined - setting top_of_memory to 0x80000.

You can set permanent values for top of memory, stack, and heap, using the Connection Properties window. Configure your debug target and define these settings so that they are used whenever you connect. See the chapter describing configuring custom targets in RealView Debugger v1.7 Target Configuration Guide for an example of how to do this.
4.3.2 Setting top of memory for a session

If you are working with an appropriate debug target, you can set the value of top of memory on a temporary basis, that is for the current session, using the @top_of_memory register.

--- Note ---

If you are using the default RVISS to simulate an ARM processor, this is not a suitable target for setting top of memory in this way because top of memory is set from a configuration file rather than from within RealView Debugger.

To set the value of top of memory for an ARM Integrator/AP board and ARM940T core, using Multi-ICE®:

1. Display the Connection Control window to connect to your debug target, as described in Connecting to a target on page 4-4.

2. Select Debug → Memory/Register Operations → Set Register... to display the Interactive Register Setting dialog box.

3. Specify the register to be changed, @top_of_memory, and press Enter to see the current value, shown in Figure 4-3.

4. Enter the new value, for example 0x40000, shown in Figure 4-3.

5. Click Set to update the register contents. The Log display is updated to record the change.

6. Click Close to close the dialog box.
The **Debug** tab, in the Register pane, displays the updated value, shown in the example in Figure 4-4.

![Figure 4-4 Changed settings in the Register pane](image)

The value of top of memory might be displayed in dark blue to show that it has changed since the last update.

If you set this value too low, loading an image to your target might generate a warning message in the **Cmd** tab:

**Warning:** No room for heap - could cause unpredictable behavior.

**In-place editing**

You can use in-place editing to change register contents in the Register pane, for example to change top_of_memory:

1. Double-click in the value you want to change. The value is enclosed in a box with the characters highlighted to show they are selected (pending deletion).
2. Type the new value.
3. Press Enter.

If you press Escape before you press Enter, any changes you have made in the highlighted field are ignored.
For full details on setting top of memory for an ARM-based target, see the chapter describing memory mapping in *RealView Debugger v1.7 User Guide*.
4.4 Loading an image

When you have connected to a suitably configured debug target you are ready to load your image for debugging. This section describes:

- Loading an image
- What is shown in the title bar? on page 4-14
- Reloading an image on page 4-15
- Unloading an image on page 4-15.

4.4.1 Loading an image

In this example, you load the image dhrystone.axf installed as part of the root installation. By default this is located in \dhrystone in the RealView Developer Suite \Examples directory.

Select File → Load Image... to load your image. This displays the Load File to Target dialog box where you can locate the required image and specify the way in which it is loaded.

——— Note ————

Do not change any default settings in the Load File to Target dialog box.

———

Your Code window looks like Figure 4-5 on page 4-13.
In this Code window **Text Coloring** is enabled by default and line numbering is turned on by selecting **Edit → Editing Controls → Show Line Numbers**.

When you load an image, the debugger:

- inserts the source filename, for the current context, in the File field at the top of the File Editor pane
- highlights the location of the *Program Counter* (PC) at the entry point with a red box
- moves the text insertion point to the current location of the PC
- updates the Code window panes as appropriate
- updates the Code window title bar to show the name of the project associated with the image
- displays the load line in the **Cmd** tab in the Output pane.
If an image is compiled to generate debug information, that is using the -g switch, loading the image enables RealView Debugger to gather debug information about the image and the associated source files. In the dhrystone example project, this opens the source file dhryst_1.c into the Src tab when you load the image because it has the context. See Code views on page 4-17 for more details.

4.4.2 What is shown in the title bar?

The Code window title bar gives details of the connection and any processes running on your debug target. If you connect to a target and load an image, your title bar looks like the one shown in Figure 4-6.

Figure 4-6 Code window title bar

In addition to the application icon, you can see (from left to right):

RVDEBUG Identifies the Code window. This changes to identify each new Code window that you open, for example RVDEBUG_1, or RVDEBUG_2.

(dhrystone) The project associated with the loaded image.

@SimARM_1:Sim The current connection, including the connection number and the target vehicle.

[Unattached] If you are working in multiprocessor debugging mode, this shows the windows attachment, that is whether the window is attached to a connection.

In multiprocessor debugging mode, a Code window is unattached by default, shown by [Unattached].

In single-processor debugging mode, this part of the title bar is blank.

If you float a pane from a Code window, the pane title bar reflects the title bar of the calling Code window.

Note

The contents of your title bar might be different from the one shown in Figure 4-6 depending on your licenses, the current connection (if any), open projects and windows attachment. For a full description of the contents, see Chapter 7 RealView Debugger Desktop.
4.4.3 Reloading an image

During your debugging session you might have to amend your source code and then recompile. Select File → Reload Image to Target from the Code window to reload an image following these changes.

Reloading an image refreshes any window displays and updates debugger resources.

— Note —

For full details on loading and reloading images, see the chapter describing working with images in RealView Debugger v1.7 User Guide.

4.4.4 Unloading an image

You do not have to unload an image from a debug target before loading a new image for execution. Display the Load File to Target dialog box and ensure that the Replace Existing File(s) check box is selected ready to load the next image.

However, you might want to unload an image explicitly as part of your debugging session, for example if you correct coding errors and then rebuild outside RealView Debugger. You can do this using the Process Control pane:

1. Select View → Pane Views → Process Control Pane from the default Code window main menu.
2. Right-click on the Image entry, for example dhrystone.axf, or on the Load entry, Image+Symbols, to display the Image context menu.
3. Select Unload.

You can also unload an image by clicking on the check box associated with the Load entry so that it is unselected.

Unloading an image does not affect target memory. It unloads the symbols and removes most of the image details from RealView Debugger. However, project details are not cleared and the image name is retained.

— Note —

To remove image details completely, right-click on the Image entry in the Process Control pane and select Delete Entry. This removes any project details associated with the image.
4.5  Debugging an image

Chapter 5 Quick-start Tutorial provides details on using the features of RealView Debugger with your images. This section summarizes how to start debugging with RealView Debugger:

- Getting started
- Code views on page 4-17
- Viewing target status on page 4-17.

4.5.1  Getting started

You can start debugging your image when you have completed the following steps:

1. Start RealView Debugger, see Starting RealView Debugger on page 4-2.
2. Connect to your target, see Making a connection on page 4-5.
3. Set top of memory, if appropriate for your debug target, see Setting top of memory for a session on page 4-9.
4. Load your image, see Loading an image on page 4-12.

To start your debugging session:

1. Select Edit → Editing Controls → Show Line Numbers to display line numbers. This is not necessary but might help you to follow the examples.
   Your Code window looks like the default window shown in Figure 4-5 on page 4-13.
2. Right-click in the first entry in the Memory pane, <NoAddr>, and select Set New Start Address... from the context menu.
3. Enter a value as the start address for the area of interest, for example 0x8008.
4. Click Set to confirm the setting and close the dialog box.
5. Click on the Src tab in the File Editor pane.
6. Set a simple, software breakpoint at line 149 in dhry_1.c, Proc_5(); by double-clicking on the line number.
   If the line number is not visible, then double-click inside the gray area at the left of the required statement in the File Editor pane to set the breakpoint.
Note
This example assumes that memory mapping is disabled (the default). If you have enabled memory mapping, RealView Debugger might set a hardware breakpoint depending on the type of memory at this location.

7. Set a watch by right-clicking on the variable Int_1_Loc at line 152 in dhry_1.c so that it is underlined in red. Select Watch from the context menu.

8. To start execution either:
   - Select Debug → Execution Control → Go (Start Execution) from the main menu.
   - Click the Go button on the Actions toolbar.

9. Enter the required number of runs, for example 50000.

10. Monitor execution until the breakpoint is reached.

11. Click Go again and monitor the program as execution continues.

4.5.2 Code views
Use the File Editor pane to view source code during your debugging session. In the example shown in Figure 4-5 on page 4-13, the File Editor pane contains three tabs:

- the Dsm tab enables you to track program execution in the disassembly-level view
- the Src tab enables you to track program execution in the source-level view
- the file tab dhry_1.c shows the name of the current source file in the editing, or non-execution, view.

The Src tab acts like a button to display the current source if it is available. If the source is not available, RealView Debugger displays a No source message in the tab.

Click on the relevant tab to toggle between the different code views.

4.5.3 Viewing target status
The State group, on the Actions toolbar, shown in Figure 4-5 on page 4-13, enables you to see the current state of your debug target:

Unknown Shows that the current state of the target is unknown to the debugger. For example it might have been running when the connection was established or it might be disconnected.
**Stopped**  Shows that the target is connected but any image loaded is not executing.

**Running**  Shows that an image is executing. In this case, a running progress indicator is also included.
Chapter 5
Quick-start Tutorial

This chapter provides a step-by-step tutorial using RealView® Debugger to debug your images. All the tasks introduced in this chapter, and the RealView Debugger options used, are described in full in RealView Debugger v1.7 User Guide.

This tutorial contains the following sections:

- How to use the tutorial on page 5-2
- Creating your first project on page 5-3
- Debugging with RealView Debugger on page 5-15
- Working with custom panes on page 5-33
- More about projects on page 5-35
- Completing the tutorial on page 5-42.
5.1 How to use the tutorial

The tutorial starts by creating a user-defined project to build an image for debugging. A user-defined project in RealView Debugger is not required for debugging, but it can provide a powerful aid to development. A project enables RealView Debugger to save your list of files, understand your build model, and maintain a record of your project-level preferences. In this tutorial, you build source files installed as part of the root installation and then debug the executable.

If you have started RealView Debugger, as described in Starting RealView Debugger on page 4-2, you can begin to use many features of the debugger, for example editing source code, creating new projects, and building existing projects. However, to begin debugging images you must connect to a suitably configured debug target.

If you do not set up your own project, you can follow the tutorial using the supplied project, named dhrystone.prj, installed in the install_directory\RVDS\Examples\... directory. This sample project comes with a ready-built image, named dhrystone.axf.

5.1.1 Getting started

Begin by making a copy of the source files provided so that the tutorial is self-contained and the installed example files are untouched:

1. Create a new directory called Tutorial, in your RealView Developer Suite examples directory install_directory\RVDS\Examples\...\windows\ This is the tutorial project base directory.

2. Copy the required files, dhry.h, dhry_1.c, and dhry_2.c, from the examples directory, that is ...\dhrystone, into the new tutorial directory.

Start your session so that you can follow the tutorial:

1. Start RealView Debugger, as described in Starting RealView Debugger on page 4-2.

2. Connect to the ARM7TDMI® core processor using RealView ARMulator® ISS (RVISS), as described in Making a connection on page 4-5.

You can complete the tutorial using the default files provided in the root installation. It is not necessary to change any of these files or to amend any configuration files.
5.2 Creating your first project

RealView Debugger enables you to set up different types of user-defined projects:

- Standard project, including Compile/Assemble/Link
- Library project, including Compile/Assemble/Add to library
- Custom project, using your makefile or defining a no-build project to hold only image and properties
- Container project, composed of existing projects
- Copy, created by copying existing projects.

When you create a new user-defined project, you can also merge a saved auto-project.

For full details on creating different types of project, merging project settings, and accessing the project management features of RealView Debugger, see RealView Debugger v1.7 Project Management User Guide.

This section takes you through the basic steps to set up a Standard user-defined project based on a set of example source files copied from the root installation. Follow this section to specify the default behavior for your C, C++, or assembly language programs, and build an image. This section describes:

- Defining your build tools on page 5-4
- Creating a new project on page 5-5
- Defining a Standard Project on page 5-6
- Viewing the project settings on page 5-7
- Setting up compiler options on page 5-8
- Viewing build settings on page 5-9
- Project base directory on page 5-10
- Building the application on page 5-10
- Project files on page 5-11
- Closing the project on page 5-12.

--- Note ---

This section introduces the Build-Tool Properties window and the Project Properties window to set up your first development project. There are full descriptions of the general layout and controls of these settings windows in the RealView Debugger online help topic Changing Settings.
5.2.1 Defining your build tools

RealView Debugger provides support for multiple toolchains and can locate your build tools automatically. You can use these tools in every project you create or to build source files outside a project. However, you can override these settings for specific projects if required.

To see your build tools:

1. **Select Project → Build-Tool Properties...** from the Code window main menu. This displays the Build-Tool Properties window shown in Figure 5-1.

![Figure 5-1 Build-Tool Properties for a Typical installation](image)

This shows a Typical installation where support for *ARM Developer Suite™* (ADS) and *RealView Compilation Tools* (RVCT) is installed (RVCT is the default toolchain). If you have installed a Custom configuration your window looks different.

When you are working with the Build-Tool Properties window, click on an entry in the left or right pane to see a one-line text explanation in the Description field at the top of the window. Right-click on an entry and select **Detailed Description...** to see extended online help.

2. **Select File → Close window** to close the Build-Tool Properties window without making any changes.
The first time you open the Build-Tool Properties window, RealView Debugger copies the file `genmake.loc` from the default settings directory (`\etc`) into your home directory ready for building operations during your debugging sessions.

The `genmake.loc` file is updated each time you amend your Build-Tool Properties window settings. You are warned before changes are saved in this file, but you can restore all entries to the installation defaults if required.

--- Note ---
For details of how to change your build tools see Customizing your build tools on page 5-37.

### 5.2.2 Creating a new project

You are now ready to set up the new project:

1. Select **Project → New Project...** from the default Code window main menu.
   
   This displays the Create New Project dialog box. The Project Base field might be prefilled with your RealView Debugger installation directory name as defined by your environment variable. You can override this.

2. Enter the project details shown in Figure 5-2.

![Create New Project](image.png)

**Figure 5-2 Creating a new project**

RealView Debugger confirms that the specified project base directory exists. If the directory does not exist, you are given the option to create the directory ready for your project files.
5.2.3 Defining a Standard Project

When you click OK to close the Create New Project dialog box, RealView Debugger displays the Create Standard Project dialog box where you specify the:

- toolchain you are using to build project files
- source files to include in the build process
- image name.

To define the project:

1. Click on the down arrow to specify the Toolchain that you are using to build your images, for example ARM-C21, or ARM-C2, or ARM-ADS. The available options depend on the tools support you have installed, for example RVCT or ADS.

2. Click the folder icon to open the project base directory, defined previously, and specify the source files to use in the build process. This displays the Select source files for Project dialog box where you can highlight one or more files. Use the Shift or Ctrl keys to select the files dhry_1.c and dhry_2.c.

3. Click Open and add the required source files to your project. The Create Standard Project dialog box looks like Figure 5-3.

![Figure 5-3 Create Standard Project dialog box](image)

The Executable field contains the image name, that is Tutorial.axf. Do not change this so that you can follow the rest of the tutorial.

You do not have to change the project Description field.
4. Click **OK** to confirm your entries and close the Create Standard Project dialog box.

Closing the dialog box creates the project settings file and opens the project into the debugger, shown in the Code window title bar.

### 5.2.4 Viewing the project settings

When you close the Create Standard Project dialog box, RealView Debugger displays the Project Properties window, shown in Figure 5-4.

![Figure 5-4 Project Properties window](image)

The Project Properties window enables you to view project settings as defined in the project file. Click on the entry `...\Tutorial.prj` at the top of the list of entries in the left pane of the window. This displays the full path of the project settings file in the Description field at the top of the window. In this new standard project this is identified as `install_directory\RVDS\Examples\...\windows\Tutorial\Tutorial.prj`.

Most entries in the Project Properties window are filled automatically from the Create Standard Project dialog box. You do not have to change any entries. Select **File → Close Window** from the menu to close the Project Properties window.

When you examine your project settings and then close the Project Properties window, RealView Debugger regenerates the makefiles. The **Build** tab in the Output pane displays information about the generation process. You must wait for this to complete before making the next change. See *Generated makefiles* on page 5-39 for more details.
5.2.5 Setting up compiler options

For this tutorial, you must specify a preprocessor macro that is included as part of the build model. You have to set the -D compiler switch to specify how the compiler processes #if directives. If you are working on Windows, you must set this to \texttt{MSC\_CLOCK} to specify the C function library to control how timing measurements are made.

To do this you must change a project setting:

1. Select \texttt{Project} \rightarrow \texttt{Project Properties...} from the default Code window main menu to display the Project Properties window.
2. Click on \texttt{ORIGINAL=arm} in the left pane to see the contents.
3. Double-click on Preprocessor in the right pane to see the contents.
4. Right-click on Define in the right pane and select \texttt{Edit Value} from the context menu.
5. Type \texttt{MSC\_CLOCK} and press Enter.
   An asterisk is placed at the front of the setting to show that it has changed from the default.
6. Select \texttt{File} \rightarrow \texttt{Save and Close} from the menu to close the Project Properties window.

RealView Debugger regenerates the makefiles, shown in the \texttt{Build} tab in the Output pane. You must wait for this to complete before making more changes.

Customizing your project

You can also make other changes to the project to specify the build model, for example to suppress compiler warning messages. To do this you must change a project setting:

1. Select \texttt{Project} \rightarrow \texttt{Project Properties...} from the default Code window main menu to display the Project Properties window.
2. Click on \texttt{ORIGINAL=arm} in the left pane to see the contents.
3. Use the \texttt{Extra_args} setting in the right pane to specify additional compiler arguments.
   Right-click on \texttt{Extra_args} and select \texttt{Edit Value} from the context menu.
   Type \texttt{-W} and press Enter.
   In the ARM\textsuperscript{\textregistered} compilers, the option \texttt{-W} suppresses all warning messages.
   An asterisk is placed at the front of the setting to show that it has changed from the default.
4. Double-click on Messages in the right pane to see the contents. These settings enable you to control error messages and warnings issued by the compiler.

5. Right-click on Brief in the right pane and select enabled from the context menu. In RVCT, the compiler option --brief_diagnostics enables or disables a mode where a shorter form of diagnostic output is used. If enabled, the original source line is not displayed. This setting is disabled by default.

6. Select File → Save and Close from the menu to close the Project Properties window.

RealView Debugger regenerates the makefiles, shown in the Build tab in the Output pane. You must wait for this to complete before making more changes.

5.2.6 Viewing build settings

The Project Properties window includes another window that enables you to see a summary of the compiler, assembler, and linker command lines generated by the current project settings. This window provides a read-only display of the command-line switches for each of the current build target configurations.

To view your current compiler settings:

1. Select Project → Project Properties... from the default Code window main menu to display the Project Properties window.

2. Click on COMPILE=arm in the left pane of the Project Properties window to see the ARM C compiler (armcc) command line.

3. Select View → Configuration Summary to display the Configuration Summary window, shown in Figure 5-5.

![Configuration Summary window](image)

Figure 5-5 Configuration Summary window

This window contains one tab for each of the build target configurations specified for the chosen group, that is Debug, Release, and DebugRel.

If you change a compiler setting, the Configuration Summary window updates automatically.
4. Click on \texttt{ASSEMBLE=arm} in the left pane of the Project Properties window to see the ARM assembler (\texttt{armasm}) command line in the Configuration Summary window.

5. If you select an entry in the left pane that does not contain compiler, assembler, or linker settings, for example \texttt{CONFIGURATION}, the Configuration Summary window displays the message:

\texttt{NO COMPILE, ASSEMBLE or BUILD group is selected.}

6. Use Alt+F4 to close the Configuration Summary window.

If you close the Project Properties window without closing the Configuration Summary window first, RealView Debugger displays both windows automatically when you next open the Project Properties window in this session.

7. Select \texttt{File} \rightarrow \texttt{Close Window} from the menu to close the Project Properties window without making any changes.

\subsection*{5.2.7 Project base directory}

When the new project setup is complete, your project base directory is updated with the files required to manage your new project. These are the:

- project file \texttt{Tutorial.prj}
- build target configuration directories \texttt{Debug}, \texttt{Release}, and \texttt{DebugRel}
- generated makefiles for each build target configuration, for example \texttt{Tutorial_Debug.mk}.

\begin{quote}
\textbf{Note}
\end{quote}

The project source files do not have to be in the project base directory, although this is recommended for single-user, self-contained projects.

\subsection*{5.2.8 Building the application}

If you have the ARM C compiler installed on your workstation, you can now build the application defined by the example project \texttt{Tutorial.prj}. If you do not have the compiler installed, you can follow the steps to complete the tutorial but you cannot build an executable.

To build the executable for the example project:

1. Select \texttt{Tools} \rightarrow \texttt{Build...} from the default Code window main menu.

2. If you have made any changes to the Project Properties, or to the Build-Tool Properties, you are prompted to rebuild all project files.
Click **Yes** to confirm the rebuild.

The build, or rebuild, completes and RealView Debugger displays the **Build** tab, in the Output pane, to report successful completion. If any errors or warnings are generated during the build, these are also reported in the **Build** tab.

### 5.2.9 Project files

The project you have just created is a single-user, self-contained project. This means that the project base directory now contains all the files associated with the tutorial project, as described in Table 5-1.

<table>
<thead>
<tr>
<th>Project contains</th>
<th>Directory/filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug directory</td>
<td>Debug</td>
<td>This area contains the object files and the executable ready for debugging or execution. By default, <em>Debug</em> is specified as the active configuration for this project. This means that this is the build target configuration that is built and loaded. Change this using the Project Properties window to view, and amend, the <strong>CONFIGURATION</strong> group, shown in Figure 5-4 on page 5-7.</td>
</tr>
<tr>
<td>DebugRel directory</td>
<td>DebugRel</td>
<td>This area is empty.</td>
</tr>
<tr>
<td>Release directory</td>
<td>Release</td>
<td>This area is empty.</td>
</tr>
<tr>
<td>Source files</td>
<td>dhry_1.c, dhry_2.c, dhry.h</td>
<td>The original source files and headers for the project. If any files have been edited, this area also includes the backup files (see <strong>Backup files</strong> on page 5-12).</td>
</tr>
</tbody>
</table>
Executable files

By default, the executable image created in this project is saved in your project directory `install_directory\RVDS\Examples\...\Tutorial\Debug\Tutorial.axf`. You can copy this file to another location or share it with others in your development team. You can load the image to a target processor without opening the project first. However, where you have created a user-defined project, it is recommended that you open the project first to load and debug the associated image. Opening the project enables you to access the project properties, save new settings, or make changes to the build model.

Backup files

If you make any changes to the project, a backup file is automatically created to enable you to recover from any accidental editing or to restore previous settings. Similarly, changing a source file in the File Editor pane also creates a backup file for safety. These files are given the .bak extension by default, for example `dhry_1.c.bak`, and `Tutorial.prj.bak`, and are located in the project base directory.

5.2.10 Closing the project

The default Code window title bar shows the name of your new project:

```
RVDEBUG(Tutorial) = @SimARM_1:Sim
```
Because you are connected to an ARM processor, the project is automatically associated with the connection. This is called project binding. The project name, Tutorial, is enclosed in round brackets to show that it is bound to the connection.

--- Note ---

The contents of the Code window title bar depend on your debugging environment:

- if you are licensed to work in multiprocessor debugging mode, the title bar shows the windows attachment, that is:
  
  RVDEBUG(Tutorial) = @SimARM_1:Sim [Unattached]

- if several projects are open, the title bar shows the name of the active project.

See the chapter describing project binding in RealView Debugger v1.7 Project Management User Guide for details on controlling how projects bind.

---

You can keep projects open while you are debugging. This might be useful to add new files to the project or if source files change. It is not necessary to keep the project open to debug the executable you just created.

To close the project, select **Project → Close Project...** from the Code window main menu. Because there is only one open project, it closes immediately.

--- Note ---

If you have loaded the image created by your project, RealView Debugger gives you the option to unload the image. Unload the image associated with the project to avoid the creation of an auto-project (see Working with images on page 5-19 for more details).

---

The Code window title bar shows that the project is no longer open.

Any files displayed in the File Editor pane remain after the parent project closes. To close the file shown on the top tab, either:

- select **File → Close** from the Code window main menu
- right-click on the file tab and select **Close** from the context menu.

If any file has been edited, you are warned and given the option to save the file before it closes. A backup copy of the previous version is saved by default, unless you have changed this in your workspace.

If you have several files displayed in the File Editor pane, the next tab is brought to the top and you can then close this one in the same way.
In the next part of the tutorial you use the debugging features of RealView Debugger to load an executable image and monitor execution. It is not necessary to exit RealView Debugger at this stage.
5.3 Debugging with RealView Debugger

This section gives you step-by-step instructions to carry out some basic debugging tasks. These examples use the sample project dhrystone.prj, supplied as part of the RealView Debugger root installation. These projects assume the ARM-C21 toolchain, that is building with the RVCT v2.1 tools for ARM processors. If you prefer, you can use the executable built in Creating your first project on page 5-3 and saved in the tutorial project, Tutorial.prj.

You can use any project to complete this section. However, RealView Debugger determines the location of your ARM build tools automatically (see Defining your build tools on page 5-4 for details). If you open an old project and a later version of the toolchain exists, RealView Debugger prompts you to upgrade the project. You must upgrade the project before completing these examples. See Upgrading projects on page 5-35 for more information.

--- Note ---
If you are not licensed to use RealView Debugger extensions, your Code window might look different to the one shown in the rest of this tutorial. This does not affect the tutorial.

This section describes:
- Getting started
- Basic debugging tasks on page 5-16
- Using breakpoints on page 5-27.

5.3.1 Getting started

Complete these steps so that you can follow the rest of the tutorial:

1. Start RealView Debugger, as described in Starting RealView Debugger on page 4-2.
2. Connect to the ARM7TDMI core processor using RVISS, as described in Making a connection on page 4-5.
3. Select Project → Open Project... to open the required project, for example dhrystone.prj in the install_directory/RVDS/Examples/... directory.
4. Click on the hyperlink in the File Editor pane to load the associated image. This location has been derived automatically from the project information.
5. In these examples, Text Coloring is enabled by default and line numbering is turned on by selecting Edit → Editing Controls → Show Line Numbers.
The default Code window title bar shows the name of your open project:

RVDEBUG(dhrystone) = @SimARM_1:Sim [Unattached]

--- Note ---
If you are using the supplied source files, RealView Debugger might warn that the source is more recent than the executable. This message can be ignored.

5.3.2 Basic debugging tasks

In your debugging session RealView Debugger enables you to examine registers, memory contents, and variables:

- *Displaying register contents*
- *Changing register contents* on page 5-17
- *Process control* on page 5-18
- *Displaying memory contents* on page 5-20
- *Displaying variables* on page 5-21
- *Tooltip evaluation* on page 5-22
- *Using the call stack* on page 5-22
- *Using the stack* on page 5-23
- *Using browsers and lists* on page 5-25
- *Setting watches* on page 5-26.

**Displaying register contents**

To display register contents for the loaded image:

1. Select **View → Pane Views → Registers** from the default Code window to display the Register pane. It looks like the example in Figure 5-6 on page 5-17.
Figure 5-6 Register pane

The Register pane displays tabs appropriate to the target processor running your image and the target vehicle used to make the connection. For the ARM7TDMI core using RVISS, the pane includes the Core tab, showing the base registers for the connected target processor, and the CycleCount tab, showing internal debugger variables and statistics.

If you are connected to real target hardware, for example using RealView® ICE, the Register pane might include other tabs, for example CP15 and Debug.

2. Click the Pane menu and select **Show Enumerations as Values** from the available options. This displays the register contents as values rather than enumerated strings. The Register pane is refreshed.

3. Click the Pane menu and unselect **Show Enumerations as Values**.

**Changing register contents**

To change the contents of registers:

1. Click on the Core tab in the Register pane.

2. Right-click in the Mode field of the *Current Program Status Register* (CPSR) register (SVC) to display the context menu shown in Figure 5-7 on page 5-18.
3. Select **Set Enumeration**... This opens a selection box where you can specify the context-sensitive value to modify the selected register entry.

4. Select **FIQ** from the list of available options and click **OK** to change the Mode field of the CPSR from SVC (0013) to FIQ (0011). This updates the contents of the ARM banked registers, shown in the pane. Similarly, if you are in SVC mode and you change the contents of R13 in the SVC bank (SP), the contents of R13 (SP) change to match.

   In this example, the value can also be changed by selecting **FIQ** from the list of values from the context menu shown in Figure 5-7.

   For full details on working in the Register pane, and the contents of tabs, see the chapter describing working with debug views in *RealView Debugger v1.7 User Guide*.

**Process control**

To display the Process Control pane:

1. Select **View → Pane Views → Process Control Pane** from the default Code window. It looks like the example in Figure 5-8 on page 5-19.
The Process tab shows details about your current process. If you are debugging a single process application, this is the same as viewing the processor details. In this example, you can identify the target processor and see details about your project and the loaded image.

2. Right-click on an entry in this pane, for example dhrystone.axf, and select Properties from the context menu. This displays a text box giving more information on the chosen item.

**Working with source files**

When working with entries in the Process Control pane, you can use type ahead facilities to locate files. This is especially useful where your project specifies a large number of source files. For example, type the first letter of the source file that you want to view. RealView Debugger expands the Sources entry and locates the first matching occurrence. When using this feature, the type ahead buffer is case insensitive and is limited to 128 characters. Do one of the following to clear the buffer:

- select a different item
- press Home to move to the top of the pane
- press Escape.

**Working with images**

Where you have created a user-defined project, it is recommended that you open this first to load and debug the associated image, or images. This enables you to access the project properties, save new settings, or make changes to the build model. In this example, the user-defined project is open so project settings have been used to populate entries in the Process tab.
When an image is loaded directly to a debug target, RealView Debugger checks to see if an auto-project exists for that image in the same location. Where no auto-project exists, RealView Debugger creates an in-memory project to use in the current session.

An auto-project is a custom, image control, project that holds project settings where the build model is unknown. You can view these settings using the Project Properties window. Using an auto-project enables you to amend and save image load parameters where you do not have a user-defined project or where you have no control over the build model. You also have the option to use the saved auto-project as the basis of a user-defined project.

Note
If you load an image built as part of a user-defined project without opening the project you cannot access all the project properties because these are unknown to RealView Debugger. In this case, RealView Debugger creates an in-memory project, or uses the saved auto-project file.

See the chapter on working with images in RealView Debugger v1.7 User Guide for details on using auto-projects in RealView Debugger.

Displaying memory contents

To display an area of memory:

1. Select View → Pane Views → Memory from the default Code window to display the Memory pane.

2. Right-click on an entry <NoAddr> to display the context menu shown in Figure 5-9.

3. Select Set New Start Address... to display the address prompt box.

4. Enter 0x8000 as the new start address and click Set to confirm your choice and close the prompt.

5. Click on the Memory pane Pane menu and select Show ASCII to display the updated memory contents, shown in Figure 5-10 on page 5-21.
Figure 5-10 Updated memory contents

Click on the Pane menu again and select Set Number of Columns to show... to choose how many columns are used in the memory display, for example 12. If you do not specify the number used (or specify zero), RealView Debugger displays as many columns as it can fit into the pane.

6. Click the Go button on the Actions toolbar to execute the image. Enter a large number of runs, for example 50000.

7. Click Stop Execution to stop the program before it finishes and view the updated memory contents, shown in dark blue in the Memory pane.

At the next update, memory contents might be colored light blue. This shows that they changed at the previous update. Seeing the update depends on the memory contents that you can see and where execution stops.

8. Click on the file tab for the source file dhry_1.c and locate scanf at line 124.

9. Double-click on scanf so that it is highlighted and then drag it to the Memory pane where you can drop it into the display. This is a quick way to display a chosen location in the Memory pane.

Use the Pane menu, or right-click in an address entry in the Memory pane, to select the display format and modify the address range of the memory area that you want to see.

You can change memory contents displayed in the Memory pane using in-place editing.

**Displaying variables**

To display the value of a variable from your source code:

1. Select File → Reload Image to Target to reload the image.
2. Click the Go button on the toolbar to execute the image for a number of runs, for example 5000.

3. Select the required variable in the current context, for example click on the file tab for the source file dhry_1.c and move to line 301. Highlight the variable Ptr_Glob in the expression:

\[
\text{structassign (}*\text{Ptr_Val_Par->Ptr_Comp, *Ptr_Glob};
\]

4. Right-click to display the Source Variable Name menu, shown in Figure 5-11.

5. Select Print to view the value of the chosen variable in the current context. This is displayed in the Cmd tab of the Output pane.

6. Select View Memory At Value to display the memory view at this location.

**Tooltip evaluation**

Use Tooltip Evaluation to see hover-style evaluation when you hold your mouse pointer, for two seconds, over a variable in the Src tab, or a register in the Dsm tab.

This option is enabled by default. Select Edit → Editing Controls → Tooltip Evaluation to disable this for the current Code window.

**Using the call stack**

The Call Stack pane enables you to follow the flow of your program by examining the current status of functions and variables. This pane shows you the path that leads from the main entry point to the currently executing function.
To monitor your execution path:

1. **Select View → Pane Views → Call Stack** from the default Code window to display the Call Stack pane.

2. **Select File → Reload Image to Target** to reload the image.

3. **Click the Go button on the Actions toolbar to execute the image. Enter a large number of runs, for example 50000.**

4. **Click Stop Execution** to stop the program before it finishes.

   The **Call Stack** tab, in the Call Stack pane, displays details of the functions currently on the stack and awaiting execution.

5. **Right-click on an entry in the Call Stack pane, to see the Function context menu.**

   This menu enables you to carry out operations on the chosen function in the stack, for example to scope to that function.

6. **Click on the Locals tab in the Call Stack pane.** This displays a list of the variables that are local to the current function, shown in the example in Figure 5-12.

   ![Figure 5-12 Local variables in the Call Stack pane](image)

   If a variable is a structure or an array, a plus sign is added to the entry in the Call Stack pane. You can click on this to expand the variable to see all elements of the structure or array.

7. **Right-click on an entry in the Locals tab, to see the Variables context menu.**

8. **Click on the Statics tab, to see non-local variables, that is module statics.**

**Using the stack**

The Stack pane enables you to monitor the contents of the stack as raw memory and to make changes to those settings. The Stack pane enables you to follow the flow of your application through the hierarchical structure and shows you the path that leads from the main entry point to the currently executing function.
Quick-start Tutorial

To view the Stack pane:

1. Select View → Pane Views → Stack from the default Code window to display the Stack pane.
2. Select File → Reload Image to Target to reload the image.
3. Click the Go button on the Actions toolbar to execute the image. Enter a large number of runs, for example 50000.
4. Click Stop Execution to stop the program before it finishes.
5. View the updated Stack pane, shown in Figure 5-13.

![Figure 5-13 Viewing the stack](image)

The stack pointer, marked by SP, is located at the bottom of the stack. The frame pointer, marked by FP, shows the starting point for the storage of local variables.

**Changing the stack pointer**

As you step through your code, the default stack pointer is used, shown in Figure 5-13. You can specify an expression or a register to use as the stack pointer from the Stack pane:

1. Right-click on an entry to display the context menu shown in Figure 5-14 on page 5-25.
2. Select **Set New Start Address**... to display the address prompt box.

3. Enter an expression or a register, for example @R9, as the new stack pointer.

4. Click **Set** to confirm your choice and close the address prompt box.

The new stack pointer is marked by **Expression Pointer (EP)**, located at the bottom of the stack.

If you enter a blank expression, or remove the existing expression, in the address prompt box, RealView Debugger reverts to using the default stack pointer register. In this example, this was R13 shown in Figure 5-6 on page 5-17.

**Using browsers and lists**

RealView Debugger provides browsers and lists to help with debugging tasks. These enable you to search through your source files to look for specific structures and to monitor their status during program execution. Browsers are available for:

- project modules and files
- functions
- variables
- C++ classes.

To access the browsers:

1. Select **Find** from the default Code window main menu to display the **Find** menu. The available lists are shown at the bottom of the menu.

2. Select **Function List**... to display the Function List.

3. Highlight the required function.
4. Use the controls in the dialog box to:
   • view details about the function
   • perform actions, for example scoping to the function or setting a breakpoint
   • display the source-level view or the disassembly-level view.

5. Click **Close** to close the Function List.

See the chapter describing working with browsers in *RealView Debugger v1.7 User Guide* for full details on using browsers and lists.

### Setting watches

Watches monitor variables during execution of your image.

To set a series of watches and to monitor their value during execution of the image:

1. Select **File → Reload Image to Target** to reload the image.
2. Click the **Go** button on the Actions toolbar to execute the image. Enter a large number of runs, for example 50000.
3. Click **Stop Execution** to stop the program before it finishes.
4. Select **View → Pane Views → Watch** from the default Code window to display the Watch pane.
5. Highlight the required variable in the current context.
6. Right-click to display the **Source Variable Name** menu, shown in Figure 5-11 on page 5-22.
7. Select **Watch** from the menu. This adds the chosen variable to the Watch pane and displays the current value, if known.
8. Click the **Go** button on the toolbar to execute the image.
9. Click the **Stop Execution** button to stop the program before it finishes.

   With the processor halted, monitor changes in the variables in the Watch pane. Depending on where you halted execution, values that changed at the last pane update are displayed in dark blue. Values that changed at a previous pane update are displayed in light blue.

   You can remove variables from the Watch pane. Highlight the entry and press **Delete**. The values list is refreshed after the entry has been removed.

For full details on setting watches, see the chapter describing working with debug views in *RealView Debugger v1.7 User Guide*. 
5.3.3 Using breakpoints

Breakpoints are specified locations where execution should stop. The breakpoint is triggered either by:

- execution reaching the specified address
- data values at the specified location, in the current context, changing or becoming equal to a particular value.

Breakpoints let you suspend program execution when the program accesses specific memory locations, variables, or functions. You can define conditions that are tested or qualify the breakpoint to define when execution stops. This section describes:

- Breakpoint types
- Running and Halted System Debug on page 5-28
- Actions and qualifiers on page 5-28
- Managing breakpoints on page 5-31.

Breakpoint types

RealView Debugger enables you to use different types of breakpoint when you are debugging your image. Breakpoint types are dependent on the hardware support provided by your debug target:

**Simple** These breakpoints enable you to test address-specific data values. These breakpoints can be either hardware or software breakpoints.

Simple breakpoints are supported by all ARM processors.

**Complex** These breakpoints use advanced hardware support on your target processor, or as implemented by your simulator software, for example RVISS.

Check your hardware characteristics, and your vendor-supplied documentation, to determine the level of support for complex breakpoints.

--- Note
For RVISS, only the RealView Connection Broker interface supports hardware breakpoints.

RealView Debugger provides the Set Address/Data Break/Tracepoint dialog box to enable you to specify your breakpoint details and define how execution continues after the breakpoint triggers.
**Viewing your hardware characteristics**

To see your hardware support for complex breakpoints select **Debug → Complex Breakpoints → Show Break Capabilities of HW...** from the default Code window main menu. This displays an information box describing the support available for your target processor.

**Running and Halted System Debug**

RealView Debugger supports different debugging modes:

**Halted System Debug**

*Halted System Debug* (HSD) means that you can only debug a target when it is not running. This means that you must stop your debug target before carrying out any analysis of your system.

HSD is always available as part of the RealView Debugger base product.

**Running System Debug**

*Running System Debug* (RSD) means that you can debug a target when it is running. This means that you do not have to stop your debug target before carrying out any analysis of your system.

RSD is only available where supported by your debug target. It relies on having RealView Debugger RTOS extensions installed and is not provided as part of the base product.

Where RSD is supported, RealView Debugger enables you to switch seamlessly between RSD and HSD mode using Code window controls or CLI commands. For full details on using RTOS extensions and running in RSD, see the chapter describing RTOS support in *RealView Debugger v1.7 Extensions User Guide*.

**Actions and qualifiers**

Breakpoints are also classified depending on the action taken by RealView Debugger to trigger the breakpoint:

**Unconditional**

The program stops if execution reaches the specified location.

**Conditional**

The program stops if execution reaches the specified location and one or more predefined conditions are met. The conditions that must be satisfied can be defined based on data values or on counters.
Setting conditional breakpoints

Use the Set Address/Data Break/Tracepoint dialog box to set a simple, conditional breakpoint:

1. Select **File** → **Reload Image to Target** to reload the image.
2. Select **File** → **Open...** and open the source file `dhry_2.c` so that it is displayed in the File Editor pane.
3. Scroll through the source file `dhry_2.c` and locate the line:
   ```c
   81 */ Proc_7 */
   ```
4. Right-click on the line number and select **Set Break...** from the context menu to display the Set Address/Data Break/Tracepoint dialog box shown in Figure 5-15.
   
   If the line number is not visible, then right-click inside the gray area at the left of the required statement or on the line of code to display the context menu.

   **Note**

   Ensure that you select **Set Break...** from the context menu. If you select **Set Break (double click)**, the Set Address/Data Break/Tracepoint dialog box dialog box is not displayed and the breakpoint is set automatically.

---

**Figure 5-15 Set Address/Data Break/Tracepoint dialog box**
In this dialog box:

- The Location field is prefilled (but you can change this if required).
- The Class field shows the default breakpoint supported by your debug target (a Standard breakpoint). You cannot change this.

The contents of this field change depending on your debugging environment and target configuration, see *RealView Debugger v1.7 User Guide* for details.

- By default, a software instruction breakpoint is highlighted. The available breakpoint types are defined by your debug target. For connections that use RVISS, you can choose from a range of supported breakpoint types.
- For certain breakpoint types, the Location, Value Match or Class fields might be unavailable. In this case, the field is grayed out and right-click menus are disabled.

5. Click **New** to specify the Qualifiers or conditions for the new breakpoint. This displays the **New Qualifiers** menu.

6. Select **When Expression True...** from the menu.

7. Enter the condition `Int_Glob==5` in the prompt.
   RealView Debugger requires that you enter the condition in C format.

8. Click **Set** to confirm the entry and close the prompt box.
   The breakpoint condition is shown in the Qualifiers group display.

9. Click **OK** to close the Set Address/Data Break/Tracepoint dialog box.
   The code view is updated and a yellow disc appears to show that you set a conditional breakpoint. The **Cmd** tab, in the Output pane, shows the RealView Debugger command used to set the breakpoint.

10. Click the **Go** button on the toolbar to execute the image. When the breakpoint is reached, execution stops and RealView Debugger displays a message in the **Cmd** tab of the Output pane to identify the point in the source code where the program stopped.

11. Click **Go** again.

If you try to set a breakpoint on a non-executable line, RealView Debugger looks for the first executable line immediately following and places the breakpoint there. If the lines preceding the breakpointed instruction are comments, declarations, or other non-executable code, they are marked with black, downward pointing arrows. Lines
marked in this way are regarded as part of the breakpoint. You cannot place two unconditional breakpoints on the same line, or on lines marked by the downward pointing arrows.

For full details on using the Set Address/Data Break/Tracepoint dialog box to specify your breakpoints, see the chapter describing working with breakpoints in *RealView Debugger v1.7 User Guide*.

**Managing breakpoints**

To display the current breakpoint details:

1. Select **View → Pane Views → Break/Tracepoints Pane** from the default Code window to display the Break/Tracepoints pane, shown in Figure 5-16.

![Figure 5-16 Break/Tracepoints pane](image)

This shows the conditional breakpoint you set previously, the address, and the RealView Debugger command used to create it. RealView Debugger has identified the appropriate class of breakpoint, based on your debug target.

The check box is selected to show that the breakpoint is enabled.

2. Right-click anywhere on the breakpoint entry, in the Break/Tracepoints pane, to display the **Break/Tracepoints** menu.

   From here, you can examine the breakpoint, edit or copy it, or disable it so that it is ignored the next time the program runs.

   You can remove entries from the Break/Tracepoints pane. Simply highlight the entry and select **Clear** from the **Break/Tracepoints** menu.

   The breakpoints list is refreshed after the entry has been removed.

**Using CLI commands**

You can use CLI commands to create and modify breakpoints, for example, to set the conditional breakpoint shown in Figure 5-16, type:

```
binstr,when:{Int_Glob==5} \DHRY_2\#81:6
```

It is not necessary to specify the breakpoint class.
For full details see the **BREAK** commands in *RealView Debugger v1.7 Command Line Reference Guide*. 
5.4 Working with custom panes

If you are working in single-processor debugging mode, the default Code window gives you full access to your code views and all debugging operations. However, you might want to display multiple memory views for your debug target, or to view multiple panes at the same time. You can customize your desktop to get the views you want.

See Chapter 7 RealView Debugger Desktop for:
- details of the contents of the Code window
- full details on working with panes
- a description of the controls used in the rest of this section.

This section describes:
- Customizing your debug views
- Hiding panes on page 5-34.

5.4.1 Customizing your debug views

To set up the default Code window and customize the panes:

1. Start RealView Debugger, as described in Starting RealView Debugger on page 4-2.
2. Connect to the ARM7TDMI core processor using RVISS, as described in Making a connection on page 4-5.
3. Select Project → Open Project... to open the required project, for example dhrystone.prj.
4. Click on the hyperlink in the File Editor pane to load the associated image. This location has been derived automatically from the project information.
5. Display memory contents starting at 0x8000, as described in Displaying memory contents on page 5-20.
6. Select View → Pane Views → Registers to view the Register pane, Core tab.
7. Click the Pane Content menu in the Register pane and select Switch Side to reposition the pane.
8. Select View → Pane Views → Call Stack to view the Call Stack pane, Call Stack tab.
9. Click the Pane Content menu in the Call Stack pane and select Process Control to change the debug view.
Your default Code window looks like the example shown in Figure 5-17.

![Figure 5-17 Example Code window](image)

In Figure 5-17, line numbering is turned on by selecting Edit → Editing Controls → Show Line Numbers.

### 5.4.2 Hiding panes

You can use the Pane Management controls, on the Actions toolbar, to define which panes are visible in your Code windows, for example click Show/Hide Bottom Pane from the Pane Management controls to hide the Output pane. Click the same button again to restore the pane view.

Click the Pane Content menu in the Register pane and select Float to float the Register pane above the default Code window. Click the Pane Content menu again and select Hide to hide the Register pane.

To restore the Code window:

1. Select View → Pane Views → Registers to view the floating Register pane.
2. Click the Pane Content menu in the Register pane and select Dock to reinstate the floating pane.
5.5 More about projects

This section gives more detail on how to manage your projects in RealView Debugger and enables you to make more changes to the tutorial project, if required. It contains the following sections:

- Upgrading projects
- Customizing your build tools on page 5-37
- Adding files to your project on page 5-39
- Generated makefiles on page 5-39
- Building an application on page 5-40
- Finding build errors on page 5-41.

5.5.1 Upgrading projects

RealView Debugger determines the location of your ARM build tools automatically. If you open a project and a later version of the toolchain exists, RealView Debugger prompts you to upgrade the project. The way in which this happens depends on whether or not the toolchain with which the project was built is installed on your system.

When you open a project, RealView Debugger examines it to find out which toolchain was used to create it. At this stage, the project becomes the active project, but remains in the validation stage instead of being fully opened. RealView Debugger then performs one of the following actions:

- If the original toolchain is no longer present on your system and an upgrade is available for the project, you must upgrade the project before you can use it. RealView Debugger displays the Upgrade Project selection box, shown in Figure 5-18.

![Figure 5-18 Upgrade Project selection box](image-url)
The name of the project you are about to upgrade appears at the top of the selection box and the display list shows available upgrades. Do one of the following:

— If you want to upgrade the project, select the required conversion and click **OK**. The project is upgraded and RealView Debugger completes the validation process. The project is then open and ready for use. A backup copy of the old project settings file is created automatically in the same location.

— If you do not want to upgrade the project, click **Cancel**. The validation process is not completed and the project is closed.

• If the original toolchain is present and an upgrade is available for the project, you can use the project whether or not you upgrade it. RealView Debugger therefore offers you the appropriate binding options before prompting you to upgrade the project (see the chapter on Project Binding in the *RealView Debugger v1.7 Project Management User Guide* for information on binding).

RealView Debugger then displays the Upgrade Project ToolChain dialog box, shown in Figure 5-19.

![Figure 5-19 Upgrade Project ToolChain dialog box](image)

Do one of the following:

— Click **Yes** if you want to upgrade your project. This displays the Upgrade Project selection box, shown in Figure 5-18 on page 5-35. A backup copy of the old project settings file is created automatically in the same location.

— Click **No** if you do not want to upgrade your project now. In this case, RealView Debugger remembers the state of the **Don’t ask me again for this project** check box. If you want to upgrade the project later, then leave this check box unchecked.

RealView Debugger completes the validation process and the project is open and ready for use.
Quick-start Tutorial

—– Note —–

If you select Don’t ask me again for this project, and you want to upgrade that project later, you can use the Project Control dialog box to upgrade the project manually. See the chapter describing managing your projects in RealView Debugger v1.7 Project Management User Guide for details.

— Click Cancel if you do not want to upgrade your project.

RealView Debugger completes the validation process and the project is open and ready for use.

• If the original toolchain is not present and an upgrade is not available for the project, you cannot use the project and an error message is displayed.

When a project is successfully upgraded, the dialog box is not displayed when you next open that project, unless you upgrade your build tools again.

—– Note —–

If you upgrade a project for which you do not have the tools installed, you must force the binding behavior. To do this, either close the project and reopen it, or use the Project Control dialog box.

—– Note —–

It is recommended that you rebuild your project after upgrading it.

5.5.2 Customizing your build tools

You can specify the location of build tools to use when building applications and images. You can set up a tools location at any time but it is recommended that you do this before you create your first project.

To customize your build tools:

1. Select Project → Build-Tool Properties... from the Code window main menu. This displays the Build-Tool Properties window shown in Figure 5-20 on page 5-38.
This shows a Custom installation where support for both ARM and DSPG toolchains is installed. If you have installed a different configuration your window looks different.

2. Double-click on a group, in the left pane, to specify the tools. This displays the contents in the right pane. This is shown in the example in Figure 5-20.

3. Right-click on the Tool_dir entry and select Edit as Directory Name from the context menu. This displays the Enter New Directory dialog box where you can specify the location of the tools you want to use. This forms the base directory for the executable tools for this family of processors.

4. Select File → Save and Close to save your changes and close the Build-Tool Properties window.

If you have a common linker command file or your own runtime libraries, for example kernel, ASIC (Application Specific Integrated Circuit), and peripheral drivers, you can specify them for all projects at this stage. To make the change:

1. Select Project → Build-Tool Properties... from the Code window main menu.

2. Double-click on the required group, in the left pane, to expand the tree.

3. Double-click on C_code in the right pane.

4. Right-click on Def_cmd_file and select Edit as Filename from the context menu. This displays the Enter New Filename dialog box where you can specify the location of the linker command file.
5. Right-click on `Def_lib_cmd_file` and select **Edit as Filename** from the context menu. This displays the Enter New Filename dialog box where you can specify the location of the librarian command file.

6. Select **File → Save and Close** to save your changes and close the Build-Tool Properties window.

### 5.5.3 Adding files to your project

You can change your project properties at any time after creation, for example you can add source files, add object files, delete files, or exclude files from the build.

#### Adding source files to a project

With a project open, you can add source files and so update the project properties automatically. There are two ways to do this, depending on the location of the file you want to add:

- Select **Project → Add This File to Project** to add the file currently displayed in the topmost tab in the File Editor pane to the list of project sources.
- Select **Project → Add Files to Project...** to display the Select File(s) to Add dialog box where one or more files can be added to the project.

--- Note ---

Ensure that the Project Properties window is not open when adding source files from the **Project** menu. Also ensure that any rebuild is complete.

---

#### Adding object files to a project

If you want to add object files to your project you must use the Project Properties window, shown in Figure 5-4 on page 5-7. For full details of how to do this, see the chapter describing customizing projects in *RealView Debugger v1.7 Project Management User Guide*.

### 5.5.4 Generated makefiles

RealView Debugger uses the template `gen_***.mk` to generate makefiles for all projects. The *** is replaced depending on the project toolchain. In this tutorial, RealView Debugger uses the template `gen_arx.mk` located in the default settings directory `\etc`.

The makefiles are generated when you do any of the following:

- create a new project
Quick-start Tutorial

- edit and save an existing project
- edit and close the Build-Tool Properties window
- update the Project Properties window, for example doing an edit and save.

Details about the makefile generation process are given in the Build tab of the Output pane.

Note
You can also force the makefiles to be regenerated if necessary. This is explained in the chapter describing working with projects in RealView Debugger v1.7 Project Management User Guide.

Viewing the makefile

You can view the makefile that is generated using RealView Debugger, for example either:

- Select File → Open... from the default Code window main menu to display the Select File to Open dialog box where you can locate the makefile and open it in the File Editor pane for viewing.
- Open Windows Explorer and navigate to the project base directory. Drag the makefile and drop it into the File Editor pane where it opens for viewing.

Do not make any changes to the generated makefiles as these are overridden when the files are next generated. It is recommended that you always use the Project Properties window to set up your preferences.

5.5.5 Building an application

To build the executable for a project:

1. Select Project → Open Project... to open the required project.
2. Select Tools → Build... from the default Code window main menu.
3. If you have made any changes to the Project Properties, or to the Build-Tool Properties, you are prompted to rebuild all project files. Click Yes to confirm the rebuild.

The build, or rebuild, completes and RealView Debugger displays the Build tab, in the Output pane, to report successful completion. If the build process fails, the Build tab reports any errors.
In the build process:

- RealView Debugger builds the executable as a background task. This is shown by an exclamation mark in the Build tab. You can still submit commands or use RealView Debugger during the build.
- The Debug build target configuration is built by default.
- The -f flag forces the make utility to read the specified file, tutorial_Debug.mk, as a makefile.
- The all flag forces a rebuild of all project files.

You can click the Stop Build button on the Actions toolbar to halt a build.

### 5.5.6 Finding build errors

Errors in your source code always result in a failed build and the executable is not built. In these cases, you must locate and correct these errors. RealView Debugger includes features to help locate errors in the source files when the build process fails.

#### Build error reporting

The error reporting system:

- highlights the error text in the Output pane Build tab
- scrolls the file in the File Editor pane to the location of the error
- places a blue pointer at the error
- places the flashing text cursor in the source file after the error.

If RealView Debugger finds more than one error, it highlights the first error. Select Tools → Next Line/Error to move through the error list after correcting an error. Alternatively, you can go directly to an error by double-clicking on the line number, shown in the Output pane Build tab.
5.6 Completing the tutorial

Because the tutorial project is self-contained, follow these steps if you want to tidy up the install_directory\RVDS\Examples\... directory after you close down RealView Debugger.

5.6.1 Removing tutorial project files

To remove the files associated with the project:

1. Delete the project file \Tutorial\Tutorial.prj, in the install_directory\RVDS\Examples\... directory.

2. Delete the build target configuration directories, for example:
   \Tutorial\Debug

3. Delete the target generated makefiles, for example:
   \Tutorial\Tutorial_Debug.mk

4. Delete any backup files with the .bak extension.

The project is now deleted from your workstation.

5.6.2 Removing the tutorial project

If you do not want to keep the files copied at the start, or the executable image, you can delete the entire tutorial project base directory, that is \Tutorial.
Chapter 6
Ending your RealView Debugger Session

This chapter describes how to end your debugging session and how to exit RealView®
Debugger. It contains the following sections:
• Saving the session on page 6-2
• Disconnecting from a target on page 6-5
• Exiting RealView Debugger on page 6-8.
6.1 Saving the session

RealView Debugger stores session details by default when you end your debugging session. Saving the session enables you to start your next session using the same working environment, connecting automatically to a specified target, and opening projects.

RealView Debugger stores session details using your:

- Workspace
- Startup file on page 6-3
- History file on page 6-3
- Settings options on page 6-4.

6.1.1 Workspace

The RealView Debugger workspace is used for visualization and control of default values, and storing persistence information. It includes:

- user-defined options and settings
- connection details
- details about open windows and, in some cases, their contents
- user-defined projects or auto-projects to open on startup.

The first time you run RealView Debugger, the default workspace settings file rvdebug.aws is created in your home directory. Each time you start RealView Debugger after this, the debugger loads this workspace automatically, but you can change the defaults or create a new workspace of your own.

To change your workspace, or close your workspace so that RealView Debugger runs without a workspace, select File → Workspace from the main menu to display the Workspace menu shown in Figure 6-1.

Alternatively, you can run RealView Debugger from the command line without loading a workspace, for example:

rvdebug.exe -aws=--
For full details on amending your workspace settings to define how session details are saved and where, see the chapter describing configuring workspaces in *RealView Debugger v1.7 User Guide*.

### 6.1.2 Startup file

The startup file contains a record of your last debugging session including:

- images and files loaded into RealView Debugger
- the list of all recently loaded files, for example source files
- the recent workspaces list
- the workspace to be used on startup, if specified
- workspace save and restart settings
- user-defined menu settings, for example pane format options
- the recent projects list.

By default, this file is called `rvdebug.sav`. The first time you close down RealView Debugger after performing an operation in the default Code window, a startup file is created in your home directory. From this point on, every time you close down RealView Debugger and exit, this startup file is updated. You can specify a different startup file, or none at all, by changing your workspace settings.

### 6.1.3 History file

The history file contains a record of:

- commands submitted during a debugging session, for example, changing directory, loading source files, loading an image for execution, or setting breakpoints

Breakpoints are only added to the history list if they are set using breakpoint dialog boxes, for example the Set Address/Data Break/Tracepoint or the Simple Break if X dialog box. If you set a breakpoint in another way, for example using a CLI command, this is not added to the list. See the chapter describing working with breakpoints in *RealView Debugger v1.7 User Guide* for more details.

- data entries examined in the Code window during debugging
- the Set Directory and Set File lists used in Open dialog boxes, for example Select File to Open, or Select File to Include Commands from
- up to 32 personal favorites such as variables, data values, watches, and breakpoints.
The first time you run RealView Debugger, carry out an operation (for example open a file or load an image) and then exit, a history file is created in your home directory. By default, this is called exphist.sav and is updated at the end of all subsequent debugging sessions.

**Note**

If you are using RealView Debugger on non-Windows platforms, the history file is only created if you create and save a favorite, for example a breakpoint or watchpoint. See Appendix B *RealView Debugger for Sun Solaris and Red Hat Linux* in *RealView Debugger v1.7 User Guide* for details.

### 6.1.4 Settings options

By default, RealView Debugger:

- writes your current settings to your startup file ready for re-use
- saves your current workspace
- reloads this workspace at the next start-up.

To change these defaults, select **File → Workspace** from the main menu to display the **Workspace** menu shown in Figure 6-1 on page 6-2.

At the bottom of this menu are your current settings options. Enable or disable the required options to decide how your next RealView Debugger session starts. For example, if **Same Workspace on Startup** is unselected, RealView Debugger does not use the current workspace at the next start-up. This means that, unless another workspace is specified on the command line, RealView Debugger opens the default workspace.

These options are saved in your startup file when you exit RealView Debugger.
6.2 Disconnecting from a target

You can disconnect from the current target quickly, depending on your current debugging mode, for example:

- Select **File → Connection → Disconnect** from the Code window main menu. This immediately disconnects the current connection.

- Use the **Disconnect** button from the Connection group on the Actions toolbar. This immediately disconnects the current connection.

- Select **File → Connection → Connect to Target...** from the Code window main menu to display the Connection Control window, and unselect the check box to disconnect the connection.

You can also click the **Connection Control** button, in the Connection group on the Actions toolbar, to display the Connection Control window quickly. If the window is hidden by other windows, click the button twice.

RealView Debugger Code windows do not close when you disconnect from a target. However, if you have an image loaded, disconnecting removes all the debug information from RealView Debugger and this clears pane contents.

Disconnecting does not close any open projects.

This section describes:
- Setting disconnect mode
- Viewing target status on page 6-6.

**Note**

It is not necessary to disconnect from a target before you close down RealView Debugger. See **Exiting RealView Debugger** on page 6-8 for more details.

6.2.1 Setting disconnect mode

You can control the way a target processor is left after you disconnect. This is particularly useful when debugging multiprocessor debug targets and working with multiple threads. In single processor debugging mode, you might want to leave an image executing but close down RealView Debugger.

If you are connected, you can set disconnect mode:

1. Select **File → Connection → Disconnect (Defining Mode)...**
2. Select the required state from the Disconnect Mode selection box.
The options listed depend on your target vehicle, for example, when connected using RealView ARMulator® ISS, the only disconnect option available is As-is with Debug.

3. Click OK to disconnect and leave the processor in the required state.

--- Note ---
If you click Cancel, the disconnection continues using the default disconnect mode as defined by the target vehicle.

If you are working in multiprocessor debugging mode you can specify a disconnect mode for any of your active connections:

1. Select File → Connection → Connect to Target..., or click the Connection Control button, to display the Connection Control window.

2. Right-click on the connection entry and select Disconnect (Defining Mode)... from the Disconnection context menu.

3. Select the required state from the selection box.
   The options listed depend on your target vehicle.

4. Click OK to disconnect and leave the processor in the required state.

--- Note ---
If you click Cancel, the disconnection continues using the default disconnect mode as defined by the target vehicle.

If you set disconnect mode from the Connection Control window, this temporarily overrides any setting in your target configuration file.

See the chapter describing connecting (and disconnecting) in RealView Debugger v1.7 Target Configuration Guide for full details of disconnect mode and how to specify this setting for your debug target.

### 6.2.2 Viewing target status

The State group, on the Actions toolbar, enables you to see the current state of your debug target, for example:

**Unknown** Shows that the current state of the target is unknown to the debugger. For example it might have been running when the disconnect took place or it might be stopped.
For more details of the State group see *Viewing target status* on page 4-17.
6.3 Exiting RealView Debugger

This section describes the options available when you exit RealView Debugger:

- Closing down without a connection
- Closing down with a connection
- Reconnecting to a target on page 6-9.

6.3.1 Closing down without a connection

Disconnect from your target, as described in Disconnecting from a target on page 6-5, and select File → Exit to display the Exit dialog box.

Click Yes to close the Exit dialog box and close down RealView Debugger.

6.3.2 Closing down with a connection

If you are connected to your target and you select File → Exit, the Exit dialog box includes the Disconnect Connection check box, shown in Figure 6-2.

![Exit dialog box, connected](image)

Use the check box to specify your disconnect options:

- Maintaining the connection
- Terminating the connection on page 6-9.

**Maintaining the connection**

To close down and maintain the connection:

1. Unselect the Disconnect Connection check box shown in Figure 6-2.
2. Click Yes to close the Exit dialog box and close down RealView Debugger.

RealView Debugger exits but the connection is maintained for future sessions, that is the Target Vehicle Server (TVS) is left running. If you are connected to a remote debug target and you close down this way, the remote RealView Connection Broker, RealView Network Broker, is also left running.
Terminating the connection

To close down and terminate the connection:

1. Leave the Disconnect Connection check box, shown in Figure 6-2 on page 6-8, so that it is selected.

2. Click Yes to close the Exit dialog box and close down RealView Debugger.

RealView Debugger exits and terminates any connections to debug targets, that is TVS stops. If you are connected to a remote debug target and you close down this way, you must stop the remote RealView Connection Broker, RealView Network Broker, manually.

Note

See Components of RealView Debugger on page 2-4 for details of TVS and RealView Connection Broker.

6.3.3 Reconnecting to a target

RealView Debugger saves all open connections in the current workspace if you close down without disconnecting first. This default behavior is independent of the status of the Disconnect Connection check box shown in Figure 6-2 on page 6-8.

This means that, if you exit a session when still connected to a debug target, RealView Debugger tries to reconnect when you next open the workspace. However, this might fail if the connection or vehicle status has changed, for example if your RealView ICE interface has been disconnected or if your Multi-ICE® server has stopped.

If RealView Debugger reconnects successfully, the connection mode specified in the target configuration file is used by default. See the chapter describing connecting in RealView Debugger v1.7 Target Configuration Guide for full details on defining the connect mode for your debug target.

Note

Use your settings options to change the way RealView Debugger closes down, see Settings options on page 6-4 for details.
Ending your RealView Debugger Session
Chapter 7
RealView Debugger Desktop

This chapter describes, in detail, the RealView® Debugger desktop. It describes the contents of the default Code window, and explains how to change them. This chapter describes items and options available from the main menu and the toolbars.

It contains the following sections:

- About the desktop on page 7-2
- Finding options on the main menu on page 7-12
- Working with button toolbars on page 7-16
- Working in the Code window on page 7-20
- Editor window on page 7-23
- Resource Viewer window on page 7-24
- Analysis window on page 7-28.
7.1 About the desktop

This section describes the default Windows desktop that you see when you run RealView Debugger for the first time after installation and highlights any key features that might be different. It contains the following sections:

- Splash screen
- Code window on page 7-3
- Default windows and panes on page 7-5
- Pane controls on page 7-8
- Button toolbars on page 7-9
- Color Box on page 7-10
- Other window elements on page 7-11.

Note

The RealView Debugger v1.7 User Guide also contains desktop-specific information for developers using RealView Debugger on non-Windows platforms. See Appendix B RealView Debugger for Sun Solaris and Red Hat Linux for details.

7.1.1 Splash screen

Run the debugger from the Windows Start menu to display the RealView Debugger splash screen. During this time, the debugger is checking your environment and creating (or updating) configuration files and a working directory.

Use the -nologo flag to run RealView Debugger from the command line without a splash screen.
7.1.2 Code window

The Windows splash screen is replaced by the default Code window when you run RealView Debugger for the first time after installation, shown in Figure 7-1.

The Code window is your main debugging and editing window. The contents of this window change as you:

- connect to targets
- load and unload application programs or files
- configure and customize your working environment.

The toolbar buttons displayed, and the options available from window and pane menus, also change depending on the licenses you have.
Title bar

The Code window title bar gives details of the connection and any processes running on your debug target. If you connect to a target and load an image, your title bar looks like the one shown in Figure 7-2.

![Figure 7-2 Code window title bar](image)

In addition to the application icon, the title bar contains (from left to right):

RVDEBUG Identifies the Code window. This changes as you open new windows, for example RVDEBUG_1, or RVDEBUG_2.

dhrystone The project associated with the image that you loaded.

In RealView Debugger, a project can be associated with a connection, that is it is *bound* to that connection, shown in the example in Figure 7-2.

Where a project is not associated with a particular connection, it is *unbound*. In this case, the project name is enclosed in angled brackets, for example `<my_project>`.

See the chapter describing project binding in *RealView Debugger v1.7 Project Management User Guide* for more details.

@SimARM_1:Sim The connection, including the connection number and the target vehicle.

[Unattached] If you are working in multiprocessor debugging mode, this shows the windows attachment, that is whether the window is attached to a connection:

- In this mode, a Code window is unattached by default, shown by [Unattached].
- You can attach a Code window to a specified connection, shown by [Board].

See the chapter describing working with multiple targets in *RealView Debugger v1.7 Extensions User Guide* for details.

If you are using an RTOS, you can attach a Code window to a thread to display debug information for that thread. See the chapter describing RTOS support in *RealView Debugger v1.7 Extensions User Guide* for details.
In single-processor debugging mode, this part of the title bar is blank and the option to attach windows to your connection is not available.

If you float a pane, the pane title bar reflects the title bar of the calling Code window.

--- Note ---
The contents of your title bar might be different from the one shown in Figure 7-2 on page 7-4 depending on your connection (if any), open projects, and windows attachment.

### 7.1.3 Default windows and panes

RealView Debugger provides a range of debug views accessible through panes:

- Registers
- Stack
- Process Control
- Symbol Browser
- Watch
- Call Stack and Locals
- Memory and Memory Map
- Threads (with the appropriate RTOS extension)
- Break/Tracepoints.

The default Code window, shown in Figure 7-1 on page 7-3, contains:

**File Editor pane**

The File Editor pane is always visible when working with RealView Debugger.

Use this area of the Code window to:

- use a shortcut to connect to a target or load an image.
- enter text to create project files
- open source files for editing and resaving
- view disassembly
- set breakpoints to control execution
- use the available menu options to search for specific text as part of debugging
- follow execution through a sequence of source-level and disassembly-level views
The File Editor pane contains a hyperlink to make your first connection to a debug target. When a connection is made, this link changes to give you a quick way to load an image.

When RealView Debugger first starts, the File Editor pane contains tabs to track program execution:

- the **Src** tab shows the current context in the source view
- the **Dsm** tab displays disassembled code with intermixed C/C++ source lines.

If you load an image, or when you are working with source files, more tabs are displayed, for example `dhry_1.c`. In this case, click on the **Src** tab to see the location of the PC.

See the chapter describing editing source code in *RealView Debugger v1.7 Project Management User Guide* for full details on using the editing facilities in RealView Debugger.

**Side pane**

By default, this contains the Register pane that displays grouped processor registers for the current target processor. When you first run RealView Debugger, this pane is positioned to the right of the File Editor pane but you can switch this pane to the left, using the **Switch Side** option from the **Pane Content** menu.

The Register pane displays different tabs depending on the target processor and the connection mechanism. For example, you might be using *RealView ARMulator*® **ISS** (RVISS) to simulate an ARM® processor. If you are connected to RVISS through the RealView Connection Broker interface, a **CycleCount** tab is available that displays debugger internals and statistics. However, if the target processor is a DSP, a **Status** tab is available displaying details of the status flags.

**Middle pane row**

The middle pane row can contain one, two, or three pane views, but you can specify which panes are visible. By default this row contains:

**Call Stack pane**

Use this pane to:

- display the procedure calling chain from the entry point to the current procedure
- monitor local variables.

The Call Stack pane contains tabs:

**Call Stack**

Displays the stack functions call chain.
Locals  Shows variables local to the current function.
Statics  Displays a list of static variables local to the current module.
This  Shows objects located by the C++ specific this pointer.

Watch pane
Use this pane to:
• set up variables or expressions to watch
• display current watches
• modify watches already set
• delete existing watches.
The Watch pane contains tabs to display sets of watched values. The first tab, Watch1, is selected by default.

Memory pane
Use this pane to:
• display the contents of a range of memory locations on the target
• edit the values stored by the application.

Bottom pane
The bottom pane of the Code window always contains the Output pane. Select the different tabs to:
• enter commands during a debugging session (Cmd)
• handle I/O with your application (StdIO)
• see the progress of builds (Build)
• see the results of Find in Files operations (FileFind)
• see the results of operations using your version control tool (SrcCtrl)
• view the results of commands and track events during debugging (Log).
The command line is located at the bottom of the Output pane. This shows the status of the current process, for example, Stop, Run, or None (no process). You can also enter debugger commands at the > prompt.
Changing panes

Select View → Pane Views to display the Pane Views menu where you can change what is displayed in a pane view or to restore the default views. You can also use this menu to display any window or pane that is hidden, without changing the view it displays. If a pane is hidden and you use the Pane Views menu to change the pane contents, it is automatically displayed to show the new view.

During your debugging session, you can use the Pane Content menu to define which view is displayed in a chosen pane (see Pane controls for details). At the end of your first session, you can configure RealView Debugger to start next time with the current set of panes or views by saving the customized setup in your workspace.

Formatting pane views

The contents of the File Editor pane and the Output pane cannot be changed, for example you cannot substitute a Memory pane for the Output pane. However, you can define how the view is formatted, for example you can change the size of text displayed in the File Editor pane or the number of lines displayed in the Output pane. See the chapter describing configuring workspaces in RealView Debugger v1.7 User Guide for details of how to do this.

7.1.4 Pane controls

Each configurable pane in the Code window, shown in Figure 7-1 on page 7-3, includes a title bar and pane controls. In the side pane, the pane title bar is displayed horizontally at the top of the pane. In the middle and bottom panes, the title bar is displayed vertically at the left side of the pane.

When you float a pane, the title bar is displayed horizontally at the top of the window.

A pane contains the controls:

Pane Content

Click this button to display the Pane Content menu where you can change the debug view in the pane.

The selected option in the menu indicates the current view.

Visual controls

The visual controls are at the bottom of the Pane Content menu. Use these to float or hide the pane.

In the side pane, use the option Switch Side to move the pane from the right side of the Code window so that it is positioned to the left of the code view.
Expand/Collapse Pane

Click this button to collapse the currently selected view. Other panes are expanded to fill the empty area.

Click the Expand Pane button to restore the view.

There is no option to collapse, or expand, a floating pane.

Pane Menu

Click this button to display the Pane menu.

Use this to:
- change the display format
- change how pane contents are updated
- extract data from the pane.

The options available from this menu depend on the pane.

7.1.5 Button toolbars

Below the Code window main menu, there are two toolbars that provide quick access to many of the features available from menu options:
- Actions toolbar, shown in Figure 7-3
- Editing toolbar, shown in Figure 7-4.

To disable a toolbar, select View → Toolbars from the main menu. This displays the Toolbars menu where you can specify which toolbars are visible.

You can move a toolbar from the default position in the Code window so that it floats on your desktop, shown in Figure 7-3 and Figure 7-4. To restore the floating toolbar, double-click anywhere on the toolbar title bar.
RealView Debugger Desktop

--- Note ---
Repositioning a toolbar in this way applies only to the calling Code window. If you open a new Code window the toolbars are in the default positions.

---

See Working with button toolbars on page 7-16 for details of the buttons available from the Actions toolbar and the Editing toolbar, and how to customize toolbar groups.

### 7.1.6 Color Box

Code windows in RealView Debugger are color-coded to help with navigation. This is particularly useful when debugging multiprocessor debug targets and working with multithreaded applications.

The Color Box, shown in Figure 7-5, identifies the connection associated with the current window.

![Color Box](image)

**Figure 7-5 Color Box**

When you first start RealView Debugger the Code window is not associated with any target or process. The Color Box changes when you make your first connection. As you create new Code windows, these are also color-coded.

Closing your connection changes the Color Box to show that there is no connection associated with the window. Any other Code windows attached to that connection are also updated to match.

When you are working with the Color Box, notice that:
- connection-independent windows, or controls, do not contain a Color Box
- floating panes contain a Color Box that matches the calling window.
7.1.7 Other window elements

There are status display areas at the bottom of the Code window, shown in Figure 7-1 on page 7-3:

**Status line** As you move through menu options, or when you view button tooltips, this line shows a more detailed explanation of the option or button under the cursor.

**Cursor location field**

As you move through files within the File Editor pane, the current location of the text insertion point is displayed in the Cursor location field at the bottom right of the Code window.

**LOG** If this appears to the right of the Cursor location field, this shows that output is being written to a log file.

**JOU** If this appears to the right of the Cursor location field, this shows that output is being written to a journal file.

**STDOUTlog** If this appears to the right of the Cursor location field, this shows that output is being written to an STDOUTlog file.

**NUM** Shows that you can use the numeric keypad to enter numbers.

**CAP** Shows that Caps Lock is selected.
7.2 Finding options on the main menu

This section provides a summary of the main menu options available from the Code window that enable you to:

- open and close files within the File Editor pane
- manage target images, projects, and workspaces
- navigate, search, and edit source files
- manage new windows and change pane contents
- work with projects and build images
- debug your images
- access the online help system.

The menus available from the main menu bar are:

File          Displays the File menu shown in Figure 7-6.

![Figure 7-6 File menu]

When you are using the File menu:

- The menu option Close Window is not enabled when the default Code window is the only window. This is the main debugging and editing window, and it must be open throughout your debugging session. Close the Code window to close down RealView Debugger.
- Some menu options are not enabled unless you have suitable support, for example the Thread option.
Edit

Displays the **Edit** menu shown in Figure 7-7. This menu enables you to work with source files as you develop your application. It includes options to define how source code is displayed in the File Editor pane.

<table>
<thead>
<tr>
<th>Command</th>
<th>Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undo</td>
<td>Ctrl+Z</td>
</tr>
<tr>
<td>Redo</td>
<td>Ctrl+R</td>
</tr>
<tr>
<td>Cut</td>
<td>Ctrl+X</td>
</tr>
<tr>
<td>Copy</td>
<td>Ctrl+C</td>
</tr>
<tr>
<td>Paste</td>
<td>Ctrl+V</td>
</tr>
<tr>
<td>Delete</td>
<td>Del</td>
</tr>
<tr>
<td>Select All</td>
<td>Ctrl+A</td>
</tr>
<tr>
<td>Find</td>
<td>F3</td>
</tr>
<tr>
<td>Find Next</td>
<td>Del+F3</td>
</tr>
<tr>
<td>Find Previous</td>
<td>Del+F3</td>
</tr>
<tr>
<td>Find in Files</td>
<td>Alt+F3</td>
</tr>
<tr>
<td>Jump to Function</td>
<td>Ctrl-Shift+F12</td>
</tr>
<tr>
<td>Jump to Function/Include file</td>
<td>Del+Shift+F12</td>
</tr>
<tr>
<td>Jump Back</td>
<td>Del+F12</td>
</tr>
<tr>
<td>Find Matching</td>
<td>Ctrl+F12</td>
</tr>
<tr>
<td>Where Am I...</td>
<td>Alt+Shift+F12</td>
</tr>
<tr>
<td>Show Insert Cursor</td>
<td>Del+F12</td>
</tr>
<tr>
<td>Show Last Changed Line</td>
<td>Del+F12</td>
</tr>
<tr>
<td>Module/File List</td>
<td>Alt+F12</td>
</tr>
<tr>
<td>Function List</td>
<td>Alt+Shift+F12</td>
</tr>
<tr>
<td>Variable List</td>
<td>Alt+Shift+F12</td>
</tr>
</tbody>
</table>

Figure 7-7 Edit menu

Find

Displays the **Find** menu shown in Figure 7-8. This menu enables you to work with source files and perform searches on those files as you debug your image. This also gives access to the browsers in RealView Debugger.

<table>
<thead>
<tr>
<th>Command</th>
<th>Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find</td>
<td>F3</td>
</tr>
<tr>
<td>Find Next</td>
<td>Del+F3</td>
</tr>
<tr>
<td>Find Previous</td>
<td>Del+F3</td>
</tr>
<tr>
<td>Find in Files</td>
<td>Alt+F3</td>
</tr>
<tr>
<td>Jump to Function</td>
<td>Ctrl-Shift+F12</td>
</tr>
<tr>
<td>Jump to Function/Include file</td>
<td>Del+Shift+F12</td>
</tr>
<tr>
<td>Jump Back</td>
<td>Del+F12</td>
</tr>
<tr>
<td>Find Matching</td>
<td>Ctrl+F12</td>
</tr>
<tr>
<td>Where Am I...</td>
<td>Alt+Shift+F12</td>
</tr>
<tr>
<td>Show Insert Cursor</td>
<td>Del+F12</td>
</tr>
<tr>
<td>Show Last Changed Line</td>
<td>Del+F12</td>
</tr>
<tr>
<td>Module/File List</td>
<td>Alt+F12</td>
</tr>
<tr>
<td>Function List</td>
<td>Alt+Shift+F12</td>
</tr>
<tr>
<td>Variable List</td>
<td>Alt+Shift+F12</td>
</tr>
</tbody>
</table>

Figure 7-8 Find menu

View

Displays the **View** menu shown in Figure 7-9. This menu enables you to set up new windows and panes as you are working with target connections. It also includes options to customize toolbar groups.

<table>
<thead>
<tr>
<th>Command</th>
<th>Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Code Window</td>
<td>Alt+1</td>
</tr>
<tr>
<td>Breakpoints Window</td>
<td>Alt+2</td>
</tr>
<tr>
<td>Resource Viewer Window</td>
<td>Alt+3</td>
</tr>
<tr>
<td>Editor Window</td>
<td>Alt+4</td>
</tr>
<tr>
<td>Custom Windows</td>
<td></td>
</tr>
<tr>
<td>Toolbars</td>
<td></td>
</tr>
<tr>
<td>Panes Views</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7-9 View menu
Note

Custom windows are not available in this release.

Project

Displays the Project menu shown in Figure 7-10. This menu enables you to work with projects so that you can organize your source files, model your build process, and share files with other developers. This can be used in conjunction with the Tools menu to rebuild images.

These features are described in full in the chapter describing working with projects in RealView Debugger v1.7 Project Management User Guide.

Tools

Displays the Tools menu shown in Figure 7-11.

This menu enables you to build files, or groups of files, to create your image ready for loading to a target. For details of this menu see the chapter describing building applications in RealView Debugger v1.7 Project Management User Guide.

The Tools menu also enables you to examine, and change, your workspace settings and global configuration options.
The **Tools** menu also provides access to options for Trace, Analysis, and Profiling. These features are described in full in *RealView Debugger v1.7 Extensions User Guide*.

---

**Note**

Supported by selected simulators from the DSP Group, the **Simulation Control** option is not available in this release.

---

**Debug**

Displays the **Debug** menu shown in Figure 7-12.

![Figure 7-12 Debug menu](image)

This menu includes the main facilities you use during a debugging session.

**Help**

Displays the **Help** menu shown in Figure 7-13.

![Figure 7-13 Help menu](image)

This menu gives you access to the RealView Debugger online help, to web downloads pages, and displays details of your version of RealView Debugger. You can also use this menu to create and submit a **Software Problem Report** (SPR).
7.3 Working with button toolbars

The Code window toolbars give access to many of the features available from the main menu and to additional debugging controls. By default, the Code window shows both toolbars but you can customize which controls are available.

When working with button toolbars, use Windows Tooltips to see hover-style help when you hold your mouse pointer over a button.

This section describes:
- Actions toolbar
- Editing toolbar on page 7-18
- Customizing the Actions toolbar on page 7-19.

7.3.1 Actions toolbar

The Actions toolbar, see Figure 7-3 on page 7-9, contains buttons used during debugging sessions and when working with source code:

- **File group**
  Use these buttons when developing applications. They enable you to open files and to save changed files in the File Editor pane. These buttons replicate selected options from the File menu.

- **Edit group**
  Use these buttons to edit source files in the File Editor pane. They replicate selected options from the Edit menu.

- **Execution group**
  Use these buttons to control program execution, for example starting and stopping execution, and stepping. These buttons are enabled when an image has been loaded. They replicate selected options from the Debug menu.

- **Context controls**
  Use these buttons to move up and down the stack levels during program execution. These buttons are enabled when an image has been loaded.

- **Command cancel**
  Commands submitted to RealView Debugger are queued for execution. Click this button to cancel the last command entered onto the queue. This does not take effect until the previous command has completed.

- **Image load group**
  Use these buttons to control images. They replicate selected options from the File menu.
Connection group

Use these buttons to control target connections. They replicate selected options from the **File** menu.

Enabled during a multiprocessor debugging session, click the **Connection** button to connect to the next available active connection. Click on the drop-down arrow to display the list of active connections with the current connection marked with an asterisk. The list also shows if the Code window is attached to this connection by adding a check mark.

Thread button

Used during a multithreading debugging session, click this button to change to the next active thread. Click on the drop-down arrow to display the list of active threads where you can identify the current thread. The list also shows if the Code window is attached to this thread by adding a check mark.

This button is only enabled when an underlying operating system is supported.

Pane Management controls

Use these buttons to control the panes displayed in the Code window.

Build group

Use these buttons to control the build process during your debugging session. They replicate selected options from the **Tools** menu.

Processor State group

This field indicates the runtime state of the debug target. This contains:

**Unknown**

Shows that the target state is not known to the debugger.

**Stopped**

Shows that the target is connected but not active.

**Running**

Shows that an image is currently running. In this case, a running progress indicator is also included.
7.3.2 Editing toolbar

The Editing toolbar, shown in Figure 7-14, becomes active when you are editing a file in the File Editor pane.

![Editing toolbar](image)

**Figure 7-14 Editing toolbar**

The Editing toolbar contains:

**File:** This read-only field shows the name of the file currently displayed in the File Editor pane. If you have changed the file since loading or saving, an asterisk, *, is appended to the end of the filename. Hold your mouse pointer over the field to see the full pathname.

If you are working on several files in the File Editor pane, the File field shows the name displayed on the topmost file tab.

**Find:** This field enables you to perform a quick text search on the file currently displayed in the File Editor pane. Type the required string into the Find field and then press Enter. If you are working on several files in the File Editor pane, the search examines only the file in the topmost file tab. The search behavior is defined by the settings in the Find dialog box displayed from the Find menu.

Click on the drop-down arrow to display a list of recently used search strings.

**Line:** Use the Line number field to enter the number of the line where the text insertion point is to be moved.

If the Line number field is empty, click inside this field and press Enter to display the line number where the text insertion point is currently located. Click inside the field again and press Enter to scroll to this location.

**Source control button**

This button indicates the read/write status of the current file. Click this button to change the status of a file.

You can edit a file only if the Read-Write icon is displayed (shown here). Click on the drop-down arrow to access the source control commands (if enabled).
7.3.3 Customizing the Actions toolbar

To customize your Actions toolbar, select **View → Toolbars → Customize...** from the Code window main menu to display the selection box shown in Figure 7-15.

![Figure 7-15 Toolbar groups selection box](image)

Check boxes show which toolbar groups are currently enabled on the Actions toolbar.

If you want to disable a button group:

1. Click a selected check box to disable the associated group.
2. Click **OK** to close the selection box and return to the Code window.

The disabled groups have been removed from the Actions toolbar.

To re-enable button groups, display the selection box and click on the check boxes so that they are selected.

If you customize a toolbar, this persists to any new Code windows that you open.

--- **Note** ---

Custom toolbar groups are not available in this release.
7.4 Working in the Code window

This section describes how to work with the Code window:

- Floating, docking, and resizing windows and panes
- Changing the focus
- In-place editing on page 7-21
- Working with tabs on page 7-22
- Working with scroll bars on page 7-22.

7.4.1 Floating, docking, and resizing windows and panes

Panes are docked to default positions in the Code window when RealView Debugger starts in the default state. You can resize the middle pane row by dragging the upper boundary to the required height. Similarly, drag the left boundary to a new position to enlarge the side pane. You can enlarge the Output pane by dragging the upper boundary.

A pane is floating when it is displayed separately from the calling window and can be moved around the desktop. To float a pane either:

- select **Float** from the **Pane Content** menu
- double-click on the pane title bar.

A floating pane is still tied to the calling window. If, for example, you float a pane from the middle pane row and then click the **Show/Hide Middle Pane** control in the Code window, the floating pane is also hidden. To restore the pane, click the **Show/Hide Middle Pane** control again.

To dock a floating pane, select **Dock** from the **Pane Content** menu or double-click on the pane title bar.

7.4.2 Changing the focus

In RealView Debugger, the **focus** indicates the window, or pane, where the next keyboard input takes effect. Use a single left-click on the title bar to move the focus to the Code window, or the pane, where you want to work. You can also move between the File Editor pane and the Output pane using Ctrl+Tab or Shift+Ctrl+Tab.

When working with several Code windows, left-click inside a pane entry to change the focus. The pane title bar changes color to show that it now has the focus, and the title bar of the calling Code window is also highlighted.

If you switch to another Code window by clicking on the title bar, the focus moves to that window and the text insertion point is located inside the File Editor pane. If the context of the Code window is unknown, the text insertion point is located at the command-line prompt.
If you double left-click in a pane entry, for example on the contents of a register in the Register pane, this moves the focus to this pane and highlights the entry ready for editing.

If you right-click in a pane that does not have the focus, the focus does not move to this pane. This action does, however, highlight the chosen entry in the new pane. In this case, use the Code window title bar to see where the focus is currently located.

### 7.4.3 In-place editing

In-place editing enables you to change a stored value and to see the results of that change instantly. RealView Debugger offers in-place editing whenever possible. For example, if you are displaying the contents of memory or registers and you want to change a stored value:

1. Double-click in the value you want to change, or press Enter if the item is already selected. The value is enclosed in a box with the characters highlighted to show they are selected (pending deletion).

2. Either:
   - enter data to overwrite the highlighted content
   - press the left or right arrow keys to deselect the existing data and position the insertion point where you want to make a change.

3. Press Enter to store the new value in the selected location.

If you press Escape before you press Enter, any changes you have made in the highlighted field are ignored.

In-place editing is not suitable for:
- editing complex data where some prompting is helpful
- editing groups of related items
- selecting values from predefined lists.

In these cases, an appropriate dialog box is displayed.

---

**Note**

When using in-place editing, you must either complete the entry and press Enter, or press Escape to cancel the operation. If you move the focus to another pane, RealView Debugger suspends the current editing operation so that you can complete it when focus returns. Similarly, if you click inside another pane, or change the current view, RealView Debugger cancels the current editing operation.
7.4.4 Working with tabs

You can access RealView Debugger debugging features using tabbed pages or tabs. In the Watch pane, for example, there are multiple tabs and each might show a different set of watched variables. The Output pane contains tabs enabling you to select the view that suits your debugging task.

Right-click on a tab to display text that explains the function of the tab or the content. If you are using the default Windows display settings and you right-click on a tab that is not at the front, the tab name being referenced is colored red for easy identification.

If you right-click over a blank area of the tab bar at the bottom of a window or pane, a context menu enables you to select a tab from a list. You can also display this menu by right-clicking on the left, or right, scroll arrow on the left-hand side of the tab bar.

7.4.5 Working with scroll bars

If you are working in a pane where vertical scroll bars are enabled, right-click on the up, or down, scroll arrow to see the Scroll menu, shown in Figure 7-16.

![Scroll menu](image)

Use the Scroll menu to control your view of the pane contents.
7.5 Editor window

RealView Debugger includes a range of editing features to help you work with source code and projects. Use a standalone Editor window to access these editing features so that you can edit files independently of the debugging session or to include these files within your project.

To display a standalone Editor window, select View → Editor Window from the Code window main menu. This displays a floating Editor window shown in Figure 7-17.

![Figure 7-17 Editor window](image)

In this example, the window is already loaded with the file displayed in the topmost tab in the parent File Editor pane.

A quick way to display an empty Editor window is to click on the Dsm tab and then select View → Editor Window from the Code window main menu.

When working with standalone Editor windows, remember:
- You can display standalone Editor windows from the Tools menu.
- You cannot customize the Actions toolbar.
- You can float the Editing toolbar.
- Tab controls are available when you have multiple tabs.
- The vertical Scroll menu is available.
7.6 Resource Viewer window

The Resource Viewer gives access to all the debugger resources as your debugging session progresses. For example, in single-processor debugging mode, this enables you to view details about the current connection.

To display the Resource Viewer window shown in Figure 7-18, start RealView Debugger in the usual way, connect to your target, and select View → Resource Viewer Window from the Code window main menu.

![Figure 7-18 Resource Viewer window]

Note

The contents of your window might be different from the one shown in Figure 7-18 depending on your debugging mode, licenses, connection (if any), open projects, and windows attachment.

Because RealView Debugger is running, opening this window displays the starting state of the debugger. The contents of this window change as you establish new target connections, start applications, and debug your images.

This example shows a debugging session where a single connection has been made. With the RealView Debugger multiprocessor extension installed, multiprocessor debugging resources are available from the Conn tab (when more connections have been made).

With the appropriate RTOS extension, the Resource Viewer window gives access to RTOS resources, for example threads, timers, queues, event flags, and memory in byte and block pools (not shown in Figure 7-18).

When you display the Resource Viewer window, the title bar and the Color Box reflect the calling Code window.
The Resource Viewer window includes the:
- **File menu**
- **View menu**
- **RSD menu** on page 7-26
- **Help menu** on page 7-27
- **Resources toolbar** on page 7-27
- **Resources list** on page 7-27
- **Details area** on page 7-27.

To view your RTOS resources and data structures, you must first configure RTOS support and then execute your image. For full details on using the Resource Viewer window with a multithreaded application, and the RTOS-specific features of this window, see the chapter describing RTOS in *RealView Debugger v1.7 Extensions User Guide*.

### 7.6.1 File menu

This menu contains:

- **Close Window**
  Closes the Resource Viewer window.

### 7.6.2 View menu

This menu contains:

- **Update List** Updates the items displayed in the Resources list.
  If you click on the **Conn** tab, choose this option to reread the board file. This might be necessary if you change your connection without closing the Resource Viewer window.

- **Display Details**
  Displays details about a selected entry in the Resources list. A short description is shown in the Details area in the window.
  You can also display details about an item by double-clicking on the entry in the Resources list.

- **Display Details as Property**
  Select this option to display details information in a properties box.
  Select this option to change the default display format while the window is open.
Close the Resource Viewer window to restore the default, that is a description is shown in the Details area.

**Display List in Log Area**
Not available in this release.

**Clear Log**
Clears messages and information displayed in the Details area.

**Auto Update Details on Stop**
Automatically updates the Details area when any image running on the connection stops.
This gives you information about the state of the connection when the process terminated.
In multiprocessor debugging mode, this applies across all connections.

**Auto Update**
Automatically updates the Resources list as you change debugger resources. This takes effect when any image running on the connection stops. Selected by default.

**Auto Update on Timer...**
Not available in this release.

### 7.6.3 RSD menu

Use this menu to control Running System Debug (RSD):

**Enable/Disable RSD**
This option enables or disables RSD. The initial state depends on the RSD setting in the RTOS group in the target configuration file.

**Stop Target Processor**
Stops the target processor and suspends RSD. RealView Debugger switches to HSD mode.
Depending on your configuration settings, click the Go button to start execution and restart RSD.

**Properties**
Select this option to see details about the Debug Agent. This includes the status of the RSD module, configuration settings, and RSD breakpoints.

---
**Note**

If RSD is not enabled, some options are grayed out.
7.6.4 Help menu

Select Help from the Resource Viewer window menu bar to display the Help menu.

This menu gives you access to the RealView Debugger online help and displays details of your version of RealView Debugger. You can also use this menu to create and submit a problem report.

7.6.5 Resources toolbar

This toolbar contains three controls:

Connection button
Used during a multiprocessor debugging session, click this button to switch to the next available active connection. Changing the active connection updates the information shown in the Resource Viewer window.

Module: This field reports the OS module, for example the name of the RTOS.

Status: This field describes the current RSD status.

7.6.6 Resources list

The Resources list box displays all the resources available to RealView Debugger. In single-processor debugging mode, it contains only the Conn tab showing the connection.

In multiprocessor debugging mode, the Conn tab shows all active connections. An asterisk (*) indicates the current connection. See the chapter describing multiprocessor support in RealView Debugger v1.7 Extensions User Guide for details.

If you are debugging a multithreaded application, the Resources list box displays a series of RTOS-specific tabs, see the chapter describing RTOS support in RealView Debugger v1.7 Extensions User Guide for details.

7.6.7 Details area

The bottom half of the Resource Viewer window displays information about a chosen entry, for example a connection or, where applicable, a thread.
7.7 Analysis window

RealView Debugger includes Trace, Analysis and Profiling for use when debugging either single targets or multiple processes. If you are using Trace, the Analysis window gives you access to these features.

You can set tracepoints for a trace capture from the Code window and then use the Analysis window to view and analyze the results of the capture. The Analysis window provides access to most of the tracing and profiling functionality and enables you to view the captured trace information using the available tabs.

If you are connected to an appropriate debug target, for example an ETM-enabled ARM7TDMI® core, select View → Analysis Window from the Code window main menu to display the Analysis window, shown in Figure 7-19.

This example shows the results of a trace capture between two tracepoints during execution of the dhrystone.axf image as part of a multiprocessor debugging session. The Profile tab enables you to perform profiling analyses of the trace data. You can use this tab to analyze control-flow information, measure the time it takes to execute certain functions, and view call-graph data. Use the Profiling data menu to specify how profiling data is displayed and interpreted when you are using this tab. The Histogram column provides a graphical representation.

To see this display, you must first configure the Embedded Trace Macrocell™ (ETM), set up your trace options and then execute your image. For full details on accessing Trace, Analysis and Profiling options, and using the Analysis window, see the chapter describing tracing in RealView Debugger v1.7 Extensions User Guide.
Appendix A
Configuration Files Reference

This appendix describes the files set up for a new installation of RealView® Debugger v1.7, where they are stored, and what information each file holds. This appendix assumes that you have chosen a Typical installation. It contains the following sections:

• **Overview** on page A-2
• **Files in the etc directory** on page A-3
• **Files in the home directory** on page A-5.
A.1 Overview

RealView Debugger creates files containing default settings and target configuration information when you first install the product. The files created (or modified) depend on what kind of installation you have chosen. If you are upgrading from an earlier version of RealView Debugger, some files are modified so that previous configuration information is not lost.

When you run the debugger for the first time, some of this information is copied into the RealView Debugger default home directory. This is an empty directory created by the installation process. If you are upgrading from an earlier version of RealView Debugger, the installer creates the home directory and copies your existing configuration details into it. This means that this area is not empty when you run the debugger for the first time.

--- Note ---

This appendix describes configuration files created (or modified) when you install RealView Debugger on Windows. The RealView Debugger v1.7 User Guide contains information for developers using RealView Debugger on non-Windows platforms. See Appendix B RealView Debugger for Sun Solaris and Red Hat Linux for details.
A.2 Files in the etc directory

When you install RealView Debugger v1.7, files containing default settings and target configuration details are installed in the default settings directory \etc. What files are installed, and their contents, depend on what other software is detected during the installation, for example RealView® Compilation Tools (RVCT), or ARM Developer Suite™ (ADS):

- **armul.var** A list of RealView ARMulator® ISS variants used by RealView Debugger.
  
  You can edit this file to add hand-made variants.

- **genmake.loc** Specifies the location of the development tools used when working with projects. RealView Debugger copies this file into the default home directory (when you first create a project) or merges project details (when you first work with an existing project).
  
  Do not edit this file manually.

- **gen_arx.mk** Toolchain-specific template used by the genmake program to generate makefiles.
  
  You can edit this file to modify the input/output of the genmake program.

- **genmake_*.loc** Toolchain-specific makefile template used to define the location of development tools, for example genmake_ARM_ADS.loc or genmake_C2.loc.
  
  Do not edit these files manually.

- **pARM*.prc** Processor-specific information files used to define support for emulators and simulators, and for project information, for example pARM.prc, pARM_C2.prc, or pARM_RVI.prc.
  
  Do not edit these files manually.

- **rvdebug.brd** The default board file used to contain target configuration settings. This file references .bcd and .jtg files.

- **rvdebug.tpl** Template file used to contain standard templates. You can edit these files for use in source files.

- **startup.mk** MSDOS DMACE startup file.
  
  You can edit this file if required.

- **targ_*.aco** Processor-specific instruction format files used by RealView Debugger to color assembler code in the File Editor, for example targ_ARM.aco.
  
  Do not edit this file manually.
Configuration Files Reference

**template.spr**  SPR template used by RealView Debugger.
Do not edit this file manually.

**.bcd**  Board/Chip definition files supplied by hardware manufacturers.
If you want to make changes to these files, copy them into your default home directory and edit them using the Connection Properties window, for example `CM740T.bcd`.

**.jtg**  JTAG configuration files, for example `arm.jtg`.

**.stp**  RealView Debugger internal template files containing debugger defaults. These are used as internal support files or as template files for settings. For example, they are used to support project properties and define options such as workspace settings, for example `prj.set.stp`.
Do not edit these files manually.

For a new installation, some of these files are copied into your RealView Debugger default home directory when the debugger runs for the first time.
A.3 Files in the home directory

When you install and run RealView Debugger v1.7 for the first time, files containing default settings and target configuration details are copied into your RealView Debugger default home directory where they can be maintained.

If you are upgrading from an earlier version of RealView Debugger, the installer creates the home directory and copies your existing configuration details into it. This means that this area is not empty when you run the debugger for the first time:

- **armreg.sig**: An internal settings file that is created or updated each time you run RealView Debugger. Do not edit this file manually.
- **exphist.sav**: Your personal history file. This file keeps a record of each session and stores your personal favorites, for example breakpoints. This file is updated at the end of each debugging session.
- **genmake.loc**: Specifies the location of the development tools used when working with projects. RealView Debugger copies this file into the default home directory when you first create a project.
- **rvdebug.aws**: Your default workspace settings file. For a new installation, RealView Debugger creates this file when it runs for the first time. By default, this file is updated with the state of the debugger at the end of each debugging session.
- **rvdebug.brd**: Your board file containing target configuration settings. For a new installation, this is a duplicate of the file installed in the default settings directory `\etc`. This file references `.rbe` and `.bcd` files.
- **rvdebug.ini**: Specifies global configuration settings used across all workspaces or when working without a workspace. For a new installation, RealView Debugger creates this file when it runs for the first time.
- **rvdebug.sav**: This file specifies how each RealView Debugger session starts. For a new installation, RealView Debugger creates this file the first time you close down after performing an operation. By default, this file is updated at the end of each debugging session.
- ***.auc**: These are default target configuration settings files created when you first run RealView Debugger, for example `default.auc`. These files are referenced by the `.rbe` files.
Configuration Files Reference

* .bcd
Board/Chip definition files that you might have copied into your home directory, from the default settings directory \etc, to make changes.
Make copies of files so that the installation files are not changed.

* .cnf
These are default target configuration settings files that are created, by the RDI configuration utility, when you first run RealView Debugger, for example armulator.cnf.
These files are referenced by the .rbe files.

* .rbe
There is one .rbe configuration file for each available RDI target, for example armulator.rbe. This file is updated when you make changes to the RDI settings.

* .rvc
Configuration files for RV-msg connections. You change the contents of one of these files when you modify the configuration of an RV-msg connection using the RVConfig dialog box.
Do not edit these files manually.

A.3.1 Backup files

When you change a configuration file, RealView Debugger makes a backup copy of the current version to enable you to restore your previous settings, for example rvdebug.aws.bak and rvdebug.brd.bak. By default, backup files are given the .bak extension and are stored in the same location as the original file. You can change this behavior in your workspace.
Appendix B
Moving from AXD to RealView Debugger

This appendix is aimed at users who are moving development projects from ARM® eXtended Debugger (AXD) to RealView® Debugger. It contains a series of Frequently Asked Questions (FAQs) taken from the ARM Support website.

It contains the following sections:
- RealView Debugger configuration on page B-2
- RealView Debugger operations on page B-6
- Project management in RealView Debugger on page B-9.
B.1 RealView Debugger configuration

This section answers FAQs about configuration options in RealView Debugger.

B.1.1 Where do I find Debugger Internal variables?

Select View → Pane Views → Registers to open the Register pane at the default location. The Debugger Internals are available from the Debug tab inside this pane. To see a list of the available tabs, right click on the double arrows to the left of the Register pane tabs.

B.1.2 How do I set top_of_memory?

In RealView Debugger, top_of_memory is used to set the application stack base for a semihosted application running on a remote target.

If top_of_memory is not set, RealView Debugger sets it to a default value of 0x20000. A warning is displayed in the Cmd and Log tabs:

No stack/heap or top of memory defined - using defaults.

The value of top_of_memory can be overridden for a single debug session from the Debug tab in the Register pane (see Where do I find Debugger Internal variables?).

The value of top_of_memory can also be set for a particular target connection using the Connection Properties window:

1. Ensure that you disconnect from the target before making these changes.
2. Select File → Connection → Connect to Target... to open the Connection Control window.
3. Right-click on the connection, for example RealView ICE or Multi-ICE, and select Connection Properties... from the context menu.
   The appropriate branch in the Connection Properties window opens automatically.
4. Drill down through the tree:
   - Advanced_Information
   - Default
   - ARM_config
5. Right-click on the Top_memory entry and set the new connection default.
**Note**

To use the new setting, you must now connect to the target.

If a Board/Chip definition file is selected for this connection, then this file might contain a value for `top_of_memory` that overrides the target connection setting.

### B.1.3 How do I configure `vector_catch`?

`vector_catch` is a mechanism used to trap processor exceptions. This feature is typically used in the early stages of development to trap processor exceptions before the appropriate handlers are installed. You select the vectors to trap by editing the `vector_catch` value.

The value of `vector_catch` can be overridden for a single debug session from the Debug tab of the Register pane (see *Where do I find Debugger Internal variables?* on page B-2). The value of `vector_catch` represents a bit field, where a set bit corresponds to a trapped exception - the LSbit corresponds to the reset vector. Although the value can only be displayed in hexadecimal, values can be entered in binary format using the notation `0b`. The default value of `0x13B` (`0b100111011`) corresponds to trapping:

<table>
<thead>
<tr>
<th>Exception</th>
<th>Trapped</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Undefined</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>SWI</td>
<td>No</td>
<td>SWI vector may also be trapped by the debugger to enable standard semihosting</td>
</tr>
<tr>
<td>Prefetch Abort</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Data Abort</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Reserved (Address)</td>
<td>Yes</td>
<td>This vector is not used on current ARM cores, there is no need to trap this</td>
</tr>
<tr>
<td>IRQ</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>FIQ</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Error</td>
<td>Yes</td>
<td>This a deprecated feature and performs no function</td>
</tr>
</tbody>
</table>

Table B-1 Trapped Processor Exceptions (defaults)
Moving from AXD to RealView Debugger

The state of individual vectors for catching can also be set for a particular target connection using the Connection Properties window:

1. Ensure that you disconnect from the target before making these changes.

2. Select File → Connection → Connect to Target... to open the Connection Control window.

3. Right-click on the connection, for example Multi-ICE, and select Connection Properties... from the context menu.
   The appropriate branch in the Connection Properties window opens automatically.

4. Drill down through the tree:
   - Advanced Information
   - Default
   - ARM_config
   - Vectors

5. Set the catching of each vector in the group for the new connection default.


   ___ Note  ________

   To use the new setting, you must now connect to the target.

   If a Board/Chip definition file is selected for this connection, then this file might contain a value for vector_catch that overrides the target connection setting.

B.1.4 How do I configure semihosting?

Semihosting is a mechanism that captures I/O requests made by code running on the target (typically library code), and communicates these to the host system for handling. For example, application printfs appear, by default, within the debugger console window. Semihosting can be enabled or disabled from the debugger.

Semihosting can be set for a single debug session from the Debug tab of the Register pane (see Where do I find Debugger Internal variables? on page B-2). The available options, if you are using Multi-ICE, are NO, STD (Standard), and DCC.

Semihosting can also be set for a particular target connection using the Connection Properties window:

1. Ensure that you disconnect from the target before making these changes.
2. Select **File → Connection → Connect to Target...** to open the Connection Control window.

3. Right-click on the connection, for example *Multi-ICE*, and select **Connection Properties...** from the context menu.
   The appropriate branch in the Connection Properties window opens automatically.

4. Drill down through the tree:
   - **Advanced Information**
   - **Default**
   - **ARM_config**
   - **Semihosting**

5. Set the **Enabled** option for the new connection default.

6. Select **File → Save and Close**.

   **Note**
   To use the new setting, you must now connect to the target.

   If a Board/Chip definition file is selected for this connection, then this file might contain a value for **Semihosting** that overrides the target connection setting.
B.2 RealView Debugger operations

This section answers FAQs about RealView Debugger during debugging.

B.2.1 Why am I unable to connect to more than one target?

RealView Debugger is sold as a basic core product to which a number of license managed additional features can be added. Examples of additional features include multi-core debug, RTOS awareness, and support for various DSPs (for example, DSP group Oak/Teak family).

The basic version of RealView Debugger only supports single core debugging. In single core debug mode you can only connect to one target at a time. If you need to perform multi-core debugging you must purchase an additional license key to enable this feature.

B.2.2 Viewing Coprocessor variables

The RealView Debugger Register pane view includes an additional tab giving the user visibility of coprocessor registers such as CP15. In RealView Debugger v1.6 the contents of these registers were displayed as hexadecimal values and not enumerated. Future releases of RealView Debugger will add support for the enumeration of these registers. Please consult the appropriate Technical Reference Manual (TRM) for your ARM core to obtain information on the function of bit fields within each CP15 register.

B.2.3 How do I load symbols for an image?

To only load the symbols for an image in RealView Debugger:

1. Select File → Load Image... to display the Load File to Target dialog box.
2. Ensure that the Symbols Only check box is selected.

B.2.4 RealView ARMulator ISS Benchmarking

Currently RealView Debugger does not support RealView ARMulator® ISS benchmarking, as described in ARM Application Note 93. This will be added in a future release.

B.2.5 Debug illusion issues

There are a number of known debug illusion issues that we are working towards fixing in future releases of RealView Debugger:
C and C++ problems

Inlined Functions:
- Breakpoints on inline functions may only place a breakpoint on a single indeterminate instance.
- Stepping inlined functions might step to locations that are unexpected by the user. In particular this applies for STEP OVER and on the boundaries of the inlined function.

Call-stack:
- Variables stored in registers that are not on the top call-frame show indeterminate values.

Split Variables:
- Variables whose location is split might not display values correctly. This primarily affects ‘long long’ variables and small structures passed into functions by value (or small structures created on the stack).

C++ problems

Templates:
- Breakpoints on templated functions might only place a breakpoint on a single indeterminate instance.

Multiple Inheritance:
- Display of base classes does not always work, especially when there are multiple base classes.
- Access to base class members does not always work, especially when there are multiple base classes.

Static data members:
- Static data members are not displayed when viewing a class.

Class Browser:
- The display of classes in the class browser can be problematic. Sometimes multiple instances of classes are shown and the request to locate to source and dsm might not function correctly.
Exceptions:

- No support for stepping exceptions when on throw statement.
- No current support for catching exceptions (except by setting breakpoints in catch handler).

Namespaces:

- Support for namespaces not yet implemented.
B.3 Project management in RealView Debugger

This section answers FAQs about project management features in RealView Debugger.

B.3.1 Can I convert CodeWarrior projects into RealView Debugger projects?

Currently it is not possible to convert CodeWarrior .mcp files directly to RealView Debugger .prj files. However, the mcp2make utility, available from the ARM Developer Suite™ (ADS) v1.2 Downloads page, can be used to assist in converting CodeWarrior projects into makefiles. Makefiles produced in this way can then be imported into RealView Debugger, as detailed in Can I use existing makefiles with RealView Debugger? on page B-10.

B.3.2 Specifying paths to header files in projects

To do this:

1. Select Project → Project Properties... to open the Project Properties window.
2. Expand the appropriate COMPILE= group.
3. Expand the appropriate Preprocessor group.
4. Right-click on the Include entry in the right pane.
5. Select Edit as Directory Name from the context menu.
6. Locate the required directory and click Select.
7. Select File → Save and Close.

This saves the project file and regenerates the makefile.

B.3.3 Adding source files to an existing project

There are two ways to do this.

From the Project Properties window

To do this:

1. Select Project → Project Properties... to open the Project Properties window.
2. Expand the appropriate COMPILE= or ASSEMBLE= group.
3. Expand the appropriate Sources group.
4. Right-click on the Files entry in the right pane.

5. Select **Edit as Filename** from the context menu.

6. Locate the required source file and click **Save**.

7. Select **File → Save and Close**.

This saves the project file and regenerates the makefile.

**From the main menu**

Alternatively, from the main menu you can select **Project → Add Files to Project**.

--- Note ---

Note that the Project Properties window must be closed to do this.

**B.3.4 How does RealView Debugger make use of makefiles?**

RealView Debugger automatically generates appropriate makefiles when projects are created or updated. Makefiles are constructed using the user-defined information in the Project Properties window and the gen_***.mk template located in the \etc subdirectory of the RealView Debugger installation. Here *** corresponds to the default processor family and toolchain.

Generated makefiles can be viewed using any standard text editor, but hand editing of generated makefiles is not recommended as RealView Debugger will automatically update any makefiles to reflect any subsequent changes made in the Project Properties window.

**B.3.5 Can I use existing makefiles with RealView Debugger?**

Yes, RealView Debugger allows you to create a new project by importing an existing makefile.

Projects that are created this way differ from projects that are created entirely in RealView Debugger and further project configuration must be performed by manually editing the makefile outside RealView Debugger.

Imported makefiles must contain the following build rules:

- `clean`
- `all`
- `rebuild`
Example makefile

Below is an example makefile suitable for use with RealView Debugger:

clean:
# for Unix:
# rm -f *.o *.axf
# for Windows
   if exist *.o del *.o
   if exist *.axf del *.axf

all: project1.axf
# for Unix
# echo Build completed
# for Windows
   if exist *.axf echo Build completed

rebuild: clean all

project1.axf: data.o io.o main.o
   armlink data.o io.o main.o -o project1.axf
io.o: io.c
   armcc -c io.c
data.o: data.c
   armcc -c data.c
main.o: main.c
   armcc -c main.c
Moving from AXD to RealView Debugger
Glossary

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

**Address breakpoint**  
A type of breakpoint.  
*See also* Breakpoint.

**ADS**  
*See* ARM Developer Suite.

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

**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.

**Backtracing**  
*See* Call Stack.
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 Debugger uses the term board to refer to a target processor, memory, peripherals, and debugger connection method.

Board file
In RealView Debugger, the board file is the top-level configuration file, normally called rvdebug.brd, that references one or more other files.

Breakpoint
A user defined point at which execution stops in order that a debugger can examine the state of memory and registers.

See also Hardware breakpoint and Software breakpoint.

Call Stack
This is a list of procedure or function call instances on the current program stack. It might also include information about call parameters and local variables for each instance.

Conditional breakpoint
A breakpoint that halts execution when a particular condition becomes true. The condition normally references the values of program variables that are in scope at the breakpoint location.

Context menu
See Pop-up menu.

Core module
In the context of Integrator, an add-on development board that contains an ARM processor and local memory. Core modules can run stand-alone, or can be stacked onto Integrator motherboards.

See also Integrator.

DCC
See Debug Communications Channel.

Debug Agent (DA)
The Debug Agent resides on the target to provide target-side support for Running System Debug (RSD). The Debug Agent can be a thread or built into the RTOS. The Debug Agent and RealView Debugger communicate with each other using the debug communications channel (DCC). This enables data to be passed between the debugger and the target using the ICE interface, without stopping the program or entering debug state.

See also Running System Debug.

Debug Communications Channel (DCC)
A debug communications channel enables data to be passed between RealView Debugger and the EmbeddedICE logic on the target using the JTAG interface, without stopping the program flow or entering debug state.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deprecated</td>
<td>A deprecated option or feature is one that you are strongly discouraged from using. Deprecated options and features will not be supported in future versions of the product.</td>
</tr>
<tr>
<td>Digital Signal Processor (DSP)</td>
<td>DSPs are special processors designed to execute repetitive, maths-intensive algorithms. Embedded applications might use both ARM processor cores and DSPs.</td>
</tr>
<tr>
<td>Doubleword</td>
<td>A 64-bit unit of information.</td>
</tr>
<tr>
<td>DSP</td>
<td>See Digital Signal Processor.</td>
</tr>
<tr>
<td>Embedded Trace Macrocell (ETM)</td>
<td>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.</td>
</tr>
<tr>
<td>EmbeddedICE logic</td>
<td>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.</td>
</tr>
<tr>
<td>ETM</td>
<td>See Embedded Trace Macrocell.</td>
</tr>
<tr>
<td>Execution vehicle</td>
<td>Part of the debug target interface, execution vehicles process requests from the client tools to the target.</td>
</tr>
<tr>
<td>Halfword</td>
<td>A 16-bit unit of information.</td>
</tr>
<tr>
<td>Halted System Debug (HSD)</td>
<td>Usually used for RTOS aware debugging. Halted System Debug (HSD) means that you can only debug a target when it is not running. This means that you must stop your debug target before carrying out any analysis of your system. With the target stopped, the debugger presents RTOS information to the user by reading and interpreting target memory. See also Running System Debug.</td>
</tr>
<tr>
<td>Hardware breakpoint</td>
<td>A breakpoint that is implemented using non-intrusive additional hardware. Hardware breakpoints are the only method of halting execution when the location is in Read Only Memory (ROM). Using a hardware breakpoint often results in the processor halting completely. This is usually undesirable for a real-time system. See also Breakpoint and Software breakpoint.</td>
</tr>
<tr>
<td>HSD</td>
<td>See Halted System Debug.</td>
</tr>
</tbody>
</table>
IEEE 1149.1 The IEEE Standard that defines TAP. Commonly (but incorrectly) referred to as JTAG.

See also Test Access Port

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

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

JTAG See Joint Test Action Group.

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

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

See also Big-endian.

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

Pop-up menu Also known as Context menu. A menu that is displayed temporarily, offering options relevant to your current situation. Obtainable in most RealView Debugger windows or panes by right-clicking with the mouse pointer inside the window. In some windows the pop-up menu can vary according to the line the mouse pointer is on and the tabbed page that is currently selected.

Processor core The part of a microprocessor that reads instructions from memory and executes them, including the instruction fetch unit, arithmetic and logic unit and the register bank. It excludes optional coprocessors, caches, and the memory management unit.

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

RDI See Remote Debug Interface.

RealView ARMulator ISS (RVISS) 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.

See also RDI and RealView Connection Broker.
Glossary

**RealView Compilation Tools (RVCT)**

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

**RealView 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 Trace**

Part of the RealView Debugger product that extends the debugging capability with the addition of real-time program and data tracing. It is available from the Code window.

**RealView ICE (RVI)**

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

**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, RealView ARMulator ISS)
- a debug monitor running on hardware that is based on an ARM core accessed through a communication link (for example, Angel)
- a debug agent controlling an ARM processor through hardware debug support (for example, RealView ICE or Multi-ICE).

**Remote_A**

Remote_A is a software protocol converter and configuration interface. It converts between the RDI 1.5 software interface of a debugger and the Angel Debug Protocol used by Angel targets. It can communicate over a serial or Ethernet interface.

**RSD**

*See* Running System Debug.

**RTOS**

Real Time Operating System.

**Running System Debug (RSD)**

Used for RTOS aware debugging. *Running System Debug* (RSD) means that you can debug a target when it is running. This means that you do not have to stop your debug target before carrying out any analysis of your system. RSD gives access to the application using a *Debug Agent* (DA) that resides on the target. The Debug Agent is scheduled along with other tasks in the system.

*See also* Debug Agent and Halted System Debug.

**RVCT**

*See* RealView Compilation Tools.

**RVISS**

*See* RealView ARMulator ISS.
Scan chain
A scan chain is made up of serially-connected devices that implement boundary-scan technology using a standard JTAG TAP interface. Each device contains at least one TAP controller containing shift registers that form the chain. Processors might contain several shift registers to enable you to access selected parts of the device.

Scope
The range within which it is valid to access such items as a variable or a function.

Semihosting
A mechanism whereby I/O requests made in the application code are communicated to the host system, rather than being executed on the target.

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

Software breakpoint
A breakpoint that is implemented by replacing an instruction in memory with one that causes the processor to take exceptional action. Because instruction memory must be altered software breakpoints cannot be used where instructions are stored in read-only memory. Using software breakpoints can enable interrupt processing to continue during the breakpoint, making them more suitable for use in real-time systems.

See also Breakpoint and Hardware breakpoint.

Software Interrupt (SWI)
An instruction that causes the processor to call a programmer-specified subroutine. Used by the ARM standard C library to handle semihosting.

SWI
See Software Interrupt.

TAP
See Test Access Port.

TAP Controller
Logic on a device which enables access to some or all of that device for test purposes. The circuit functionality is defined in IEEE1149.1.

See also Test Access Port and IEEE1149.1.

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

Target vehicle
Target vehicles provide RealView Debugger with a standard interface to disparate targets so that the debugger can connect easily to new target types without having to make changes to the debugger core software.

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

Test Access Port (TAP)
The port used to access the TAP Controller for a given device. Comprises TCK, TMS, TDI, TDO, and nTRST (optional).
### Glossary

**Tracepoint**  
A tracepoint can be a line of source code, a line of assembly code, or a memory address. In RealView Debugger, you can set a variety of tracepoints to determine exactly what program information is traced.

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

**Trigger**  
In the context of breakpoints, a trigger is the action of noticing that the breakpoint has been reached by the target and that any associated conditions are met.

In the context of tracing, a trigger is an event that instructs the debugger to stop collecting trace and display the trace information around the trigger position, without halting the processor. The exact information that is displayed depends on the position of the trigger within the buffer.

**TVS**  
*See* Target Vehicle Server.

**Watch**  
A watch is a variable or expression that you require the debugger to display at every step or breakpoint so that you can see how its value changes. The Watch pane is part of the RealView Debugger Code window that displays the watches you have defined.

**Watchpoint**  
In RealView Debugger, this is a hardware breakpoint.

**Word**  
A 32-bit unit of information.
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
Active projects 5-13
Analysis viewing results 7-28
Analysis window 7-28
Audience, intended vi
Auto-projects creating for images 5-20
properties 5-20
see also Projects
AXD moving to RealView Debugger B-1

B
Base directory projects 5-10
Binding projects 5-13
Board file 2-4, 4-4
Board/Chip Definition file 2-4
Book, about this vi
Breakpoints 5-27
complex 5-27
setting simple 4-16
simple 5-27
Browsers 5-25
Build 5-10
changing tools 5-37
finding errors 5-41
tools location 5-4

C
Cancel command 7-16
Close down connected 6-8
disconnected 6-8
Code views 4-14, 4-17, 7-6
disassembly tab 4-14, 4-17
Dsm tab 4-14, 4-17
source code tab 4-14, 4-17
Src tab 4-14, 4-17
Code window 7-20
Actions toolbar 7-9, 7-16
Analysis window 7-28
attachment 7-4
Button toolbars 7-16
button toolbars 7-9
CAPS 7-11
changing panes 7-8
Color Box 7-10
components 7-3
Cursor location field 7-11
customizing toolbars 7-19
debug views 7-5
default 4-2, 7-3
default panes 7-5
docking panes 7-20
Editing toolbar 7-9, 7-16, 7-18
Editor Window 7-23
File Editor pane 7-5
floating panes 7-20
focus control 7-20
Index

in-place editing 4-10, 7-21
JOU 7-11
LOG 7-11
Main menu 7-12
NUM 7-11
pane controls 7-8
panes 7-5
Resource Viewer 7-24
Status line 7-11
STDIOlog 7-11
title bar 4-14, 7-4
windows and panes 7-5
Color Box 7-10
Comments
on documentation x
on RealView Debugger x
Concepts
connection 2-2
debug target 2-2
DSP 2-2
multithreading 2-3
RTOS 2-2
single connection access 2-2
Concepts and terminology 2-2
see also Glossary
Connect mode 4-7
Connecting 4-4, 4-6
and close down 6-8
RDI details 4-6
Connection
concept 2-2
Connection Broker 2-5
Local mode 2-5
Remote mode 2-5
Connection Control window managing connections 4-6
Create New Project
dialog 5-5
Create Standard Project
dialog 5-6
Customizing
panes 5-33
toolbars 7-19
windows 5-33
D
DA
Debug Agent 5-28
Debug Agent 5-28
Debug target
collection 2-2
configuration 4-4
Connect mode 4-7
collection 4-4, 4-6
Disconnect mode 6-5
disconnecting 6-5
displaying target status 4-17, 6-6
execution vehicles 2-5
interface 2-5
Multi-ICE 2-5
Processor State 4-17, 6-6
RealView ARMulator ISS 2-5, 3-2
RealView ICE 2-5
reconnecting using workspace 6-9
Debug target interface 2-5
Debugger internals
Register pane 5-17
Debugging
basic tasks 5-16
going started 4-16
tutorial 5-16
Debugging environment 2-4
Debugging mode 2-7
Desktop
Actions toolbar 7-9, 7-16
attaching windows 7-4
bottom pane 7-7
Call Stack pane 7-6
CAPS 7-11
changing panes 7-8
Code window 7-3
Collapse Pane 7-9
Color Box 7-10
Cursor location field 7-11
customizing toolbars 7-19
Debug menu 7-15
docking panes 7-20
Edit menu 7-13
Editing toolbar 7-9, 7-18
Expand Pane 7-9
File Editor pane 7-5
File menu 7-12
Find menu 7-13
floating panes 7-20
floating toolbars 7-9
focus 7-20
formatting views 7-8
Help menu 7-15
hiding toolbars 7-9
in-place editing 4-10, 7-21
JOU 7-11
LOG 7-11
Main menu 7-12
Memory pane 7-7
middle row pane 7-6
NUM 7-11
Pane Content menu 7-8
pane controls 7-8
Pane Menu 7-9
Processor State 7-17
Project menu 7-14
Register pane 7-6
scroll bars 7-22
Scroll menu 7-22
side pane 7-6
splash screen 7-2
Status line 7-11
STDIOlog 7-11
tabs 7-22
Tools menu 7-14
View menu 7-13
viewing toolbars 7-9
Watch pane 7-7
Dialog box
Create New Project 5-5
Create Standard Project 5-6
Interactive Register Settings 4-9
Set Address/Data Break/Tracepoint 5-29
Upgrade Project 5-35
Upgrade Project ToolChain 5-36
Directories
default home A-5
default settings A-3
project base 5-2
projects 5-11
\etc A-3
\Examples vi
\home A-5
Disassembly view 4-14, 4-17
Disconnect mode 6-5
Disconnecting 6-5
Index

Documentation
DynaText 3-9
online 3-9
PDF 3-9
structure of this book vii
using the suite 1-3
Documentation feedback x
DSP concept 2-2

and close down 6-8

Editor Window 7-23
Enquiries x
Execution vehicle
see Debug target
Exiting
RealView Debugger 6-8
Expression pointer 5-25

Favorites persistence info 6-3
Feedback
on documentation x
on RealView Debugger x
File Editor
File Editor pane 7-5
Files
adding to projects 5-39
armul.var A-3
board file 2-4, 4-4
Board/Chip Definition 2-4
configuration A-3
genmake.loc A-3
genmake_ARM_ADS.loc A-3
gen_arx.mk A-3
installed in \etc A-3
installed in \home A-5
pARM.prc A-3
project 5-11
rvdebug.brd A-3
rvdebug.tpl A-3
startup.mk A-3
targ_ARM.aco A-3

template.spr A-4
*.auc A-5
*.aws 6-2, A-5
*.axf 4-12, 5-12
*.bak 5-12, A-6
*.bcd 2-4, A-4, A-6
*.brd 2-4, 4-4, A-5
*.cnf A-6
*.ini A-5
*.jtg A-4
*.loc A-5
*.mk 5-10
*.prj 5-10
*.rbe A-6
*.rvc A-6
*.sav 6-3, A-5
*.sig A-5
*.stp A-4
Focus
in Code window 7-20
Frame pointer 5-24

Glossary Glossary-1

Halted System Debug
HSD 5-28
History list
saving 6-3
HSD
Halted System Debug 5-28
switching to RSD 5-28

Images
loading 4-12
reloading 4-15
unloading 4-15
In-place editing 4-10, 7-21
focus 7-21
Intended audience vi
Interactive Register Settings dialog 4-9

Line numbers
setting breakpoints 4-16, 5-29
turning on 4-13, 4-16, 5-15, 5-34
Loading
images 4-12

Make
command 5-41
Makefiles
generated 5-7, 5-10, 5-39
Memory 5-20
for ARM processors 4-8
Menus
Debug 7-15
Edit 7-13
File 7-12
Find 7-13
Help 7-15
Main 7-12
Pane Content 7-8
Pane Menu 7-9
Pane Views 7-8
Project 7-14
Tools 7-14
View 7-13
Workspace 6-2
Mode
debugging 2-7
MSC_CLOCK
project compiler option 5-8
Multi-ICE 2-5
Multiprocessor access
concept 2-2
Multithreading
concept 2-3

Network Broker 2-5
Numbers
Index

turn on lines 4-13, 4-16, 5-15, 5-34

P

Panes
- bottom 7-7
- Break/Tracepoints 5-31
- Call Stack 5-23, 7-6
- changing 7-8
- collapsing 7-9
- customizing 5-33
- docking 7-20
- expanding 7-9
- File Editor 7-5
- floating 7-20
- focus 7-20
- hiding 5-34
- in Code window 4-2
- Memory 5-20, 5-24, 7-7
- middle row 7-6
- multiple 5-33
- Process Control 5-18
- Register 4-10, 5-16, 7-6
- side 7-6
- Stack 5-23, 5-24
- Watch 5-26, 7-7
- PC 4-13
- Persistence info 2-6, 6-2, 6-4
- favorites 6-3
- files A-5
- Problem solving x
- Process Control pane
- type ahead 5-19
- Processor State 4-17, 6-6
- Product feedback x
- Program Counter 4-13
- Projects
- active 5-13
- adding files 5-39
- assembler settings summary 5-9
- associated images 5-19
- auto-projects 5-20
- base directory 5-10
- binding 5-13
- bound 7-4
- build errors 5-41
- build tools 5-4, 5-37
- building applications 5-10
- closing projects 5-12
- compiler options 5-8
- compiler settings summary 5-9
- files 5-11
- generated makefiles 5-7, 5-10, 5-39
- getting started 5-3, 5-35
- linker settings summary 5-9
- new projects 5-3
- properties 5-7
- unbound 7-4
- upgrading 5-35, 5-37

Q

Queries x

R

RDI
- connection details 4-6
- Real Time Operating Systems 2-2
- RealView ARMulator ISS 2-5, 3-2
- breakpoints 5-30
- connecting 4-6
- debugger internals 5-17, 7-6
- in Connection Control window 4-6
- in examples vi
- memory 4-8
- setting top of memory 4-8
- setting top of stack 4-8
- RealView Connection Broker 2-5
- RealView Debugger
- closing down 6-8
- Code window 4-2, 7-3
- components 2-4
- concepts 2-2
- connecting to a target 4-4
- debug target interface 2-5
- debugging mode 2-7
- desktop 7-2
- environment 2-4
- exiting 6-8
- for AXD users B-1
- image loading 4-12
- image reloading 4-15
- image unloading 4-15
- introduction 2-2
- persistence info 2-6
- projects 5-3, 5-35
- start debugging 4-16
- starting 4-2
- stopping 6-8
- target configuration 4-4
- tutorial 5-2
- TVS 2-4
- using the documentation 1-3
- window attachment 7-4
- RealView ICE 2-5
- debugger internals 5-17
- RealView Network Broker 2-5
- RealView Simulator Broker 2-5
- Registers 5-16
- banks 5-18
- changing contents 4-9, 4-10
- Reloading
- images 4-15
- Resource Viewer 7-24
- RSD
- Running System Debug 5-28
- switching to HSD 5-28
- RTOS 2-2
- concept 2-2
- HSD 5-28
- RSD 5-28
- threads 2-3
- viewing resources 7-24
- Running System Debug
- RSD 5-28

S

Save
- session 6-2
- Settings options 6-4
- Scrolling
- menu 7-22
- Session
- history list 6-3
- saving 6-2
- Settings options 6-4
- startup file 6-3
- workspace 6-2
- Set Address/Data Break/Tracepoint
- dialog 5-29
- Single connection access
## Index

<table>
<thead>
<tr>
<th>Concept</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>concept</td>
<td>2-2</td>
</tr>
<tr>
<td>Software Problem Report</td>
<td>x, 7-15</td>
</tr>
<tr>
<td>Source view</td>
<td>4-14, 4-17</td>
</tr>
<tr>
<td>SPR</td>
<td></td>
</tr>
<tr>
<td>Stack</td>
<td>5-25</td>
</tr>
<tr>
<td>EP</td>
<td></td>
</tr>
<tr>
<td>for RealView ARMulator ISS</td>
<td>4-8</td>
</tr>
<tr>
<td>FP</td>
<td>5-24</td>
</tr>
<tr>
<td>SP</td>
<td>5-24, 5-25</td>
</tr>
<tr>
<td>Stack pointer</td>
<td>5-24</td>
</tr>
<tr>
<td>default</td>
<td>5-25</td>
</tr>
<tr>
<td>Starting</td>
<td>4-16</td>
</tr>
<tr>
<td>debugging</td>
<td></td>
</tr>
<tr>
<td>RealView Debugger</td>
<td>4-2</td>
</tr>
<tr>
<td>Starting the tutorial</td>
<td>5-2</td>
</tr>
<tr>
<td>Startup file</td>
<td>6-3</td>
</tr>
<tr>
<td>Structure of this book</td>
<td>vi</td>
</tr>
</tbody>
</table>

### T

| Tabbed pages                   | 7-22 |
| Tabs                           | 7-22 |
| list of available              | 7-22 |
| red text                       | 7-22 |
| right-click                    | 7-22 |
| right-click on blank area      | 7-22 |
| right-click on scroll arrow    | 7-22 |
| using                          | 7-22 |
| Target                         |      |
| see Debug target               |      |
| Target Vehicle Server (TVS)    | 2-4  |
| Terminology                    |      |
| Glossary-1                     |      |
| Terminology and concepts       | 2-2  |
| Threads                        | 2-3  |
| HSD                            | 5-28 |
| RSD                            | 5-28 |
| Toolbars                       |      |
| Actions                        | 7-16 |
| Build group                    | 7-17 |
| Command cancel                 | 7-16 |
| Connection button              | 7-27 |
| Connection group               | 7-17 |
| Context controls               | 7-16 |
| customizing                    | 7-19 |
| Edit group                     | 7-16 |
| Editing                        | 7-18 |
| Execution group                | 7-16 |
| File field                     | 7-18 |
| File group                     | 7-16 |
| Find field                     | 7-18 |
| Image load group               | 7-16 |
| Line number field              | 7-18 |
| Pan Management controls        | 7-17 |
| Processor State group          | 7-17 |
| Source control button          | 7-18 |
| Thread button                  | 7-17 |
| Tooltip Evaluation             | 5-22 |
| Trace                          |      |
| profiling                      | 7-28 |
| viewing results                | 7-28 |
| Tutorial                       |      |
| breakpoints                    | 5-27 |
| building applications          | 5-10 |
| changing memory contents       | 5-21 |
| clearing breakpoints           | 5-31 |
| closing projects               | 5-12 |
| conditional breakpoints        | 5-29 |
| creating a project             | 5-3  |
| custom panes                   | 5-33 |
| debugger internals             | 5-17 |
| debugging tasks                | 5-16 |
| deleting                       | 5-42 |
| displaying memory              | 5-20 |
| displaying registers           | 5-16 |
| finding build errors           | 5-41 |
| generated makefiles            | 5-7, 5-39 |
| getting started                | 5-2  |
| hiding panes                   | 5-34 |
| how to use                     | 5-2  |
| managing breakpoints           | 5-31 |
| multiple panes                 | 5-33 |
| process control                | 5-18 |
| project base directory         | 5-10 |
| project files                  | 5-11 |
| project properties             | 5-7, 5-9 |
| setting compiler options       | 5-8  |
| setting up a project           | 5-3, 5-35 |
| setting watches                | 5-26 |
| using browsers                 | 5-25 |
| using the call stack           | 5-23 |
| using the stack                | 5-23 |
| using tooltip evaluation       | 5-22 |
| using tooltips                 | 7-16 |

### U

| Unloading images               | 4-15 |

### V

| Views                          |      |
| code                           | 4-14, 4-17 |

### W

| Watches                        | 5-26 |
| Who should read this book      | vi   |
| Windows                        |      |
| Analysis                       | 7-28 |
| attachment                     | 4-3, 4-14, 7-4 |
| Build-Tool Properties          | 5-4, 5-38 |
| Code                           | 4-2  |
| Configuration Summary          | 5-9  |
| Connection Control             | 4-4, 4-5 |
| docking                        | 7-20 |
| Editor                         | 7-23 |
| floating                       | 7-20 |
| focus                          | 7-20 |
| Project Properties             | 5-7  |
| Resource Viewer                | 7-24 |
| Workspace                      |      |
| connection details             | 6-5, 6-8, 6-9 |
| saving                         | 6-2  |