RealView® Debugger

Version 4.1

Essentials Guide

ARM®

ARM DUI 0181N (ID052111)

RealView Debugger
Essentials Guide

Release Information

The following changes have been made to this document.

Change History

Date	Issue	Confidentiality	Change
April 2002	A	Non-Confidential	Release v1.5
September 2002	B	Non-Confidential	Release v1.6
February 2003	C	Non-Confidential	Release v1.6.1
September 2003	D	Non-Confidential	Release v1.6.1 for RealView Developer Suite v2.0
January 2004	E	Non-Confidential	Release v1.7 for RealView Developer Suite v2.1
December 2004	F	Non-Confidential	Release v1.8 for RealView Developer Suite v2.2
May 2005	G	Non-Confidential	Release v1.8 SP1 for RealView Developer Suite v2.2 SP1
March 2006	H	Non-Confidential	Release v3.0 for RealView Development Suite v3.0
March 2007	I	Non-Confidential	Release v3.1 for RealView Development Suite v3.1
September 2008	J	Non-Confidential	Release v4.0 for RealView Development Suite v4.0
27 March 2009	K	Non-Confidential	Release v4.0 SP1 for RealView Development Suite v4.0
28 May 2010	L	Non-Confidential	Release 4.1 for RealView Development Suite v4.1
30 September 2010	M	Non-Confidential	Release 4.1 SP1 for RealView Development Suite v4.1 SP1
31 May 2011	N	Non-Confidential	Release 4.1 SP2 for RealView Development Suite v4.1 SP2

Proprietary Notice

Confidentiality Status

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

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

Preface

This preface introduces the *RealView® Debugger Essentials Guide*. It contains the following sections:

About this book

RealView Debugger provides a powerful tool for debugging applications targeted at ARM® architecture-based processors. This book contains:

- an introduction to the software components that make up RealView Debugger
- 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.

Intended audience

This book is written for developers who are using RealView Debugger to debug applications targeted at ARM architecture-based processors. It assumes that you are experienced in developing applications for ARM platforms, but does not assume that you are familiar with 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 programs stored in the `\Examples` directory in your *RealView Development Suite* (RVDS) installation root, that are targeted at ARM architecture-based processors.

In general, examples use *RealView ARMulator® Instruction Set Simulator* (RVISS) to simulate an ARM architecture-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 ***About RealView Debugger***

Read this chapter for an introduction to RealView Debugger concepts and terminology, features, and a summary of the RealView Debugger documentation suite.

Chapter 2 ***Getting Started with RealView Debugger***

Read this chapter for a details of how to begin using RealView Debugger for the first time. It describes how to start RealView Debugger, make a connection, load an image ready to start debugging, and how to perform some basic debugging tasks.

Chapter 3 ***Changes to RealView Debugger***

Read this chapter for a summary of the changes to RealView Debugger in this release.

Appendix A *About Previous Releases*

Read this appendix for a details of RealView Debugger v4.1 and earlier releases.

Typographical conventions

The typographical conventions are:

italic	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.
`mono`space	Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.
`monospace italic`	Denotes arguments to commands and functions where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
< and >	Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`

Further reading

This section lists publications by ARM and by third parties.

See also:

- Infocenter, `http://infocenter.arm.com` for access to ARM documentation.
- ARM web site , `http://www.arm.com` for current errata, addenda, and Frequently Asked Questions.
- ARM Glossary, `http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html`, for a list of terms and acronyms specific to ARM.

ARM publications

This book contains information that is specific to this product. See the following documents for other relevant information:

- *RealView Debugger User Guide* (ARM DUI 0153)
- *RealView Debugger Target Configuration Guide* (ARM DUI 0182)
- *RealView Debugger Trace User Guide* (ARM DUI 0322)
- *RealView Debugger RTOS Guide* (ARM DUI 0323)
- *RealView Debugger Command Line Reference Guide* (ARM DUI 0175).

For details on using the compilation tools, see the books in the ARM Compiler toolchain documentation.

For an introduction to all the components of RVDS, see the *RealView Development Suite Getting Started Guide* (ARM DUI 0255).

For details on using RVISS, see the following book:

- *RealView ARMulator ISS User Guide* (ARM DUI 0207).

For details on using and configuring *Real-Time System Models* (RTSMs), see:

- *RealView Development Suite Real-Time System Model User Guide* (ARM DUI 0424).

For general information on software interfaces and standards supported by ARM tools, see the PDF files in:

install_directory\Documentation\Specifications\...

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

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

- *ARM DSTREAM Setting Up the Hardware* (ARM DUI 0481)
- *ARM DSTREAM System and Interface Design Reference* (ARM DUI 0499)
- *ARM DSTREAM and RVI Using the Debug Hardware Configuration Utilities* (ARM DUI 0498)
- *ARM RVI and RVT Setting Up the Hardware* (ARM DUI 0515)
- *ARM RVI and RVT System and Interface Design Reference* (ARM DUI 0517).

Other publications

For a comprehensive introduction to ARM architecture see:

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

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

Brian W. Kernighan, Dennis M. Ritchie, *The C Programming Language* (2nd edition, 1989). Prentice-Hall, ISBN 0-13-110362-8.

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

If you have any problems with this product, submit a Software Problem Report:

1. Select **Send a Problem Report...** from the RealView Debugger **Help** 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. E-mail the report to your supplier.

Feedback on this book

If you have comments on content then send an e-mail to `errata@arm.com`. Give:

- the title
- the number, ARM DUI 0181N
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Chapter 1
About RealView Debugger

This chapter explains how ARM® RealView® Debugger provides a debugging environment for embedded systems applications running on ARM architecture-based processors. It contains:

1.1 RealView Debugger concepts and terminology

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

The following terminology is used throughout the RealView Debugger documentation suite when describing debugging concepts:

Code window

The *Code window* is the starting point for all your debugging tasks and gives you access to features in the product. It displays connection state information and source code views, gives access to other windows, handles CLI commands, and displays debugger messages.

Connection The link between the debugger and the target. RealView Debugger enables you to connect to one or more targets, depending on your application debugging requirements.

Debug Configuration

A *Debug Configuration* defines a debugging environment for your development platform.

Debug Interface

The *Debug Interface* identifies the targets on your development platform, and provides the mechanism that enables RealView Debugger to communicate with those targets. The Debug Interface corresponds directly to a hardware run control unit or a software interface to simulated targets.

Development Platform

The *Development Platform* contains the components, either hardware or simulated, that you are using to develop your application. It can include:

- a base board, such as an Integrator/CP
- peripherals
- one or more ARM architecture-based processors
- CoreSight™ components.

OS-awareness

OS-awareness is a feature provided by RealView Debugger that enables you to:

- debug applications running on an embedded OS development platform, such as a *Real-Time Operating System* (RTOS)
- present thread information and scope some debugging operations to specific threads.

You must obtain and install the required OS-awareness plug-in.

Target A *Target* is the part of your development platform to which RealView Debugger can connect, and on which debugging operations can be performed. A target can be:

- A runnable target, such as an ARM architecture-based processor. When connected to a runnable target, you can perform execution-related debugging operations on that target, such as stepping and tracing.
- A non-runnable CoreSight component. CoreSight components provide a system-wide solution to real-time debug and trace.

These can be hardware or software targets.

See also

- the following in the *RealView Debugger User Guide*:
  - *Target connection* on page 1-23.

1.2 About the debugging environment

RealView Debugger uses a three-tier environment to debug applications:
- debugger software
- a debug interface layer, incorporating the Debug Interface
- the target.

RealView Debugger uses connection information in a Debug Configuration to describe:
- how the debugger connects to the target
- information required to use that connection
- the processor type of the target.

A single RealView Debugger instance can deal with multiple simultaneous connections.

See also:
- *Graphical User Interface*
- *Command Line Interface*
- *Supported Debug Interfaces*
- *Persistence information* on page 1-7.

1.2.1 Graphical User Interface

The *Graphical User Interface* (GUI) gives access to the main features of the debugger, command processing, and one or more Code windows.

1.2.2 Command Line Interface

The *Command Line Interface* (CLI) gives access to many of the features of the debugger through RealView Debugger commands. This enables you to create command scripts to automate debugging. You can access the CLI from the GUI, or run RealView Debugger in command-line mode. Some features are not available when running RealView Debugger in command-line mode.

1.2.3 Supported Debug Interfaces

The interface between RealView Debugger and a target is provided by a Debug Interface. Each Debug Interface processes requests from the client tools to the target. A Debug Interface might be a JTAG interface, a simulator, or a ROM monitor.

RealView Debugger supports the following hardware and software Debug Interfaces:

- ARM DSTREAM™ and RealView ICE are hardware Debug Interfaces
- software Debug Interfaces are:
  - *Instruction Set System Model* (ISSM)
  - Model Library
  - Model Process
  - *Real-Time System Model* (RTSM)
  - *RealView Instruction Set Simulator* (RVISS)
  - SoC Designer
  - VSTREAM.

DSTREAM, RealView ICE, RealView Trace, and RealView Trace 2

DSTREAM or RealView ICE enable you to debug a hardware development platform containing one or more ARM architecture-based processors. You can connect to a DSTREAM or RealView ICE unit using either a USB port or your local network.

Note

RealView Debugger does not support tracing from the external trace port of a SoC with DSTREAM.

If you have a RealView ICE unit:

- the addition of a RealView Trace unit enables you to perform tracing and analysis of:
  - processors containing an *Embedded Trace Module*™ (ETM™)
  - development platforms containing an *Embedded Trace Buffer*™ (ETB™).
- the addition of a RealView Trace 2 unit enables you to:
  - perform tracing and analysis of processors containing an *Embedded Trace Module*™ (ETM™)
  - perform tracing and analysis of development platforms containing an *Embedded Trace Buffer*™ (ETB™)
  - perform real-time profiling in cooperation with the ARM Profiler, and perform tracing and analysis at greater clock speeds.

Note

You must purchase DSTREAM, RealView ICE, RealView Trace, and RealView Trace 2 units separately.

Instruction Set System Model

ISSM simulates the following processors:

- Cortex™-A8
- Cortex-M0
- Cortex-M1
- Cortex-M3
- Cortex-R4.

ISSM runs on the same host computer as the debugger.

Model Library

Model Library enables you to connect to targets defined in your own CADI model libraries.

Model Process

Model Process enables you to connect to targets in your own CADI model that is currently running on your workstation.

Real-Time System Model

An RTSM contains a hard-coded system containing one or more specific simulated processors. When you attempt to connect to an RTSM target, RealView Debugger starts up the RTSM session before connecting to that target.

The following RTSMs are provided with RVDS Professional edition:

- Versatile Emulation Baseboard with ARM926EJ-S
- Versatile Emulation Baseboard with ARM1136JF-S
- Versatile Emulation Baseboard with ARM1176JZF-S
- Versatile Emulation Baseboard with Cortex-A5_MPx1
- Versatile Emulation Baseboard with Cortex-A5_MPx2
- Versatile Emulation Baseboard with Cortex-A8
- Versatile Emulation Baseboard with Cortex-A9_MPx1
- Versatile Emulation Baseboard with Cortex-A9_MPx2
- Versatile Emulation Baseboard with Cortex-R4
- MPS with Cortex-M3
- MPS with Cortex-M4.

Note

Be aware that *Emulation Baseboard* (EB) RTSMs are not intended to be software implementations of particular revisions of EB hardware.

Note

If you want to create your own RTSMs, you must purchase the RealView System Generator application.

RealView Instruction Set Simulator

RVISS 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 RVDS v2.2 SP1 and earlier releases.

SoC Designer

Models created with Carbon SoC Designer Plus can be debugged using RealView Debugger. You can debug a SoC Designer model by either:

- launching RealView Debugger from Carbon SoC Designer Simulator to debug the model you are currently viewing
- creating a SoC Designer Debug Configuration in RealView Debugger.

Note

You must purchase Carbon SoC Designer Plus separately.

VSTREAM

You can debug RTL models in a hardware emulation environment in a similar way to JTAG targets in a real SoC in silicon.

You must purchase and install:
- a supported EDA product
- the VSTREAM client and transactor software.

See also

- the following in the *RealView Debugger User Guide*:
  - Chapter 3 *Target Connection*
- the following in the *RealView Debugger Target Configuration Guide*:
  - Chapter 2 *Customizing a Debug Interface configuration*
  - Chapter 3 *Customizing a Debug Configuration*
  - Appendix B *ISSM Configuration Reference*
- *RealView Development Suite Getting Started Guide*
- *ARM DSTREAM and RVI Using the Debug Hardware Configuration Utilities*
- *RealView Development Suite Real-Time System Models User Guide*
- *RealView ARMulator ISS User Guide*
- the following documents provided with your VSTREAM product:
  - *VSTREAM Client User Guide*
  - *VSTREAM Transactor Integration and User Guide*
- Carbon SoC Designer Plus, `http://carbondesignsystems.com/SocDesignerPlus.aspx`.

1.2.4 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
- desktop settings, for example all windows and views that are currently displayed.

If you choose to, you can also enable auto saving of breakpoints. This feature saves any breakpoints you have set for an image when that image is unloaded.

See also

- the following in the *RealView Debugger User Guide*:
  - *Enabling the auto save breakpoints feature* on page 11-12.

1.3 Multiprocessor debugging

The RealView Debugger enables you to debug multiprocessor applications. The processors can be all hardware, all simulated, or a mixture of both.

RealView Debugger supports such multiprocessor debugging by maintaining connections to multiple targets through one or more Code windows. When working with multiple processors, you can use one Code window to cycle through the connected targets, or multiple Code windows to view different targets.

1.4 Environment variables used by RealView Debugger

Table 1-1 shows the environment variables use by RealView Debugger.

Table 1-1 Main RVDS environment variables

Environment variable	Setting
ARMROOT	Your installation directory root (*install_directory*). The default is C:\Program Files\ARM.
ARMCONF	Used to locate the RVISS configuration files: *install_directory*\RDI\armperip\...\...; *install_directory*\RVARMulator\MPCore\ARMulator\...\...\...\win_32-pentium; *install_directory*\RVARMulator\v6ARMulator\...\...\win_32-pentium; *install_directory*\RVARMulator\ARMulator\...\...\win_32-pentium
ARMDLL	Used to locate the RVISS DLL files: *install_directory*\RVARMulator\MPCore\ARMulator\...\...\...\win_32-pentium; *install_directory*\RVARMulator\v6ARMulator\...\...\win_32-pentium; *install_directory*\RVARMulator\ARMulator\...\...\win_32-pentium; *install_directory*\RDI\rdimsvr\...\...\win_32-pentium
ARMLMD_LICENSE_FILE	The location of your ARM RealView license file. See the *FLEXnet for ARM Tools License Management Guide* for information on this environment variable.
ARM_RTSM_PATH	Location of the RTSMs provided with RVDS, that are used by RealView Debugger: *install_directory*\SysGen\PVExamples\...\external\lib\Win32_VC2005\Release
ISSM_ARM_CORTEXDLL	The location of the ISSMs: *install_directory*\ISSModel\Cortex\...\win_32-pentium
RVD_CS_ETM_LOCK	Set to the hexadecimal value that represents the password required to unlock the ETM hardware. See *ETM with Lock Access implemented causes APB lock up* on page 4-2 in the *RealView Debugger Trace User Guide* for more details.
RVD_FLASH_BASE	The location of the Flash files for supported development boards. The default is: *install_directory*\RVD\Flash\...\windows
RVD_SOCD_CONN_TIMEOUT_SECS	The timeout value, in seconds, to be used when attempting to connect to a Soc Designer target. The default timeout value is 60 seconds.
RVDEBUG_HLPPATH	The RealView Debugger online help files: *install_directory*\Documentation\RVD\...\release\windows\onlinehelp
RVDEBUG_HOME	Enables you to override the default location for the RealView Debugger home directory.
RVDEBUG_INSTALL	The RealView Debugger executables: *install_directory*\RVD\Core\...\win_32-pentium
RVDEBUG_SDK	The location of th e supported OS awareness files: "%RVDEBUG_INSTALL%\sdk"
RVDEBUG_SHADOW_DIR_BASE	Use this to override the directory used to provide a working copy of the RealView Debugger shodow base directory. The default location is: C:\Documents and Settings*userID*\Local Settings\Application Data\ARM\rvdebug*version*\shadowbase
RVDEBUG_SHADOW_DIR_ETC	Use this to override the working copy of the RealView Debugger etc directory. The default location is: "%RVDEBUG_SHADOW_DIR_BASE%\etc"

1.5 The RealView Debugger documentation suite

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

- *RealView Debugger User Guide*
- *RealView Debugger Target Configuration Guide*
- *RealView Debugger Trace User Guide*
- *RealView Debugger RTOS Guide*
- *RealView Debugger Command Line Reference Guide*.

The following description explains how you might use the books:

1. For a comprehensive description of the debugging features available in RealView Debugger, see the *RealView Debugger User Guide*. This describes, in detail, how to debug your images and how to configure RealView Debugger to customize your working environment. It also describes the features available for debugging multiple targets.
2. *RealView Debugger Target Configuration Guide* describes how to customize existing Debug Configurations that are set up in the base product, and how to create your own custom Debug Configurations. It also describes how to create the files required to program Flash.
3. If you have the appropriate trace hardware, you can perform RealView Debugger tracing, which is described in the *RealView Debugger Trace User Guide*.
4. If you have the appropriate plug-ins, you can debug OS applications using the OS-awareness features in RealView Debugger. These features are described in the *RealView Debugger RTOS Guide*.
5. If you want to use the RealView Debugger *Command Line Interface* (CLI) to control your debugging tasks, the *RealView Debugger Command Line Reference Guide* provides a detailed description with examples of:
   - CLI commands
   - predefined macros
   - keywords.

See also:

- the following in the *RealView Debugger User Guide*:
  - Appendix E *RealView Debugger on Red Hat Linux* for information on using RealView Debugger on Red Hat Linux platforms.
- the installation notes delivered with your product for details on installing RealView Debugger.

Chapter 2
Getting Started with RealView Debugger

This chapter gives step-by-step instructions to start debugging an application with RealView® Debugger. It is in the form of a tutorial, where each task assumes that you have performed the preceding tasks. This chapter contains the following sections:

2.1 How to use the tutorial

The tutorial describes the main tasks required to begin debugging an application. It assumes that you already have an image to debug. The classic Dhrystone benchmark is provided as an example with *RealView Development Suite* (RVDS), which has a prebuilt image. This image is used in the tutorial.

The Dhrystone example is installed in the directory:

install_directory\RVDS\Examples\...\...\...\dhrystone

The main ...\dhrystone directory contains:

- the subdirectories ...\Debug and ...\Release containing prebuilt debug and release images of dhrystone.axf
- C source files required to build the image
- build files, for building the image with ARM Compiler toolchain supplied with RVDS.

Before you start the tutorial, make a copy of the Dhrystone example, and use your copy during the tutorial.

See also:

- *Starting the tutorial* on page 2-3.

2.2 Starting the tutorial

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 own directory:

   `C:\MyExamples\Tutorial`

   This is the tutorial project *base directory*.

2. Copy the following files from the examples directory, that is `...\dhrystone`, into your new tutorial directory:
   - C source files `dhry.h`, `dhry_1.c`, and `dhry_2.c`
   - the build files:
     - `dhry.bat`
     - `dhry.mk`.
   - the `Debug` directory.

You can complete this tutorial using the files you have copied. You do not have to change any of these files or amend any configuration files.

2.3 Starting RealView Debugger

To start RealView Debugger:

- on Windows, select:

  Start → All Programs → ARM → RealView Development Suite v4.1 → RealView Debugger v4.1

- on Red Hat Linux, select:

  Start Menu → Programs → ARM → RealView Development Suite v4.1 → RealView Debugger v4.1

The first time you run RealView Debugger after installation, it creates a unique working 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. The location depends on your host platform:

- On Windows, the default location of the home directory is in:

  `"%APPDATA%\ARM\rvdebug\`*`version`*`"`

  If a user ID is not available, then RealView Debugger creates a general-purpose directory called `owner`.

- On Red Hat Linux, the location is in `~/rvd`.

The main RealView Debugger window is known as the Code window. An example of the default layout of the Code window is shown in Figure 2-1 on page 2-5.

Note

You might want to position and resize the main RealView Debugger window to your requirements. Also, resize the views as required. RealView Debugger remembers the position and size of the window and views between debugging sessions.

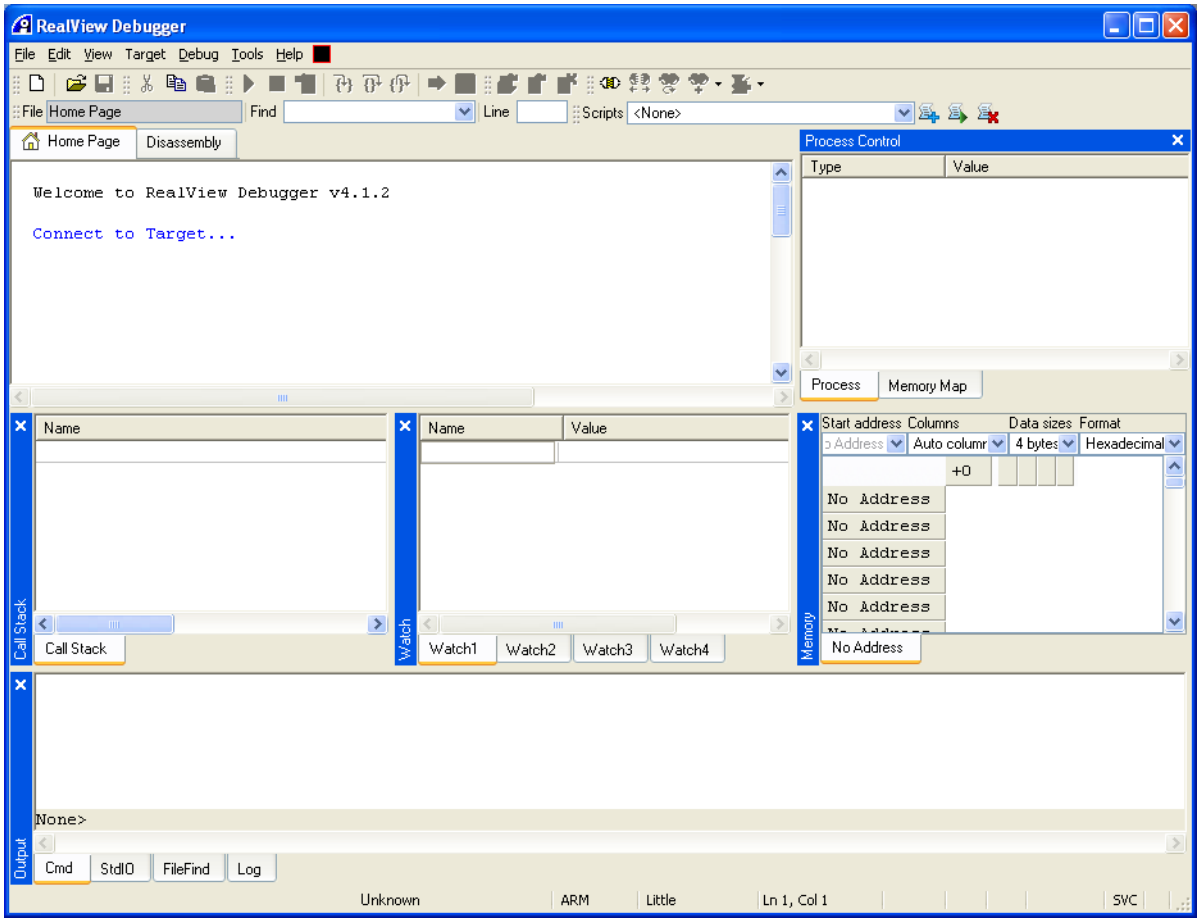

Figure 2-1 Default Code window

2.4 Connecting to a debug target

Before you can load an image to a debug target, you must connect to the target. You access connections using the Connect to Target window.

Note

Be aware that the default connection mode stops the target.

See also:

- *How to open the Connections window*
- *Elements of the Connect to Target window*
- *Making a connection* on page 2-8.

2.4.1 How to open the Connections window

Click **Connect to Target...** in the **Home Page** to open the Connect to Target window. Figure 2-2 shows an example:

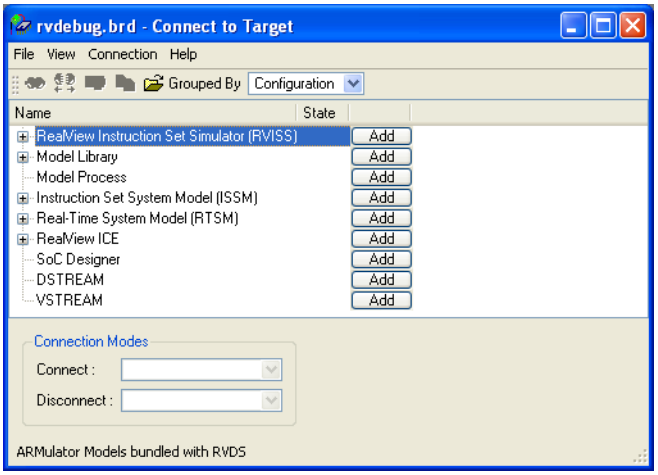

Figure 2-2 Connect to Target window

See also

- *Elements of the Connect to Target window*
- the following in the *RealView Debugger User Guide*:
  - Chapter 3 *Target Connection*.

2.4.2 Elements of the Connect to Target window

The Connect to Target window includes a tree control, which comprises:

Debug Interface

This group shows the type of debug target interface used to support the connection. For DSTREAM or RealView ICE, this is the ARM® JTAG debug tool for embedded systems.

Debug Configuration

A Debug Configuration identifies the targets that are available on the associated development platform. A Debug Configuration enables you to set up a custom debugging environment.

Target grouping

You can display the targets using the following groupings:

Target All the targets are shown as a single list in each Debug Interface. The targets are listed in the order of the associated Configuration name. Figure 2-3 shows an example:

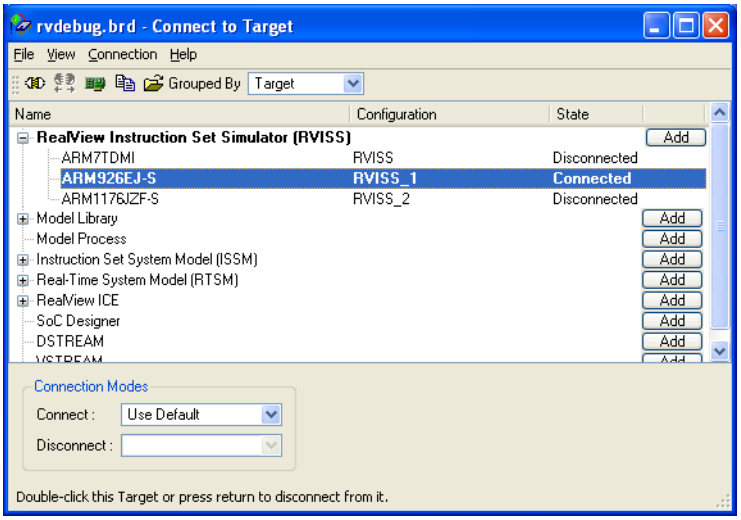

Figure 2-3 Example Connect to Target window (Target group)

Configuration

The targets are listed under the name of the associated Debug Configuration. Figure 2-4 shows an example:

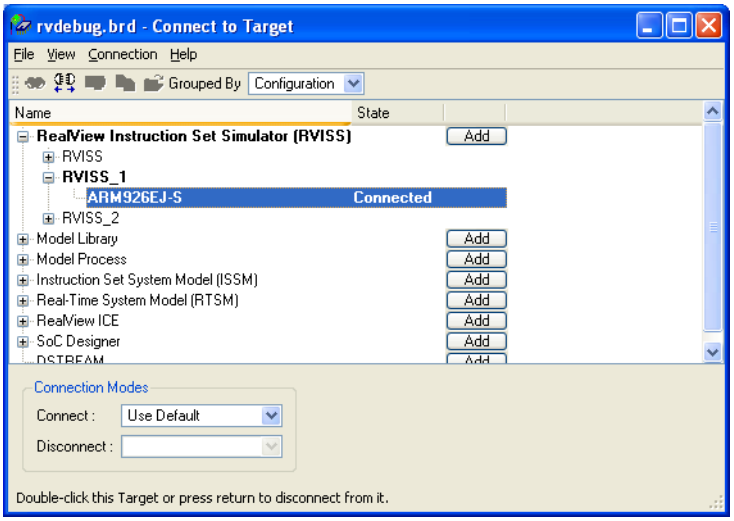

Figure 2-4 Example Connect to Target window (Configuration group)

You can have multiple Debug Configurations for the same development platform. If you have multiple debugging platforms, then you must create a separate Debug Configuration for each platform. A Debug Configuration enables you to set up a custom debugging environment.

Note

A Synchronization Control window is also available for debugging multiprocessor applications.

See also

- the following in the *RealView Debugger User Guide*:
  - Chapter 3 *Target Connection*
  - Chapter 7 *Debugging Multiprocessor Applications*.

2.4.3 Making a connection

This tutorial uses the `RealView Instruction Set Simulator (RVISS)` Debug Interface. However, if you want to connect to a hardware target, first make sure your debug hardware and target system are powered on, and configured as described in your debug hardware User Guide.

To connect to a debug target, do the following:

1. Select **Target → Connect to Target...** from the Code window main menu to open the Connect to Target window. Figure 2-2 on page 2-6 shows an example.
2. Select **Configuration** from the Grouped By drop-down list.

   Note

   It is recommended that you add and configure a Debug Configuration using the **Configuration** grouping.
3. Expand the `RealView Instruction Set Simulator (RVISS)` Debug Interface. The `RVISS`, `RVISS_1`, and `RVISS_2` Debug Configurations are available.
4. Expand the `RVISS` Debug Configuration to show the target processor available for the configuration. For the `RVISS` Debug Configuration, the target is called `ARM7TDMI`.
5. Double-click on the `ARM7TDMI` target processor to connect.

With the connection established, your Code window is updated:

- The Code window title bar is updated with the name of the current connection. In this example, the connection name is `ARM7TDMI@RVISS`. The connection name is also used in floating views to identify the connection to which the view contents relate.
- The color box is updated with a color for the connection. This color is also used in floating views to identify the connection to which the view contents relate.
- The **Home Page** is updated as follows:
  - The connection is added to a Recent Connections... list. For this example, the connection entry is `ARM7TDMI@RVISS (connected)`.

    Note

    RealView Debugger adds an entry to this list each time you connect to a different target, up to 10 a maximum of ten connections.
  - A **Load Image...** entry is added, to enable you to load an image to the target.
- The **Cmd** tab of the Output view displays connection details.

- Views are updated with debug information, for example the Process Control view shows process information for the connected target.

Figure 2-5 shows an example:

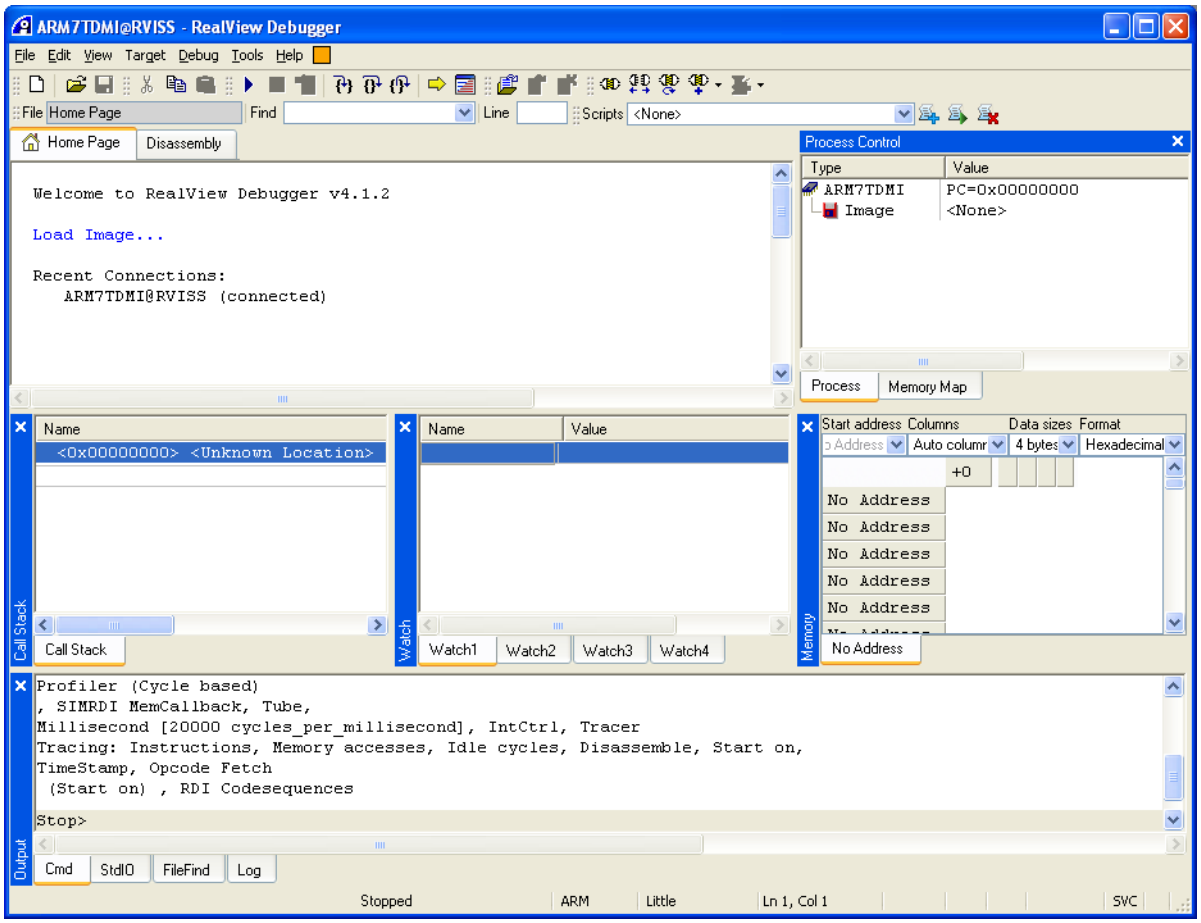

Figure 2-5 Home Page with a connection

The default configuration files installed as part of the base product enable you to connect to an ARM7TDMI®, ARM926EJ-S™, and an ARM1176JZF-S™ simulated processor using *RealView ARMulator® ISS* (RVISS).

See also

- the following in the *RealView Debugger User Guide*:
  - Chapter 3 *Target Connection*
  - Chapter 7 *Debugging Multiprocessor Applications*.

2.5 Loading an image ready for debugging

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

See also:

- *Loading an image directly*
- *Loading multi-image applications to a single debug target* on page 2-11.

2.5.1 Loading an image directly

To load an image directly to your debug target:

1. Click **Load Image...** in the **Home Page** to open the Select Local File to Load dialog box.

   Note

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

2. Locate the `dhrystone.axf` image in your `Tutorial` directory (see *Starting the tutorial* on page 2-3):

   `C:\MyExamples\Tutorial\Debug`

3. Click **Open**.

   The image is loaded to the debug target.

Note

In the Code window Text Coloring and source code line numbering are enabled by default.

When you load an image, the debugger updated the Code window with the image details:

- Inserts the source filename, for the current context, in the File field of the Find toolbar.
- Inserts a blue hyperlink for the loaded image in a **Recent Images...** list on the **Home Page**. You can click this hyperlink if you want to load the image in future.

  Note

  RealView Debugger adds an entry to this list each time you load an image, up to 10 a maximum of ten images.

- Updates the Code window views as appropriate, for example, it updates the process information in the Process Control view.
- Displays the load command, used by RealView Debugger to load the image, in the **Cmd** tab in the Output view.

Figure 2-6 on page 2-11 shows an example:

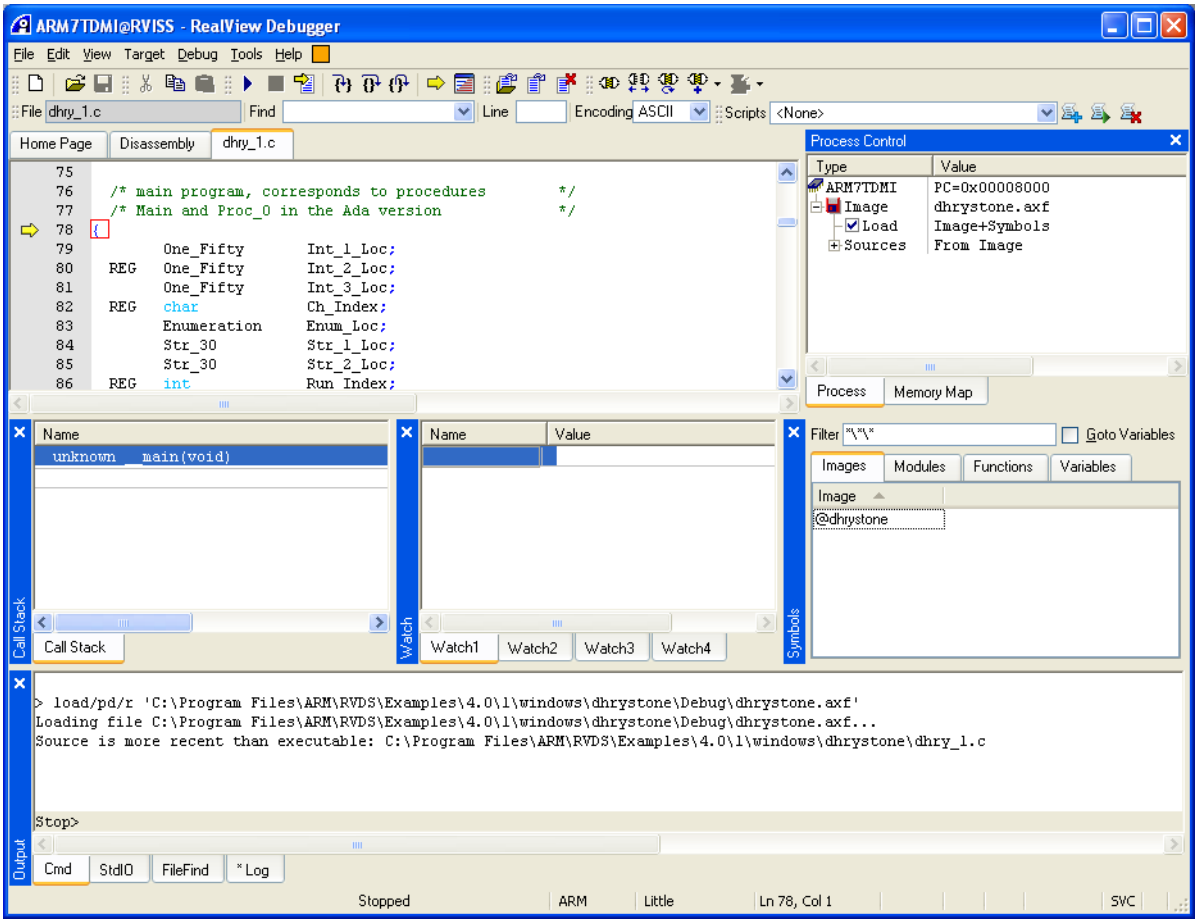

Figure 2-6 Code window with image loaded

If an image is compiled to generate debug information, that is using the `--debug` switch, loading the image enables RealView Debugger to gather debug information about the image and the associated source files. In the `dhrystone` example, the source file `dhry_1.c` is opened into a source tab when you load the image because it has the context.

See also

- the following in the *RealView Debugger User Guide*:
  - Chapter 4 *Loading Images and binaries*.

2.5.2 Loading multi-image applications to a single debug target

If your application contains multiple images that run on a single target, you can load all the images to the target. Each image must not overlap any of the other images.

To load a multi-image application:

1. Load the first image.
2. Load any subsequent images, and make sure that the **Replace Existing File(s)** check box is not selected on the Select Local File to Load dialog box.

See also

- *Loading an image directly* on page 2-10

- the following in the *RealView Debugger User Guide*:
  - Chapter 4 *Loading Images and binaries*
  - Chapter 7 *Debugging Multiprocessor Applications*

2.6 Setting a simple breakpoint

A breakpoint enables you to stop image execution at a point of interest, so that you can examine various parts of your application at that point.

To set a simple breakpoint:

1. Select the `Sources` entry in the Process Control view.
2. Type the characters `dh`. The source file `dhry.h` is selected.
3. Double-click on the source file `dhry_1.c`. The source file is opened.
4. Click **Locate PC** in the Debug toolbar. The PC is located in the source file, shown in Figure 2-7:

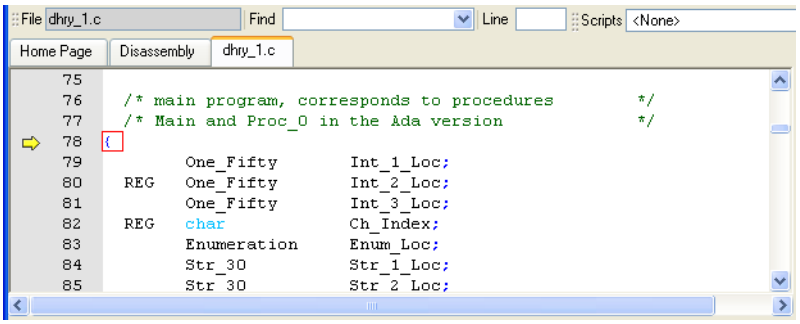

Figure 2-7 PC located in a source file

5. Scroll down until line 149 is visible.
6. Double-click in the gray margin at line 149. A simple breakpoint is set at line 149, as indicated by a red circle shown in Figure 2-8:

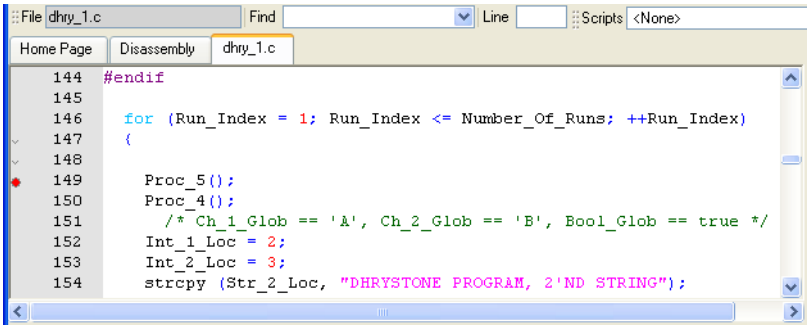

Figure 2-8 Simple breakpoint set

   Because the breakpoint location is in RAM, RealView Debugger sets a software breakpoint.

7. Run the image.

See also:

- *Running the image* on page 2-14
- the following in the *RealView Debugger User Guide*:
  - Chapter 11 *Setting Breakpoints*.

2.7 Running the image

To run an image, either:

- select **Debug** → **Run** from the Code window main menu
-  click the **Run** button on the Debug toolbar.

The Code window is updated as follows:

- the processor status changes to `Running`, and includes an animated progress indicator
- the process information in the Process Control view is updated to show that the image is running
- the **StdIO** tab of the Output view is selected, and application messages and prompts are displayed
- the CLI prompt in the **Cmd** tab of the Output view changes to `Run>`.

See also:

- *Continuing the tutorial*.

2.7.1 Continuing the tutorial

If you are running the Dhrystone example described in *Setting a simple breakpoint* on page 2-13, then continue this tutorial as follows:

1. Run the `dhrystone.axf` image:
   a. Click the **Run** button to start execution.
   b. Enter the required number of runs, for example `20000`.

   When the breakpoint that you set in *Setting a simple breakpoint* on page 2-13 is reached:
   - A red box is drawn around the source line to show that the PC is pointing to this location.
   - Messages are displayed in the **Cmd** tab in the Output view, to show what caused the execution to stop, and the location. In this case:

     ```
     Stopped at 0x00008480 due to SW Instruction Breakpoint
     Stopped at 0x00008480: DHRY_1\main Line 149
     ```

2. You can now do the following:
   - Examine any variables that are in scope. For example, double-click on the variable `Int_2_Loc`, then drag and drop it onto the Watch view.
   - Step through the image. For example, select **Step Into** from the Code window **Debug** menu.
   - Click the **Run** button to restart execution until the breakpoint is reached again.
   - Double-click on the red breakpoint indicator to clear the breakpoint, then click the **Run** button to restart execution.

See also

- *Setting a simple breakpoint* on page 2-13
- the following in the *RealView Debugger User Guide*:
  — Chapter 8 *Executing Images*.

2.8 Unloading an image

RealView Debugger automatically unloads an image from a debug target when you:

- disconnect from the debug target
- exit RealView Debugger.

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 do not have to unload an image from a debug target before loading a new image for execution. Open the Select Local File to Load dialog box and make sure that the **Replace Existing File(s)** check box is selected.

See also:

- *How to explicitly unload an image*.

2.8.1 How to explicitly unload an image

You can unload an image by using the Process Control view. To do this:

1. If the Process Control view is hidden, select **View → Process Control** from the default Code window main menu.

   Note

   If you still have the default views visible in the Code window, then you do not have to do this step.

2. Do one of the following:
   - Click the **Unload an image file** button on the Debug toolbar.
   - Right-click on either the `Image` or `Load` entry to display the context menu, then select **Unload** from the context menu.
   - In the Process Control view deselect the check box associated with the `Load` entry.

Note

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

To remove image details completely, right-click on either the `Image` or `Load` entry in the Process Control view and select **Delete Entry** from the context menu.

See also

- *Loading an image directly* on page 2-10
- the following in the *RealView Debugger User Guide*:
  - Chapter 4 *Loading Images and binaries*.

2.9 Disconnecting from a target

You can disconnect from the current target in one of the following ways:

- Select **Target → Disconnect** from the Code window main menu.

  This immediately disconnects the connection shown in the Code window title bar.

- Click the **Disconnect** button on the Connect toolbar.

  This immediately disconnects the connection shown in the Code window title bar.

- Do the following:

  1. Select **Target → Connect to Target...** from the Code window main menu to open the Connect to Target window.
  2. Right-click on the required connection to display the context menu
  3. Select **Disconnect** from the context menu.

You can also click the **Connect** button on the Connect toolbar to open the Connect to Target window.

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 view contents.

Disconnecting changes the Processor status to `Unknown`, and the CLI prompt changes to `None>`.

Note

You do not have to disconnect from a target before you close down RealView Debugger.

See also:

- *Exiting RealView Debugger* on page 2-17
- the following in the *RealView Debugger User Guide*:
  - Chapter 3 *Target Connection*.

2.10 Exiting RealView Debugger

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

If you chose to save the current state of your Code window and connection when you exit RealView Debugger, then next time you start RealView Debugger:

- the Code window appears in the same state as your previous debugging session
- RealView Debugger automatically attempts to reconnect to a debug target, if it was previously connected when you last exited RealView Debugger.

See also:

- *Closing down RealView Debugger*
- *Reconnecting to a target*
- *Storing breakpoints for an image when exiting RealView Debugger*.

2.10.1 Closing down RealView Debugger

To exit RealView Debugger:

1. Select **File → Exit** from the Code window main menu to open the Exit dialog box.
2. If you do not want to be prompted again on exit, select **Do Not Show This Dialog Again**.
3. Click **Yes** to close the Exit dialog box and close down RealView Debugger.

   If any connections are still established, RealView Debugger stores the connection details, disconnects from the connected targets, and exits. When you next start RealView Debugger, these connections are re-established.

See also

- *Reconnecting to a target*
- *Storing breakpoints for an image when exiting RealView Debugger*.

2.10.2 Reconnecting to a target

RealView Debugger saves all open connections in the current workspace if you close down without disconnecting first. This means that RealView Debugger tries to reconnect when you next open the workspace. However, this might fail if the connection or Debug Interface status has changed, for example if your RealView ICE interface unit has been disconnected.

If RealView Debugger reconnects successfully, the connection mode specified in the target configuration file is used by default.

See also

- the following in the *RealView Debugger User Guide*:
  - Chapter 3 *Target Connection*.

2.10.3 Storing breakpoints for an image when exiting RealView Debugger

If you exit, and you have enabled auto save breakpoints, RealView Debugger when an image is loaded that has breakpoints set, then the breakpoints are stored for later use. The breakpoints are automatically loaded when you next load the image.

Be aware of the following:

- Because there is an open connection associated with the image, the connection is also saved in the current workspace.

- When you next start a RealView Debugger session, and the saved connection is established, the image is not loaded automatically.

See also

- *Reconnecting to a target* on page 2-17
- the following in the *RealView Debugger User Guide*:
  - *Enabling the auto save breakpoints feature* on page 11-12.
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *QUIT* on page 2-217
  - *RVDCONTEXT* on page 2-233
  - *UNLOAD* on page 2-316.

2.11 Cleaning up after the tutorial

If you created the tutorial project directory described in *Starting the tutorial* on page 2-3 you might want to remove it from your workstation. Do this in the usual way.

2.12 Localizing the RealView Debugger interface

By default, the RealView Debugger interface is configured for the US English language. However, you can configure the language for the RealView Debugger interface to Japanese.

If you do not want to change the internationalization settings, then you can skip this section.

See also:
- *Font recommendations*
- *Procedure summary*
- *Configuring the internationalization settings*
- *Configuring the views* on page 2-21.

2.12.1 Font recommendations

If you are changing the language to Japanese then it is recommended that you also change the font to **MSGothic** or **MSMincho**.

See also
- *Configuring the internationalization settings*.

2.12.2 Procedure summary

To localize the RealView Debugger interface, you must:

1. Configure the internationalization settings.
2. Configure the views that display normal text to show the text in the correct text encoding.

Images built from C or C++ sources must be compiled with the `Multibyte_chars` project setting enabled (that is, the `--multibyte_chars` compiler option). You can optionally set the `Multibyte_locale` project setting (that is, the `--locale` compiler option). To build your image either:
- create an Eclipse project
- create a makefile.

See also
- *Configuring the internationalization settings*
- *Configuring the views* on page 2-21
- the following in the *RealView Debugger User Guide*:
  - Chapter 3 *Target Connection*.
- *ARM® Compiler toolchain Introducing the ARM® Compiler toolchain*
- *ARM® Workbench IDE User Guide*.

2.12.3 Configuring the internationalization settings

You can set the internationalization settings using:

- Global settings, if you have a shared RealView Debugger environment, and you want all users to use the same configuration. To do this, select **Tools → Options...** from the Code window main menu to open the Options window.
- Individual workspace settings, for individual users. To do this, select **File → Workspace → Workspace Options...** from the Code window main menu to open the Workspace Options window.

You can use the following procedure to edit the internationalization settings for both global and workspace settings:

1. Expand the following groups in the left pane:
   - `...\rvdebug.ini`
   - `ALL`
   - `Text`
2. Select the `Internationalization` group in the left pane.
3. Change the settings to the required values:
   a. Right-click on the `Enabled` setting, and select **True** from the context menu
   b. Right-click on the `Language` setting, and select the required language from the context menu, **English** or **Japanese**.
   c. Right-click on the `Default_encoding` setting, and select the required encoding from the context menu, either **ASCII**, **UTF-8**, or **Locale**.
4. Select the `Font_information` group in the left pane.
5. Change the settings to the required values:
   a. Right-click on the `Pane_font` setting, and select **Edit as font...** from the context menu to open the Font dialog box.
   b. In the Font dialog box, change the font to that required for your language.
   c. Set up the remaining font settings as required. For the Japanese language, select either **MSGothic** or **MSMincho**.
   d. Click **OK** to close the Font dialog box.
6. Select **File** → **Save and Close** from the menu to save your changes and close the settings window.
7. Select **File** → **Exit** from the Code window main menu to exit RealView Debugger.
8. Start RealView Debugger again.

   In the Code window, an `Encoding` field is displayed in the Find toolbar, shown in Figure 2-9:

Figure 2-9 Find toolbar with Encoding field

See also

- *Starting RealView Debugger* on page 2-4.

2.12.4 Configuring the views

To display messages and variable contents correctly in the Code window, you must set the encoding format to the format you specified in the settings window. You can change the format only for the following elements in the Code window:

- **Disassembly** tab, source code tabs, and text searches
- Output view, all tabs
- individual variables in a Watch view tab
- individual variables in the **Locals** tab and **Statics** tab of the Locals view.

Changing the encoding format for the source code view and searches

To change the encoding format for the source code view, click the down arrow in the Encoding field on the Find toolbar, and select the required encoding, for example, **UTF-8**.

This also enables you to search your sources for multibyte text that matches the selected format.

Changing the encoding format for the Output view

To change the encoding format for the Output view:

1. Right-click in the Output view to display the context menu.
2. Select **Format...** from the context menu to open the List Selection dialog box.
3. Select the required encoding from the dialog box, for example, **UTF-8**.
4. Click **OK**.

Changing the encoding format for individual Watch view entries

To change the encoding for an individual Watch view entry:

1. Right-click on the Watch entry to display the context menu.
2. Select **Format...** from the context menu to open the List Selection dialog box.
3. Select the required encoding from the dialog box, for example, **UTF-8**.
4. Click **OK**.

Changing the encoding format for individual Locals view entries

To change the encoding for an individual Locals view entry in either the **Locals** tab or the **Statics** tab:

1. Select **View** → **Locals** from the Code window main menu to open the Locals view.
2. Select the required tab in the Locals view, for example **Locals**.
3. Right-click on an entry in the tab to display the context menu.
4. Select **Format...** from the context menu to open the List Selection dialog box.
5. Select the required encoding from the dialog box, for example, **UTF-8**.
6. Click **OK**.

See also

- *Configuring the internationalization settings* on page 2-20.

2.13 Saving a debugging 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, and connecting automatically to specific targets.

See also:
- *Workspace*
- *Startup file*
- *History file*.

2.13.1 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.

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.

See also:
- *Startup file*
- *History file*.

2.13.2 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 view format options.

By default, this file is called `rvdebug.sav`. The first time you exit 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.

See also
- the following in the *RealView Debugger User Guide*:
  - Chapter 17 *Configuring Workspace Settings*.

2.13.3 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.

- Data entries examined in the Code window during debugging.
- The Set Directory and Set File lists used in the various open dialog boxes, such as the Select File to Open or Select File to Include Commands from dialog boxes.
- Up to 32 personal favorites such as variables, data values, and breakpoints.

RealView Debugger creates the history file when you carry out any of these operations for the first time, and then exit RealView Debugger. By default, the file is called `exphist.sav` and is stored in your home directory. The file is updated at the end of all subsequent debugging sessions.

Note

If you are using RealView Debugger on Red Hat Linux, the history file is only created if you create and save a favorite, for example a breakpoint.

See also

- the following in the *RealView Debugger User Guide*:
  - Appendix E *RealView Debugger on Red Hat Linux*.

Chapter 3
Changes to RealView Debugger

This chapter describes the changes between RealView® Debugger v4.1 SP2 and the RealView Debugger v4.1 SP1 release. It contains the following sections:

- *Debug target support* on page 3-2
- *GUI changes* on page 3-3
- *Changes to CLI commands* on page 3-4
- *Deprecated features* on page 3-5.

If you are using RealView Debugger on Red Hat Linux, see Appendix E *RealView Debugger on Red Hat Linux* in the *RealView Debugger User Guide*.

3.1 Debug target support

The following processors, models, and boards are supported:

Processor support

The following additional processors are supported:

- Cortex-A5 multiprocessor trace
- Cortex-M4
- Cortex-R5, including multiprocessor trace.

Model support

The following additional *Real-Time System Models* (RTSMs) are supported:

- Cortex-A5 MPx1
- Cortex-A5 MPx2
- Cortex-M4.

Connections to RTL models are supported through the VSTREAM Debug Interface.

Board support

Board-Chip Definition (BCD) files and corresponding Flash methods, where appropriate, are supported for the following boards:

- Versatile Express Cortex-A9 based processor
- OMAP 4430 Board basics.

See also:

- the following in the *RealView Debugger User Guide*:
  - Chapter 3 *Target Connection*.
- the following in the *RealView Debugger Target Configuration Guide*:
  - *About connection configuration* on page 1-2
  - *Summary of supplied BCD files* on page 1-19
  - *Summary of files used to program Flash on supported development platforms* on page 6-2.

3.2 GUI changes

The following changes have been made to the RealView Debugger documentation:

- Multiprocessor models are inherently synchronized. For this reason, you cannot set up synchronzed execution for more than one processor from a multiprocessor model that is a Model Library, Model Process, ISSM, RTSM, or SoC Designer model. When you select a simulated processor from any of these simulation Debug Interfaces, the **Sync** check boxes on the **Execution** tab of the Synchronization Control window are disabled for the other processors in these interfaces.

  Note

  If you have connections to targets in these interfaces, use caution when setting up synchronized execution with the `SYNCHEXEC` command.

- You can choose to save any breakpoints that are set when you unload an image.
- The Code window status bar includes a new Processor mode field that indicates the value of the CPSR Mode field.

See also:

- the following in the *RealView Debugger User Guide*:
  - *Code window* on page 1-4
  - *Synchronized start and stop operations, cross-triggering, and skid* on page 7-6
  - *Enabling the auto save breakpoints feature* on page 11-12.
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *SYNCHEXEC* on page 2-271.

3.3 Changes to CLI commands

The `RVDCONTEXT` is supported. It allows you to enable or disable the auto save breakpoints feature.

See also:

- the following in the *RealView Debugger User Guide*:
  - *Enabling the auto save breakpoints feature* on page 11-12
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *RVDCONTEXT* on page 2-233.

3.4 Deprecated features

The following features are deprecated:

- support for *RealView Instruction Set Simulator* (RVISS)
- support for ISSM
- support for the ARM10 family of processors.

See also:

- Chapter 3 *Changes to RealView Debugger*.

Appendix A
About Previous Releases

This appendix describes the major differences between the previous releases of RealView® Debugger. The changes are described in:

A.1 Changes between RealView Debugger v4.1 and v4.0 SP1

This section describes the changes between RealView Debugger v4.1 and the RealView Debugger v4.0 release. It contains the following sections:

- *RealView Debugger command line options*
- *Debug target support*
- *Debug Interface support* on page A-3
- *GUI changes* on page A-3
- *Changes to CLI commands* on page A-4
- *Trace, analysis, and profiling* on page A-4
- *Documentation changes* on page A-4
- *Deprecated features* on page A-5.

If you are using RealView Debugger on Red Hat Linux, see Appendix E *RealView Debugger on Red Hat Linux* in the *RealView Debugger User Guide*.

A.1.1 RealView Debugger command line options

The following `rvdebug` command-line options are available:

- `--help`
- `--version`.

See also:

- the following in the *RealView Debugger User Guide*:
  - *Starting RealView Debugger from the command line* on page 2-2.

A.1.2 Debug target support

The following processors, models, and boards are supported:

Processor support

The following processors are supported:

- Cortex-A5
- Marvell Sheeva 88SV581x-v7 PJ4.

CoreSight support

You can now connect to the CoreSight *Instrumentation Trace Macrocell* (ITM) device. This enables you to configure the CoreSight ITM using the Registers view.

Model support

The following *Real-Time System Models* (RTSMs) are supported:

- Cortex-A5
- Cortex-A9 Dual Core.

Board support

Board-Chip Definition (BCD) files and corresponding Flash methods, where appropriate, are supported for the following boards:

- Atmel AT91SAM9261-EK
- Atmel AT91SAM9263-EK
- Atmel AT91SAM9G45-EKES
- Atmel AT91SAM9RL-EK

- Icytecture iMX35 Starter board
- i.MX31
- i.MX31 LiteKit
- Freescale iMX25 PDK
- Freescale iMX27 LiteKit
- Texas Instruments Zoom OMAP34x-II Mobile Development Platform
- PBX-A9
- PHYTEC phyCORE-iMX35
- Samsung SMDK C100
- TMS320DM355
- `VAB926EJ-S.bcd` is replaced by:
  - `vpb926ej-s_256KB.bcd`
  - `vpb926ej-s_64KB.bcd`
- Zoran ZJP4100.

See also:

- the following in the *RealView Debugger User Guide*:
  - Chapter 3 *Target Connection*.
- the following in the *RealView Debugger Target Configuration Guide*:
  - *Summary of supplied BCD files* on page 1-19
  - *Summary of files used to program Flash on supported development platforms* on page 6-2.

A.1.3 Debug Interface support

RealView Debugger supports the following additional Debug Interface connections:

- DSTREAM
- Model Library
- Model Process.

Also, tooltips are available for the Debug Interfaces.

See also:

- the following in the *RealView Debugger User Guide*:
  - *Connect to Target window* on page 1-6
  - Chapter 3 *Target Connection*.
- the following in the *RealView Debugger Target Configuration Guide*:
  - Chapter 2 *Customizing a Debug Interface configuration*
  - Chapter 3 *Customizing a Debug Configuration*.

A.1.4 GUI changes

The following changes have been made to the RealView Debugger GUI:

- an **Unload** tool button has been added to the Image toolbar, as shown in Figure A-1:

Figure A-1 Image toolbar

- tooltips are available for Debug Interfaces in the Connect to Target window.

See also:

- Appendix A *About Previous Releases*.

A.1.5 Changes to CLI commands

This following CLI commands are supported:

- `CWD` to change the current working directory
- `PWD` to display the current working directory.

See also:

- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12.

A.1.6 Trace, analysis, and profiling

The major changes to the trace, analysis, and profiling features are:

- Support for ETM v3.5 has been added, to enable tracing from a Cortex-A5 processor.
- Tracing is supported on both Windows and Red Hat Linux in RealView Debugger v4.0 SP3 and later.
- Trace autoconfiguration scripts are provided for the following platforms:
  - AT91SAM9263
  - i.MX25
  - i.MX31
  - i.MX51
  - OMAP35xx
  - STM32E.

  The scripts are located at:

  `"%RVDEBUG_SHADOW_DIR_ETC%\platform"`

  To run a script at the RealView Debugger command-line, enter:

  `include '$RVDEBUG_SHADOW_DIR_ETC\platform\script_name.inc'`

A.1.7 Documentation changes

The following changes have been made to the RealView Debugger documentation:

- The *RealView Debugger User Guide* includes details on how to connect to targets through the new Debug Interface entries.
- The *RealView Debugger Target Configuration Guide* includes details on the new Debug Interface support and updated board support, including Flash programming support.

See also:

- the following in the *RealView Debugger Target Configuration Guide*:
  - Chapter 1 *Introduction*
  - Chapter 6 *Programming Flash with RealView Debugger*.

A.1.8 Deprecated features

The following features are deprecated:

- support for *RealView Instruction Set Simulator* (RVISS)
- support for ISSM
- support for the ARM10 family of processors.

See also:

- Appendix A *About Previous Releases*.

A.2 Changes between RealView Debugger v4.0 SP1 and v4.0

This section describes the changes between RealView Debugger v4.0 SP1 and the previous release, RealView Debugger v4.0. It includes:

- *RealView Debugger command line options in RealView Debugger v4.0 SP1*
- *Processor support in RealView Debugger v4.0 SP1*
- *Simulator Support in RealView Debugger v4.0 SP1*
- *Miscellaneous changes to the GUI in RealView Debugger v4.0 SP1*
- *Changes to CLI commands and predefined macros in RealView Debugger v4.0 SP1* on page A-7
- *Documentation changes in RealView Debugger v4.0 SP1* on page A-9
- *Deprecated features in RealView Debugger v4.0 SP1* on page A-9
- *Obsolete features in RealView Debugger v4.0 SP1* on page A-9.

A.2.1 RealView Debugger command line options in RealView Debugger v4.0 SP1

The command-line option `--cleanstart` is available when starting RealView Debugger. It enables you to start RealView Debugger with the default set of configuration files in your RealView Debugger home directory.

See also:

- the following in the *RealView Debugger User Guide*:
  - *Starting RealView Debugger from the command line* on page 2-2.

A.2.2 Processor support in RealView Debugger v4.0 SP1

The Cortex-M0 processor is supported.

See also:

- the following in the *RealView Debugger User Guide*:
  - Chapter 3 *Target Connection*.

A.2.3 Simulator Support in RealView Debugger v4.0 SP1

The Cortex-M0 *Instruction Set System Model* (ISSM) is supported.

See also:

- the following in the *RealView Debugger User Guide*:
  - Chapter 3 *Target Connection*.

A.2.4 Miscellaneous changes to the GUI in RealView Debugger v4.0 SP1

The following changes have been made to the RealView Debugger GUI:

- The source code tabs are positioned at the top of the source code view in the Code window.

- A new Diagnostic Log view is provided, and displays the messages that are generated during target connection. It is useful for debugging connections to targets that are under development.
- The name of the Set/Edit Tracepoint dialog box has changed to Create Tracepoint.
- The tabs on the Synchronization Control dialog box have been re-ordered, and tooltips have been added.
- The Memory view has been re-engineered. You can now view *Memory Management Unit* (MMU) page table descriptors.
- The Memory view context menu has an option to display a caching color scheme. Cached memory areas are colored according to:
  - the level, L1 or L2
  - the cache state, clean or dirty.
- The Classes view has been re-engineered. Library symbols are now hidden by default.
- The Connect to Target window has a menu option that enables you to show or hide the trace components in a Debug Configuration.
- The `RVD_CS_ETM_LOCK` environment variable has been added to enable the unlocking of ETM hardware on connection before powering up an ETM.
- You can perform copy and paste operations in the Output view.
- You can save and close multiple source files in a single operation.
- New settings for use when defining memory map blocks for Flash devices are provided in Connection Properties:
  - `Flash_write_mode`
  - `Flash_write_clock`.

  Corresponding settings are also available on the Create Map Entry and Edit Map Entry dialog boxes.

See also:

- the following in the *RealView Debugger User Guide*:
  - *Code window* on page 1-4
  - *Diagnostic Log view* on page 1-14
  - *Showing the trace components in the Connect to Target window* on page 3-36
  - *Viewing memory contents* on page 13-39
  - *Setting up a temporary memory map* on page 9-12
  - *Editing a memory map entry* on page 9-20
  - *Viewing memory contents* on page 13-39
  - Chapter 3 *Target Connection*.
- the following in the *RealView Debugger Trace User Guide*:
  - *ETM with Lock Access implemented causes APB lock up* on page 4-2.

A.2.5 Changes to CLI commands and predefined macros in RealView Debugger v4.0 SP1

This following changes have been made to CLI commands and macros:

- On Windows, a new user-specific default settings directory is created. The `RVDEBUG_SHADOW_DIR_ETC` environment variable points to this directory.

This directory is created when you first start RealView Debugger after installation, and the configuration files are copied to it from the installed settings directory.

- When you use the `INCLUDE` command to run a command script, RealView Debugger sets the `RVDEBUG_INCLUDE_BASE` environment variable to the location of that script. You can use this environment variable in your command script if required.

  The environment variable definition exists only for the current debugging session, and changes when you run additional command scripts. However, the environment variable does not change when `INCLUDE` is used in command script.

- The default behavior of the `BREAKINSTRUCTION` command has changed when attempting to set a software breakpoint in read-only memory. The operation now fails.

  However, a `failover` qualifier is added to the `BREAKINSTRUCTION` command, to allows a software breakpoint to be converted to a hardware breakpoint in read-only memory.

  Note

  This failover operation is not available from the Create Breakpoint dialog box.

- The `,a` qualifier is added to the following commands:
  - `CLEARBREAK`
  - `DISABLEBREAK`
  - `ENABLEBREAK`
  - `RESETBREAKS`.

  This qualifier enables you to identify a breakpoint by address, or multiple breakpoints within an address range.

- The following predefined macros have been added:
  - `atoi`
  - `atol`
  - `atoul`
  - `isdigit`
  - `islower`
  - `isprint`
  - `isspace`
  - `isupper`
  - `itoa`
  - `strtolower`
  - `strtoupper`
  - `strtrim`
  - `tolower`
  - `toupper`.

- You can format the value of substituted integer variables in a CLI command as either decimal or hexadecimal values.

See also:

- the following in the *RealView Debugger User Guide*:
  - *Considerations when running command scripts* on page 15-12
  - *Using variable substitution in commands within a macro* on page 16-6
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12

— Chapter 3 *RealView Debugger Predefined Macros*.

A.2.6 Documentation changes in RealView Debugger v4.0 SP1

The following changes have been made to the RealView Debugger documentation:

- The *RealView Debugger User Guide* includes details on using the new Diagnostic Log view.
- The new predefined macros are documented in the *RealView Debugger Command Line Reference Guide*.

See also:

- the following in the *RealView Debugger User Guide*:
  - *Diagnostic Log view* on page 1-14
  - *Examining details in the Diagnostic Log view* on page 3-64
- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 3 *RealView Debugger Predefined Macros*.

A.2.7 Deprecated features in RealView Debugger v4.0 SP1

The following features are deprecated:

- support for *RealView Instruction Set Simulator* (RVISS)
- support for ISSM
- support for the ARM10 family of processors.

See also:

- *Changes between RealView Debugger v4.0 SP1 and v4.0* on page A-6.

A.2.8 Obsolete features in RealView Debugger v4.0 SP1

Support for *Digital Signal Processors* (DSPs) is obsolete.

See also:

- *Changes between RealView Debugger v4.0 SP1 and v4.0* on page A-6.

A.3 Changes between RealView Debugger v4.0 and v3.1

This section describes the changes between RealView Debugger v4.0 and the previous release, RealView Debugger v3.1. It includes:

- *Processor support in RealView Debugger v4.0*
- *Simulator support in RealView Debugger v4.0*
- *Miscellaneous changes to the GUI in RealView Debugger v4.0*
- *Documentation changes in RealView Debugger v4.0* on page A-11
- *Deprecated features in RealView Debugger v4.0* on page A-11
- *Obsolete features in RealView Debugger v4.0* on page A-11.

If you are using RealView Debugger on Red Hat Linux, see Appendix E *RealView Debugger on Red Hat Linux* in the *RealView Debugger User Guide*.

A.3.1 Processor support in RealView Debugger v4.0

The following processors are supported:

- Cortex-A9 processor
- Faraday FA526, FA626, and FA626TE processors
- Marvell Feroceon 88FR101 and 88FR111 processors.

Note

A license is provided with *RealView Development Suite* (RVDS) Professional edition to support the Cortex-A9 processor.

A.3.2 Simulator support in RealView Debugger v4.0

The following changes have been made to simulated targets:

- The *Real-Time System Models* (RTSMs) provided with RVDS Professional edition are based around the Versatile Emulation Baseboard.

  Note

  The Cortex-A9 RTSM and a license for its use are provided with RVDS Professional edition.

- Cortex-M1 *Instruction Set System Model* (ISSM) has been updated.

A.3.3 Miscellaneous changes to the GUI in RealView Debugger v4.0

The following changes have been made to the RealView Debugger GUI:

- A new Connection Properties dialog box is provided, that enables you to easily customize a Debug Configuration. The dialog box contains the more commonly used settings.
- A new Comms Channel view is provided that enables you to preform writes and reads over the *Debug Communications Channel* (DCC).
- A `Real-Time System Model (RTSM)` Debug Interface is provided for you to create and customize Debug Configurations for RTSM targets.
- The Memory Map tab in the Process Control view shows the Secure World and Normal World memory maps for a target that supports TrustZone® technology.

- The `Memory_block` settings in Connection Properties include a `Tz_world` setting that enables you to specify the TrustZone world for the memory block.

See also:

- the following in the *RealView Debugger User Guide*:
  - Chapter 1 *RealView Debugger Features*.

A.3.4 Documentation changes in RealView Debugger v4.0

The following changes have been made to the RealView Debugger documentation:

- The *RealView Debugger User Guide* includes details on using the new Comms Channel view.
- All panes that show various debugging features are referred to as views. For example, the Memory pane is called the Memory view.

A.3.5 Deprecated features in RealView Debugger v4.0

The following features are deprecated in RealView Debugger v4.0:

- support for *RealView ARMulator® ISS* (RVISS)
- support for ISSM
- support for the ARM10 family of processors.

A.3.6 Obsolete features in RealView Debugger v4.0

Support for the `ARM Ltd. Direct Connect` Debug Interface to Versatile boards is obsolete in RealView Debugger v4.0.

A.4 Changes between RealView Debugger v3.1 and v3.0

This section describes the changes between RealView Debugger v3.1 and the previous release RealView Debugger v3.0. It includes:

- *Processor support in RealView Debugger v3.1*
- *Simulator support in RealView Debugger v3.1*
- *Command line options in RealView Debugger v3.1* on page A-13
- *Target connection and configuration in RealView Debugger v3.1* on page A-14
- *CoreSight support in RealView Debugger v3.1* on page A-15
- *Multiprocessor debugging in RealView Debugger v3.1* on page A-16
- *Cache debugging in RealView Debugger v3.1* on page A-17
- *Trace, analysis, and profiling in RealView Debugger v3.1* on page A-17
- *Changes to the views in RealView Debugger v3.1* on page A-18
- *Miscellaneous changes to the GUI in RealView Debugger v3.1* on page A-18
- *Changes to CLI commands and macros in RealView Debugger v3.1* on page A-20
- *Documentation changes in RealView Debugger v3.1* on page A-21
- *Deprecated features in RealView Debugger v3.1* on page A-21
- *Obsolete features in RealView Debugger v3.1* on page A-22.

A.4.1 Processor support in RealView Debugger v3.1

Cortex-A8 processor support has been added.

See also

- *Changes between RealView Debugger v3.1 and v3.0*.

A.4.2 Simulator support in RealView Debugger v3.1

The following simulated targets are supported:

- An MPCore™ simulated target is supported in RVISS. However, this simulates only a single processor.
- RVISS Debug Configurations are provided to simulate the following processors:
  - ARM7TDMI®
  - ARM926EJ-S™
  - ARM1176JZF-S™.

  You can connect to these without having to configure them.
- The following new *Instruction Set System Model* (ISSM) simulator are supported:
  - Cortex-M1
  - Cortex-R4.

  In addition, the Cortex-A8 and Cortex-M3 models have been enhanced.

  A single ISSM Debug Interface is provided, and is preconfigured to the Cortex-A8 processor. You can reconfigure this to any of the other Cortex processors. If you want to simulate additional processors, you must create a new ISSM Debug Configuration for each processor.
- SoC Designer models can be debugged. RealView Debugger starts SoC Designer Simulator automatically when you attempt to connect to a target in a SoC Designer model. Connections to other targets in the same model can then be established. You must purchase the SoC Designer software separately.

- *Real-Time System Models* (RTSMs) can be debugged. If you want to create your own RTSMs, you must purchase the RealView System Generator application.

See also

- *RealView Debugger Target Configuration Guide*.

A.4.3 Command line options in RealView Debugger v3.1

The command line options available when starting RealView Debugger have been modified and extended, as follows:

- All options accept a double hyphen prefix (`--`). However, you can still use the single hyphen prefix (`-`).
- The following options have been added to support project templates:
  - `--no_project`
  - `--project`
  - `--reinitialize_workdir`
  - `--workdir`.
- Many options have new names:
  - `--image` is an alias of `--exec`
  - `--journal` is an alias of `--jou`
  - `--no_logo` is an alias of `--nologo`
  - `--no_workspace` and `--no_aws` are aliases of `--aws=-`
  - `--script` is an alias of `--inc`
  - `--target` is an alias of `--init`
  - `--workspace` is an alias of `--aws`.

  The original names are still supported.

See also

- the following in the *RealView Debugger User Guide*:
  - *Starting RealView Debugger from the command line* on page 2-2.

A.4.4 Target connection and configuration in RealView Debugger v3.1

This section describes the improvements to target connection and configuration.

The Connection Control window

The Connection Control window has been renamed to the Connect to Target window, to match the menu option on the **Target** menu. In addition, the window has been re-engineered to simplify the connection and configuration of debug targets:

- Targets are grouped under the Debug Interface used to access them, such as RealView ICE. The following target groupings are available:

  Target Provides a list of all targets that are accessible through each type of Debug Interface.

  Configuration

  Lists the targets for each Debug Configuration that you create. A single Debug Configuration corresponds to a single development platform. The Debug Configurations are listed under each type of Debug Interface to which they relate.

  The chosen grouping persists between your debugging sessions.

- You can add, copy, rename, and delete Debug Configurations in a single operation, without having to use the Connection Properties window. However, the Connection Properties window is still available for customizing your Debug Configurations, such as setting up OS awareness and assigning *Board/Chip Definition* (BCD) files.

- If your development platform contains CoreSight™ components, then those components are included in the target list for the related Debug Configuration.

- The RVISS `localhost` connection entry has been removed. Connecting to RVISS targets is performed in a similar way to other targets. RVISS target connections show the simulated processor name, for example:

  ```
  ARM7TDMI@RVISS
  ```

See also

- *CoreSight support in RealView Debugger v3.1* on page A-15
- *Connecting to a debug target* on page 2-6
- the following in the *RealView Debugger User Guide*:
  - Chapter 3 *Target Connection*.
- the *RealView Debugger Trace User Guide*.

Recent connections list

As you establish connections to your targets, the connection name for that target is added to a recent connection list. This list shows the last 10 connections, which you can access from:

- A new **Home Page** tab in the Code window. With a single mouse click you can connect to frequently used targets without having to use the Connect to Target window.

  You must still use the Connect to Target window if you want to:
  - add, copy, rename, customize, and delete Debug Configurations
  - connect to, and disconnect from, multiple processors in a single operation
  - connect using a Connect Mode.
  - disconnect using a Disconnect Mode.

- A **Recent Connections** submenu on the Code Window **Target** menu.

Note

The **Home Page** tab also lists the last 10 images that you have loaded onto a processor of the same type as that for the current connection. This enables you to load an image with a single mouse click. These images can also be accessed from the **Recent Images** submenu on the Code Window **Target** menu.

A.4.5 CoreSight support in RealView Debugger v3.1

RealView Debugger provides support for development systems that contain CoreSight components.

Supported CoreSight Components

The following CoreSight components are supported:

- CoreSight Debug components:
  - A *Debug Access Port* (DAP) for connecting a Debug Interface unit to the target, and comprises a *JTAG Debug Port* (JTAG-DP) and *JTAG Access Port* (JTAG-AP).
  - An *Embedded Cross Trigger* (ECT) to specify cross triggering between multiple targets, and comprises *Cross Trigger Interface* (CTI) and *Cross Trigger Matrix* (CTM).
- CoreSight Trace components:
  - Sources generate trace data, and include the *Embedded Trace Macrocell*™ (ETM), *Program Flow Trace Macrocell* (PTM), and *AHB Trace Macrocell* (HTM).
  - Sinks are the end points for trace data on the SoC, and include the *Embedded Trace Buffer*™ (ETB™) and *Trace Port Interface Unit* (TPIU).
  - Links provide connection, triggering, and flow of trace data, and include the Trace funnel.
  - Control and access components configure, access, and control the generation of trace, and include DAP and ECT. They do not generate or process trace data.

  Note

  In this release, you can collect trace only from one ETM trace source in a multiple trace source system. Trace capture from an HTM or PTM is not supported.

Connecting to CoreSight components

CoreSight components, except for the CTM, are visible in the Connect to Target window for connections through a RealView ICE unit. The Configuration grouping shows a basic relationship between the CoreSight components and the runnable targets.

You can connect to a CoreSight component in the same way that you connect to a target processor. However, all execution-related features are disabled and the Code window shows a limited view as follows:

- registers in the Register view
- memory in the Memory view, if appropriate.

See also

- the following in the *RealView Debugger User Guide*:
  - *Configuring CoreSight embedded cross-triggering* on page 7-27
- the following in the *RealView Debugger Target Configuration Guide*:
  - *About customizing a DSTREAM or RealView ICE Debug Interface configuration* on page 2-3
- the following in the *RealView Debugger Trace User Guide*:
  - Appendix B *Setting up the Trace Software*.

A.4.6 Multiprocessor debugging in RealView Debugger v3.1

The following changes have been made to multiprocessor debugging:

- Support for cross trigger-related CoreSight components is available.
- RealView Debugger v3.0.SP1 provided a major enhancement to the Synchronization Control window and related CLI commands. These enhancements are documented in the RealView Debugger v3.1, and include the following changes:
  - The Synchronization Control window has been re-engineered. The synchronization and cross-triggering features are available on the following tabs:

    Actions This is a new tab, which enables you to synchronize the following actions:
    - image load, reload, and unload
    - reset the target processor
    - reset the PC
    - set the PC
    - load a binary file.

    Execution

    This tab enables you to synchronize the Step, Run, and Stop operations.

    Cross Triggering

    This tab enables you to set up cross triggering for multiple targets. Although this appears on the Synchronization Control window, synchronization and cross triggering can be set independently.
  - A new `SYNCHACTION` command is also available to specify the synchronization of actions that are available on the **Actions** tab of the Synchronization Control window.
  - The `OPTION,pendmode` command `OPTION,pendmode`, which enables you to specify the pend mode for CLI commands when debugging synchronized processors

See also

- *CoreSight support in RealView Debugger v3.1* on page A-15
- the following in the *RealView Debugger User Guide*:
  - Chapter 7 *Debugging Multiprocessor Applications*
- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 2 *RealView Debugger Commands*.

A.4.7 Cache debugging in RealView Debugger v3.1

Cache debugging is supported as described in the following sections:

Supported processors

Cache debugging support is provided for the following processors:

- ARM1136
- ARM1156
- Cortex-A8.

Memory view coloring scheme

The Memory view context menu has an option to display a caching color scheme. Cached memory areas are colored according to the level (L1 or L2) and state (clean or dirty) of the cache.

See also

- the following in the *RealView Debugger User Guide*:
  - *Viewing memory contents* on page 13-39.

Cache-related CLI commands and predefined macros

The following new cache-related CLI commands are available:

- `CACHEFIND`, which searches for an address within the cache
- `CACHEINFO`, which displays details about the cache
- `CACHELINE`, which prints information about a specific cache line.

The following new cache-related predefined macros are available:

- `cache_find_set`, which returns the set index associated with a specified address in the cache
- `cache_find_way`, which returns the way index associated with a specified address in the cache.

See also

- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 2 *RealView Debugger Commands*
  - Chapter 3 *RealView Debugger Predefined Macros*.

A.4.8 Trace, analysis, and profiling in RealView Debugger v3.1

The major change to the trace, analysis, and profiling features is the support for trace-related CoreSight components.

Note

The Analysis window supports trace capture from an ETM, ETB, CoreSight ETM, CoreSight ETB, and RVISS models. It does not support trace capture from a CoreSight PTM.

See also

- *CoreSight support in RealView Debugger v3.1* on page A-15

- *RealView Debugger Trace User Guide*.

A.4.9 Changes to the views in RealView Debugger v3.1

The following changes have been made to the RealView Debugger views:

- The display format for undocked views has changed. The related connection and associated color box are displayed at the bottom of each view.
- The view menu buttons have been removed.
- View operations are provided on the context menu.
- The Call Stack view contains only the **Call Stack** tab.
- The **Locals**, **Statics**, and **This** tabs are available in the new Locals view.
- The Memory view has been re-engineered:
  - A view toolbar has been added, which contains fields to specify the start address, the number of columns, the data size, and the format of memory cells.
  - A new cache coloring scheme has been added.
  - You can set values of memory locations by entering ASCII characters in the ASCII view.
  - When entering ASCII characters at a memory location, precede the characters by a single quote. You can enter as many characters as the currently selected data size implies.

See also

- the following in the *RealView Debugger User Guide*:
  - *Overview of RealView Debugger windows and views* on page 1-2.

A.4.10 Miscellaneous changes to the GUI in RealView Debugger v3.1

The following changes have been made to the RealView Debugger GUI:

- The **Src** tab has been removed from Code window.
- The **Dsm** tab has been renamed to **Disassembly**.
- The **Map** tab in the Process Control view has been renamed to **Memory Map**.

- A new **Show Next Statement** button has been added to the Debug toolbar. This shows the location of the PC in the **Disassembly** tab or a source file tab, and is identified by a yellow arrow in the margin with a red box around the instruction or line of source.
- A Scripts toolbar is provided, which enables you to quickly add, run and delete command scripts, shown in Figure A-2.

Figure A-2 Scripts toolbar

- A new button to reset the target processor has been added to the Connect toolbar.
- Some buttons have been removed from the Debug toolbar.
- Many changes have been made to the context menus, including:
  - renaming of the menu options for setting breakpoints and tracepoints

  - removal of some less useful menu options.
- The following **ARM on the Web** menu options are available on the **Help** menu:
  - **Goto ARM Technical Support**
  - **Goto ARM Development Tool FAQs**
  - **Goto ARM Technical Support Downloads**
  - **Goto ARM PDF Documentation**
  - **Goto ARM Self Help Forums**.

See also

- the following in the *RealView Debugger User Guide*:
  - Chapter 1 *RealView Debugger Features*.

A.4.11 Changes to CLI commands and macros in RealView Debugger v3.1

This section describes the changes that have been made to CLI commands and macros.

Changes to CLI commands

The following new CLI commands are provided:

- cache-related commands
- `COREINFO`, which enables you to display information about the current target
- `CORESTATE`, which enables you to display the execution state of the current target
- `REGINFO`, which enables you to display details of the registers available for the current target
- synchronization-related commands
- `VA2PA`, which enables you to convert a virtual address to a physical address.

See also

- *Multiprocessor debugging in RealView Debugger v3.1* on page A-16
- *Cache-related CLI commands and predefined macros* on page A-17
- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 2 *RealView Debugger Commands*.

Changes to macros

This sections describes the changes that have been made to macros.

New predefined macros

New cache-related predefined macros are provided.

Using integer variables

You can substitute the value of an integer variable in a CLI command before the command is executed. The value is converted to hexadecimal. To do this, you must enclose the variable name between the characters `${` and `}`, for example:

```
int num;
num = 1;
$FOPEN 150, "C:\\myfiles\\myfile${num}.txt"$; // substitution
```

The filename in this example is `myfile0x1.txt`.

See also

- *Cache-related CLI commands and predefined macros* on page A-17
- the following in the *RealView Debugger User Guide*:
  - *Using variable substitution in commands within a macro* on page 16-6
- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 3 *RealView Debugger Predefined Macros*.

A.4.12 Documentation changes in RealView Debugger v3.1

The following changes have been made to the RealView Debugger documentation:

- All documentation has been updated to reflect the changes to the RealView Debugger GUI and CLI commands.
- The *RealView Debugger Target Configuration Guide* has been restructured. A new chapter has been included, which provides a tutorial on memory mapping.
- The *RealView Debugger Trace User Guide* has been restructured. A tutorial has been included, which is to be used in conjunction with the `trace.c` program. The program is provided with *Application Note 168 Tracing with RVD*. To obtain the program, navigate to the `Application Notes` page under `Documentation` on the ARM website (`http://www.arm.com`).
- The chapter that describes target connection in the *RealView Debugger Target Configuration Guide* has been incorporated into the *RealView Debugger User Guide*.

A.4.13 Deprecated features in RealView Debugger v3.1

The following features are deprecated in RealView Debugger v3.1:

- The `/B`, `/H`, and `/W` qualifiers to the following commands are deprecated:
  - `DUMP`
  - `FILL`
  - `MEMWINDOW`
  - `SEARCH`
  - `SETMEM`
  - `TEST`.

  Use the `/8`, `/16`, and `/32` qualifiers as appropriate.
- The `rawb` and `rawh` qualifiers to the following commands are deprecated:
  - `READFILE`
  - `VERIFYFILE`
  - `WRITEFILE`.

  Use the `raw8` and `raw16` qualifiers as appropriate.
- The following command qualifiers are deprecated:
  - `ANALYZER,edit_properties`
  - `ETM_CONFIG,addronly`
  - `ETM_CONFIG,fulltrace`
  - `TRACEBUFFER,amount`.
- The following Workspace settings and groups are deprecated:
  - `_ctrl` settings group
  - `Vi` setting.
- The following top-level Debug Configuration settings in Connection Properties are deprecated:
  - the `Project` setting
  - the `Remote` settings group
  - the `Shared` setting (this is not the same as the `Shared` setting in the `Attributes` group of a `Memory_block` definition, which is still supported).

See also

- the following in the *RealView Debugger User Guide*:
  - Appendix A *Workspace Settings Reference*
- the following in the *RealView Debugger Target Configuration Guide*:
  - Appendix A *Connection Properties Reference*
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *Alphabetical command reference* on page 2-12.

A.4.14 Obsolete features in RealView Debugger v3.1

The following features are obsolete in RealView Debugger v3.1:

- Remote RVISS connection using RealView Simulator Broker.
- Connections to *Remote Debug Interface* (RDI) targets, which includes:
  - Multi-ICE®
  - Agilent Debug Interface
  - Remote_A.
- ETMv2 is no longer supported.
- The following CLI commands have been removed:
  - `NAMETRANSLATE`
  - `PATHTRANSLATE`.

A.5 Changes between RealView Debugger v3.0 and v1.8

This section describes the changes between RealView Debugger v3.0 and the previous release RealView Debugger v1.8. It includes:

- *TrustZone technology support in RealView Debugger v3.0*
- *Thumb-2EE Support in RealView Debugger v3.0*
- *OS support in RealView Debugger v3.0* on page A-24
- *Trace, Analysis, and Profiling in RealView Debugger v3.0* on page A-24
- *RealView Simulator Broker support in RealView Debugger v3.0* on page A-25
- *Multi-ICE direct connect in RealView Debugger v3.0* on page A-25
- *Simulator support in RealView Debugger v3.0* on page A-25
- *Changes to the GUI in RealView Debugger v3.0* on page A-25
- *Changes to the CLI commands and predefined macros in RealView Debugger v3.0* on page A-26
- *Updated documentation in RealView Debugger v3.0* on page A-27.

A.5.1 TrustZone technology support in RealView Debugger v3.0

RealView Debugger provides support for processors that support TrustZone® technology. The impact on RealView Debugger is as follows:

- The Analysis window identifies Secure World (`S:`) and Normal World (`N:`) addresses when trace information is collected from a processor that supports the TrustZone technology.
- When specifying addresses in dialog boxes or CLI commands you can include an address prefix to indicate either a Secure World (`S:`) or Normal World (`N:`) address, see:
  - *Trace CLI commands* on page A-27
  - *Other CLI commands* on page A-27.
- The RealView Debugger windows and panes that display addresses also show the Secure World (`S:`) or Normal World (`N:`) address prefix:
  - Analysis window
  - **Dsm** tab
  - Break/Tracepoints pane
  - Call Stack pane
  - Memory pane
  - Stack pane
  - Symbols pane.

A.5.2 Thumb-2EE Support in RealView Debugger v3.0

RealView Debugger provides support for *Thumb®-2 Execution Environment* (Thumb-2EE) enabled targets, such as Cortex-A8. For these targets:

- The following files are provided with RealView Debugger:
  - `thumb2ee.bcd`
  - `thumb2ee.inc`.

  See the *RealView Debugger Target Configuration Guide* for more details about these files.

- You can change the display of disassembly listings to Thumb-2EE format, using one of the following methods:
  - the **Set Disassembly Format...** option on the context menu in the disassembly view, that is, the **Dsm** tab (see the *RealView Debugger User Guide*)
  - the workspace (see the *RealView Debugger User Guide*)
  - the `SETTINGS` CLI command (see the *RealView Debugger Command Line Reference Guide*).

A.5.3 OS support in RealView Debugger v3.0

Operating system (OS) application debugging has been enhanced. For those OS plug-ins that support the enhancements, such as Linux HSD, you can perform the following debugging operations:

- capture events that are defined by your OS plug-in
- specify filters that enable you to selectively load debugging symbols.

To configure these operations, the following options are available on the connection context menu in the Resource Viewer pane:

- **Events Filter...**
- **Debug Symbols Filter...**.

Also, see *Changes to the CLI commands and predefined macros in RealView Debugger v3.0* on page A-26 for details of changes to the OS-related CLI commands.

See the *RealView Debugger RTOS Guide* for full details of debugging OS-aware targets in RealView Debugger.

A.5.4 Trace, Analysis, and Profiling in RealView Debugger v3.0

The following changes have been made to the trace, analysis, and profiling features:

- *Embedded Trace Macrocell* (ETM) configuration enables you to specify:
  - values for extended external inputs 1-4, for ETMv3.1 and later
  - a synchronization frequency, for ETMv2 and later.
- You can set tracepoints using extended external inputs 1 to 4, for ETMv3.1 and later.
- The details view has been removed from the Analysis window, together with the corresponding option on the **View** menu.
- The following changes have been made to the Analysis window **Filter** menu:
  - the **Invert Filtering (NOT)** option is included
  - the **AND Filters (versus OR)** option is two separate options, **OR All Filters** and **AND All Filters**
  - the **Filter on Raw Address Match...** option has been removed.
  - the text `Match` has been removed from all option names.
- The following changes have been made to the Analysis window **Find** menu:
  - the **Find Raw Address Match...** option has been removed
  - the text `Match` has been removed from all option names.
- The **Print Trace Lines...** option has been removed from the Analysis window **File** menu.

- The Set Address/Data Break/Tracepoint dialog box has been replaced with the Set/Edit Tracepoint dialog box.

Also, see *Changes to the CLI commands and predefined macros in RealView Debugger v3.0* on page A-26 for details of changes to the trace-related CLI commands.

See the *RealView Debugger Trace User Guide* for full details of tracing in RealView Debugger.

A.5.5 RealView Simulator Broker support in RealView Debugger v3.0

RealView Simulator Broker (RealView Broker) has been re-engineered. Although RealView Debugger still runs RealView Broker automatically for local host connections, starting RealView Broker for remote connections has changed. You must specify a username when starting RealView Broker manually. See the *RealView Debugger Target Configuration Guide* for more details.

A.5.6 Multi-ICE direct connect in RealView Debugger v3.0

Connections using Multi-ICE® direct connect are no longer supported. Therefore, you cannot use Multi-ICE to connect to DSP targets. To debug a DSP target use RealView-ICE, which you must purchase separately.

A.5.7 Simulator support in RealView Debugger v3.0

The following simulated targets are supported:

- An MPCore simulated target is supported in RVISS. However, this simulates only a single processor.
- New *Instruction Set System Model* (ISSM) simulator models are supported for the following processors:
  - Cortex-A8
  - Cortex-M3.

  A single ISSM Target Access is provided, which you can configure to either of these processors. If you want to simulate additional processors, you must create and configure a new Target Access for each processor.

See the *RealView Debugger Target Configuration Guide* for more details about configuring these simulated targets.

A.5.8 Changes to the GUI in RealView Debugger v3.0

The following changes have been made to the RealView Debugger GUI:

- The Connection Control window has been re-engineered to simplify the connection and configuration of debug targets.
- The **Synch** tab on the older Connection Control window is a separate Synchronization Control window. To open the Synchronization Control window, select **Target → Synchronization Control...** from the Code window main menu.
- All references to *software interrupt* (SWI) have been changed to *Supervisor Call* (SVC).
- The project management feature has been removed. Therefore:
  - the Code window **Project** menu has been removed
  - the source control toolbar button has been removed
  - you cannot load RealView Debugger project files.

However, the source code edit and search features are still available.

- There is a new Load Binary dialog box that simplifies the loading of binary files. To open the Load Binary dialog box, select **Target → Load Binary...** from the Code window main menu.

  Binary files you load in this way are added to the Recent Binaries list. To display the Recent Binaries list, select **Target → Recent Binaries** from the Code window main menu.

- The Set Address/Data Break/Tracepoint dialog box has been replaced with the following dialog boxes:
  - Create Breakpoint
  - Copy Breakpoint
  - Edit Breakpoint
  - Set/Edit Tracepoint.

- The Registers pane has been re-engineered:
  - The individual fields of the CPSR and SPSR registers are no longer shown as individual registers. To change these registers, a PSR Format dialog box is provided.
  - Extended formatting options are available for all registers.
  - Registers that have enumerated values appear as list selection boxes.
  - You can create your own user-specific register view by copying registers from the other tabs in the Registers pane.

- The Data Navigator pane is called the Symbols pane. To open the Symbols pane, you select **View → Symbols** from the Code window main menu.

- The Symbol Browser pane is called the Classes pane. To open the Classes pane, you select **View → Classes** from the Code window main menu.

- The Flash Memory Control dialog box includes a field that enables you to specify the clock frequency, if supported by your Flash device. See the *RealView Debugger Target Configuration Guide* and the *RealView Debugger User Guide* for more details.

- The Resource Viewer window is now a pane, which you can float and dock like any other pane.

See the *RealView Debugger User Guide* for more details about how to use these features.

A.5.9 Changes to the CLI commands and predefined macros in RealView Debugger v3.0

This section describes the changes that have been made to the CLI commands and predefined macros.

See the *RealView Debugger Command Line Reference Guide* for full details.

Connection CLI commands

The following change has been made to the connection CLI commands:

- The `DISCONNECT` command has the `debug` and `nodebug` qualifiers for specifying the disconnect mode.

OS-aware CLI commands

The following change has been made to the OS-aware CLI commands:

- If your OS-aware plug-in supports event debugging, then the `OSCTRL` command enables you to specify events and filters for the selective loading of debugging symbols.

See the *RealView Debugger RTOS Guide* for more details on using the events and filters.

Trace CLI commands

The following changes have been made to the trace CLI commands:

- The `TRACEDATAACCESS`, `TRACEDATAREAD`, `TRACEDATAWRITE`, `TRACEINSTREXEC`, and `TRACEINSTRFETCH` commands enable Secure World and Normal World data comparisons for processors that implement the TrustZone technology.
- The `TRACEEXTCOND` command supports extended external inputs 1 to 4, for ETMv3.1 and later.
- The `TRACEBUFFER` command includes an option to invert the sense of the specified filter conditions.
- The `ETM_CONFIG` command enables you to specify:
  - values for extended external inputs 1 to 4, for ETMv3.1 and later
  - a synchronization frequency, for ETMv2 and later.

Other CLI commands

The changes made to other CLI commands are as follows:

- the `BREAKACCESS`, `BREAKEXECUTION`, `BREAKINSTRUCTION`, `BREAKREAD`, and `BREAKWRITE` commands enable you to specify an address prefix to identify Secure World and Normal World addresses on a target that supports TrustZone technology
- the `DISASSEMBLE` command enables you to show Thumb-2EE disassembly
- the `LOAD` command enables you to load images to either the Secure World or the Normal World on a target that supports TrustZone technology
- the access size options of the `READFILE`, `VERIFYFILE`, and `WRITEFILE` commands have changed
- the `SETTINGS` command no longer supports the project-related settings
- a new `STATS` command enables you to display bus and processor cycles for RVISS targets.

Predefined macros

The following change has been made to the predefined macros:

- There is a new `fclose` macro to complement the `fopen` macro.

A.5.10 Updated documentation in RealView Debugger v3.0

The following changes have been made to the RealView Debugger documentation:

- The documentation suite comprises the documents listed in *RealView Development Suite Documentation*.
- The *RealView Debugger User Guide* is task-based.

- The chapter that describes target connection in the *RealView Debugger Target Configuration Guide* has been incorporated into the *RealView Debugger User Guide*.
- The *RealView Debugger Extensions User Guide* has been removed. The contents of the *RealView Debugger Extensions User Guide* are in the following documents:
  - the chapter and appendixes that describe tracing are in the new document *RealView Debugger User Guide*
  - the chapter that describes OS support is in the new document *RealView Debugger RTOS Guide*
  - the chapter that describes multiprocessor debugging has been incorporated into the *RealView Debugger User Guide*
  - the chapter that describes DSP support has been incorporated into the *RealView Debugger User Guide*.
- The *RealView Debugger Project Management User Guide* is no longer provided.

A.6 Changes between RealView Debugger v1.8 and v1.7

This section describes the changes between RealView Debugger v1.8 and the previous release RealView Debugger v1.7. It includes:

- *Updated documentation in RealView Debugger v1.8*
- *Trace, Analysis, and Profiling in RealView Debugger v1.8*
- *Support for gcc built images in RealView Debugger v1.8* on page A-30
- *Changes to the GUI in RealView Debugger v1.8* on page A-30
- *Changes to the CLI in RealView Debugger v1.8* on page A-33
- *RealView ARMulator ISS support in RealView Debugger v1.8* on page A-33
- *RealMonitor support in RealView Debugger v1.8* on page A-33.

A.6.1 Updated documentation in RealView Debugger v1.8

The changes to the documentation includes the following:

- There is a new chapter about programming Flash in the *RealView Debugger Target Configuration Guide*. This chapter describes in detail how to create the files required to program Flash with RealView Debugger, whether you are using the Flash devices and boards currently supported by RealView Debugger, or your own custom Flash devices and boards.
- The *RealView Debugger Essentials Guide* includes:
  - all information that relates to moving from *ARM® eXtended Debugger* (AXD) or *ARM Symbolic Debugger* (armsd) to RealView Debugger
  - background information about RealView Debugger projects, and how to set up the basic compilation tasks for a project
  - details on how to get started using the RealView Debugger CLI
  - an appendix showing the mapping of the main menu options to the toolbar buttons.
- The task-related information from the *RealView Debugger Command Line Reference Guide* is in the chapter that describes getting started using the RealView Debugger CLI in the *RealView Debugger Essentials Guide*.
- Enhancements to tracing with RealView Debugger have been documented (see *Trace, Analysis, and Profiling in RealView Debugger v1.8* for a summary).
- All documents reflect the enhancements to the RealView Debugger GUI (see *Changes to the GUI in RealView Debugger v1.8* on page A-30 for a summary).
- The RealView Debugger online help is in HTML format, and is opened in your default web browser.

A.6.2 Trace, Analysis, and Profiling in RealView Debugger v1.8

RealView Debugger v1.8 includes enhancements to the Trace and profiling features:

- tracepoints are categorized as unconditional or conditional
- the Analysis window has changed (see *Changes to the Analysis window* on page A-30 for a summary of the changes)
- the Configure ETM dialog has changed to support the new *Embedded Trace Macrocell v3* (ETMv3), ETB11™ port widths (24-bit and 32-bit)
- there is a new trace command, `TRACEEXTCOND`.

Changes to the Analysis window

The changes to the Analysis window include:

- the following new features:
  - display of interleaved inferred register values
  - functionality to sum profiling data over a number of runs
  - rationalization of the window tabs
- the following improvements:
  - block fetching of trace data, to enhance performance
  - changes to column layouts and the addition of new columns
  - new options in the profiling window.

These changes have the following implications:

- sorting of trace data is supported only in the **Profile** tab
- appending trace data is no longer supported.

For full details on tracing with RealView Debugger, see the *RealView Debugger Extensions User Guide*.

A.6.3 Support for gcc built images in RealView Debugger v1.8

RealView Debugger supports images built with gcc as follows:

- Images built with gcc v3.2 and v3.4 are fully supported.
- Images built with gcc v2.95.3 are supported, but with no stack backtrace. In addition, these images require converting to ELF format using the `coff2elf` utility. You can obtain this utility from the ARM Technical Support web page.

A.6.4 Changes to the GUI in RealView Debugger v1.8

This section describes the major changes to the RealView Debugger GUI.

Changes to the Code window main menu structure

The Code window main menus have been restructured, with new menus added, and menu options moved to reflect their functionality:

File The following changes have been made to this menu:

- the image load options are available on the new **Target** menu
- connections are available on the new **Target** menu
- logs and journals are available on the **Tools** menu
- all workspace-related options are available from the **Workspace** submenu.

Note

The **Threads** menu option has been removed, because this feature is available using the **Cycle Threads** toolbar button.

Edit All editing-related functionality is available from this menu, including copying and pasting, and searching. The changes include the following:

- the new **Advanced** submenu contains the options from the old **Format** and **Editing Controls** submenus

- the new **Go To** submenu contains the original **Jump** options.
- removal of some little-used options, such as **VI mode**.

View All pane views are accessible from the main **View** menu, instead of the **New Pane Views** submenu. A new Data Navigator option is available to display the Data Navigator pane. The Browse Symbols dialog box is replaced by a Symbol Browser pane, which has enhanced functionality.

Target This is a new menu that includes all options relating to connections and image loading.

Project This menu contains all the project-related options available in previous versions of RealView Debugger, but the grouping has changed.

Build This is a new menu that includes all the build-related options from the **Tools** menu.

Debug The following changes have been made to this menu:

- the options on the **Execution Control** menu are available on the main **Debug** menu.
- the breakpoint options are combined into a single **Breakpoints...** submenu
- the **Debug/Simple Breakpoints** submenu options are redistributed between the new **Breakpoints** submenu and the new **Breakpoints** → **Conditional** submenu
- the options on the **Complex Breakpoints** submenus are available from the **Breakpoints** → **Hardware...** submenu
- the **Processor Events** option is available from the option **Processor Exceptions...**
- the **Include Commands from File...** option is available on the **Tools** menu.

Tools The following changes have been made to this menu:

- the build-related options are available on the new **Build** menu
- the workspace options are available from the **File** → **Workspace** menu
- a new **Logs and Journal** submenu is available for opening and closing log and journal files
- the **Include Commands from File...** option has been moved to this menu from the **Debug** menu
- the **New Editor** submenu has been removed.

Help The following changes have been made to this menu:

- all web-related options are on the **ARM on the Web** submenu
- the **Full Online Documentation** option has been removed
- access to the RealView Debugger online help is simplified, and is available through the **RealView Debugger Help** option.

Toolbar changes

The Code window toolbar is split into separate function-specific toolbars. Each toolbar can be hidden, moved, or floated independently. The toolbars are:

- File
- Edit
- Debug

- Image
- Connect
- Build
- Find.

Internationalization is supported

Internationalization is supported, so that you can localize the RealView Debugger interface. You can:

- set the language to English (the default), or Japanese
- set the default encoding to ASCII (the default), UTF-8, or Locale.

This displays the main menu and various context menu options in the chosen language. For instructions on how to change the these settings, see the *RealView Debugger User Guide*.

Changes to pane views

The following changes have been made to panes in RealView Debugger:

- You have greater flexibility over the docking of panes.
- All panes, except for the File Editor pane, can be floated, hidden, and repositioned on the RealView Debugger desktop.
- You can set the font used in the pane views. You can set the font name, style, size, and script.
- There is a new Data Navigator pane that enables you to quickly locate a module, function, or variable in an image.
- The Browse Symbols dialog box is replaced by a Symbol Browser pane, which has enhanced functionality.
- The **Expand/Collapse Pane** button has been removed from the pane toolbar.

Changes to breakpoints

Breakpoints are no longer categorized as Simple and Complex. They are categorized as Software and Hardware, and as unconditional or conditional. As a consequence, the names of the various dialog boxes used to set breakpoints have changed. See the *RealView Debugger User Guide* for more details.

The Set Address/Data Break/Tracepoint dialog box can only be used to set breakpoints. Therefore, this dialog box is called Set Address/Data Breakpoint.

Named breakpoints are no longer supported. Although the `Named_breaks` group is still present in the Project Properties, the **Named...** menu option has been removed. The `Named_breaks` group is to be removed in a future release.

Changes to tracepoints

The following changes have been made to tracepoints:

- Tracepoints are no longer categorized as Simple and Complex. They are categorized as unconditional or conditional.
- You can no longer use the Set Address/Data Break/Tracepoint dialog box to set tracepoints. Therefore, this dialog box is called Set Address/Data Breakpoint.

See the chapter that describes tracing in RealView Debugger in the *RealView Debugger Extensions User Guide* for more details.

Changes to editing facilities

The following changes have been made to the editing facilities in RealView Debugger:

- You have more control over source code coloring. You can:
  - set both the foreground text color and the background color for various code elements
  - choose the default colors from a list of color schemes, such as Visual Studio
  - set colors for the additional code element identifiers, preprocessor keywords, and operators.
- You can set the font used in the pane views. You can set the font name, style, size, and script.
- The standalone Editor window has been removed. Also, there is no longer support for using personal standalone editors with RealView Debugger.
- You can no longer use Vi mode when editing.

A.6.5 Changes to the CLI in RealView Debugger v1.8

The following changes have been made to the CLI:

- Spaces are no longer allowed in connection names. Therefore, the RealView ICE connection is called RealView-ICE.
- A new CLI command `PRINTDSM` is available to disassemble the contents of target memory to the **Cmd** tab of the Output pane. The disassembly is in the same format as that shown in the **Dsm** tab of the File Editor pane. Therefore, if you have a journal file open, you can output the disassembly to a file.
- A new trace command, `TRACEEXTCOND`, is available to set a tracepoint that triggers when an instruction in the specified address range is executed.
- The switches `/MB` and `/R` have been added to the `PRINTVALUE` command:
  - `/MB` enables multibyte character values to be displayed correctly
  - `/R` prevents the address being displayed when you specify a variable in a loaded image.
- A `/S` switch has been added to the `INCLUDE` command to stop the CLI commands being echoed to the display.
- The `ETM_CONFIG` command supports the ETB11 port widths.

A.6.6 RealView ARMulator ISS support in RealView Debugger v1.8

Map files are supported for RVISS connections through the RealView Broker interface.

A.6.7 RealMonitor support in RealView Debugger v1.8

You can use RealMonitor with RealView ICE connections. Also, how you configure a Multi-ICE connection to use RealMonitor has changed.

For details on how to configure and use RealMonitor with RealView ICE and Multi-ICE, see the *RealView Debugger Target Configuration Guide*.

A.7 Changes between RealView Debugger v1.7 and v1.6.1

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

- *Updated documentation in RealView Debugger v1.7*
- *Advanced debugging facilities in RealView Debugger v1.7*
- *RealView ARMulator ISS support in RealView Debugger v1.7*
- *Trace, Analysis, and Profiling in RealView Debugger v1.7*
- *Enhanced RTOS support in RealView Debugger v1.7* on page A-36
- *New GUI elements in RealView Debugger v1.7* on page A-36.

A.7.1 Updated documentation in RealView Debugger v1.7

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 AXD.
- 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*.
- *RealView Debugger User Guide* for 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.7.2 Advanced debugging facilities in RealView Debugger v1.7

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

A.7.3 RealView ARMulator ISS support in RealView Debugger v1.7

In RealView Developer Suite, 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*.

Note

The RDI connection to RVISS in RealView Debugger is deprecated in this release.

A.7.4 Trace, Analysis, and Profiling in RealView Debugger v1.7

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.

A.7.5 Enhanced RTOS support in RealView Debugger v1.7

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 pane.

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 that describes RTOS support in the *RealView Debugger Extensions User Guide* for full details of all the RTOS support provided by RealView Debugger v1.7.

A.7.6 New GUI elements in RealView Debugger v1.7

New toolbar buttons and menu changes mean that RealView Debugger v1.7 users 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 **Cycle Threads** button
- a new Actions toolbar button to hide or open 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.