RealView® Debugger

Version 3.0

Essentials Guide

ARM®

ARM DUI 0181H

RealView Debugger
Essentials Guide

Release Information

The following changes have been made to this document.

Change History

Date	Issue	Change
April 2002	A	RealView Debugger v1.5 release
September 2002	B	RealView Debugger v1.6 release
February 2003	C	RealView Debugger v1.6.1 release
September 2003	D	RealView Debugger v1.6.1 release for RVDS v2.0
January 2004	E	RealView Debugger v1.7 release for RVDS v2.1
December 2004	F	RealView Debugger v1.8 release for RVDS v2.2
May 2005	G	RealView Debugger v1.8 SP1 release for RVDS v2.2 SP1
March 2006	H	RealView Debugger v3.0 release for RVDS v3.0

Proprietary Notice

Confidentiality Status

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 v3.0 Essentials Guide*, that shows you how to start using RealView Debugger to debug your application programs. It contains the following sections:

About this book

RealView Debugger provides a powerful tool for debugging applications targeted at ARM® architecture-based processors and supported *Digital Signal Processors* (DSPs). 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 has been written for developers who are using RealView Debugger to debug applications targeted at ARM architecture-based processors and supported DSPs. 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® ISS* (RVISS) to simulate an ARM core-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 using this book. It describes how this book is organized, where to find specific features, and how to use the tutorial and the rest of the documentation suite.

Chapter 2 ***Changes to RealView Debugger***

Read this chapter for a summary of the changes to RealView Debugger v3.0.

Chapter 3 ***Getting Started with RealView Debugger***

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

Appendixes

Appendix A ***About Previous Releases***

Read this chapter for a history of RealView Debugger releases v1.8, v1.7, and v1.6.1.

Typographical conventions

The following typographical conventions are used in this book:

italic	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes ARM processor signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.
`mono`space	Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.
`monospace italic`	Denotes arguments to commands and functions where the argument is to be replaced by a specific value.

monospace bold Denotes language keywords when used outside example code.

Further reading

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

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

ARM publications

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

- *RealView Debugger v3.0 User Guide* (ARM DUI 0153)
- *RealView Debugger v3.0 Target Configuration Guide* (ARM DUI 0182)
- *RealView Debugger v3.0 Trace User Guide* (ARM DUI 0322)
- *RealView Debugger v3.0 RTOS Guide* (ARM DUI 0323)
- *RealView Debugger v3.0 Command Line Reference Guide* (ARM DUI 0175).

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

For an introduction to all the components of RVDS, see the following books:

- *RealView Development Suite Getting Started Guide* (ARM DUI 0255)
- *RealView Development Suite Glossary* (ARM DUI 0324).

For details on using RVISS, see the following book:

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

For general information on software interfaces and standards supported by ARM, 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:

- *RealView ICE and RealView Trace User Guide* (ARM DUI 0155)
- *Multi-ICE® User Guide* (ARM DUI 0048)
- *ARM Agilent Debug Interface User Guide* (ARM DUI 0158).

Other publications

For a comprehensive introduction to ARM architecture see:

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

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.

For more information about CEVA-Oak, CEVA-TeakLite, and CEVA-Teak processors from CEVA, Inc. see `http://www.ceva-dsp.com`.

For more information about the ZSP400 and ZSP500 processors from the ZSP division of LSI Logic see `http://www.zsp.com`.

Feedback

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

Feedback on RealView Debugger

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

1. Select **Help → Send a Problem Report...** from the RealView Debugger main menu.
2. Complete all sections of the Software Problem Report.
3. To get a rapid and useful response, give:
   - a small standalone sample of code that reproduces the problem, if applicable
   - a clear explanation of what you expected to happen, and what actually happened
   - the commands you used, including any command-line options
   - sample output illustrating the problem.
4. Email the report to your supplier.

Feedback on this book

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

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

General suggestions for additions and improvements are also welcome.

Chapter 1
About RealView Debugger

This chapter introduces RealView® Debugger. It explains how the debugger provides a development environment for embedded systems applications using ARM® architecture-based processors and supported *Digital Signal Processors* (DSPs).

This chapter contains the following sections:

1.1 RealView Debugger

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.

1.1.1 RealView Debugger concepts and terminology

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

Debug target

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

Target Access

A piece of hardware or software simulator that provides the interface between the debugger and the debug target.

Connection The link between the debugger and the debug target.

Single connection access

The RealView Debugger base product enables you to carry out debugging tasks in single-processor debugging mode, that is you can connect only to a single processor. For a multiprocessor core, such as MPCore™, you can connect to only one processor on that core.

Multiprocessor access

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

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

OS Operating systems (OSs) provide software support for applications running on a target. For example, *Real Time Operating Systems* (RTOSs) are operating systems that are designed for systems that interact with real-world activities where time is critical.

Multithreaded operation

When RealView Debugger is used with a suitable OS-aware plugin it can present thread information and scope some debugging operations to specific threads.

1.2 About the debugging environment

RealView Debugger uses a three-tier environment to debug applications:

- debugger software
- a debug interface layer, incorporating the Target Access interface
- the debug target.

RealView Debugger uses connection information to describe:

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

Connection information might also include cached copies of processor registers or memory. This approach means that you can switch between debug targets without having to start a second or third instance of the debugger program.

This section describes the RealView Debugger debugging environment:

- *Components of RealView Debugger*
- *Target Access interface* on page 1-5
- *Persistence information* on page 1-6.

1.2.1 Components of RealView Debugger

RealView Debugger comprises:

Graphical User Interface (GUI)

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

Command Line Interface (CLI)

This gives access to many of the features of the debugger through RealView Debugger commands. This enables you to create command scripts to automate debugging.

RealView Simulator Broker

RealView Simulator Broker, `rvbroker2.exe`, handles connections to a simulated Target Access that resides on the same workstation as RealView Debugger or to a simulated Target Access on any other workstation on your network.

1.2.2 Target Access interface

RealView Debugger works with either a hardware or a software debug target. An ARM development board communicating through RealView ICE or Multi-ICE® is an example of a hardware Target Access. RealView ARMulator® ISS and Instruction Set System Model are examples of a software Target Access.

Each Target Access processes requests from the client tools to the target. A Target Access might be a JTAG interface, a simulator, or a ROM monitor.

For details on how to connect to targets, see the *RealView Debugger v3.0 User Guide*.

For details on how to configure targets and connections, see the *RealView Debugger v3.0 Target Configuration Guide*.

Instruction Set System Model

Instruction Set System Model (ISSM) simulates Cortex™-A8 and Cortex-M3 models. ISSM runs on the same host computer as the debugger, and includes facilities for communicating with the debugger.

RealView ARMulator ISS

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

RVISS runs on the same host computer as the debugger, and includes facilities for communicating with the debugger. Alternatively, using RealView Simulator Broker, you can connect to RVISS on a remote host.

Note

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

1.2.3 Persistence information

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

- current target connections
- loaded images
- desktop settings, for example pane selections and window positions.

1.3 Debugging mode

The RealView Debugger base product enables you to debug your images in single-processor debugging mode, that is, you can connect to a single processor only.

If you have the appropriate licenses, you can also debug multiprocessor applications, which can be:

- all ARM architecture-based cores
- a mixture of ARM architecture-based cores and DSPs.

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

Multiprocessor debugging mode is a separately licensed feature of RealView Debugger and is described in detail in the *RealView Debugger v3.0 User Guide*.

1.4 The RealView Debugger documentation suite

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

- *RealView Debugger v3.0 User Guide*
- *RealView Debugger v3.0 Target Configuration Guide*
- *RealView Debugger v3.0 Trace User Guide*
- *RealView Debugger v3.0 RTOS Guide*
- *RealView Debugger v3.0 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 v3.0 User Guide*. This describes, in detail, how to debug your images and how to configure RealView Debugger to customize your working environment. It also includes debugging multiprocessor targets. This book also contains examples of debugging software and details shortcuts and tips for the developer.

   Note

   Appendix E *RealView Debugger for Sun Solaris and Red Hat Linux* of the *RealView Debugger v3.0 User Guide* contains information for developers using RealView Debugger on non-Windows platforms.

2. *RealView Debugger v3.0 Target Configuration Guide* describes how to amend existing targets that are set up in the base product, and how to customize your own targets. 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 v3.0 Trace User Guide*.

4. If you have the appropriate plugins, you can debug OS applications using the OS-awareness features in RealView Debugger. These features are described in the *RealView Debugger v3.0 RTOS Guide*.

5. If you want to use the RealView Debugger *Command Line Interface* (CLI) to control your debugging tasks, the *RealView Debugger v3.0 Command Line Reference Guide* provides a detailed description of every CLI command and includes examples of their use.

See the installation notes delivered with your product for details on installing RealView Debugger.

Chapter 2
Changes to RealView Debugger

This chapter describes the changes between RealView® Debugger v3.0 and the previous release, RealView Debugger v1.8. It contains the following sections:

Note

For a description of the differences between previous versions of RealView Debugger, see Appendix A *About Previous Releases*.

2.1 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 contains the following sections:

2.1.1 TrustZone support

RealView Debugger provides support for processors that implement the TrustZone® architecture. 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 architecture.
- When specifying addresses in dialog boxes or CLI commands you can include an address prefix to indicate either a Secure (`S:`) or Normal (`N:`) world address, see:
  - *Trace CLI commands* on page 2-7
  - *Other CLI commands* on page 2-8.
- The RealView Debugger windows and panes that display addresses also show the Secure (`S:`) or Normal (`N:`) world address prefix:
  - Analysis window
  - **Dsm** tab
  - Break/Tracepoints pane
  - Call Stack pane
  - Memory pane
  - Stack pane
  - Symbols pane.

2.1.2 Thumb-2EE Support

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 v3.0 Target Configuration Guide* for more details about these files.
- You can now 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 v3.0 User Guide*)
  - the workspace (see the *RealView Debugger v3.0 User Guide*)
  - the `SETTINGS` CLI command (see the *RealView Debugger v3.0 Command Line Reference Guide*).

2.1.3 OS support

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

- capture events that are defined by your OS plugin
- 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* on page 2-7 for details of changes to the OS-related CLI commands.

See the *RealView Debugger v3.0 RTOS Guide* for full details of debugging OS-aware targets in RealView Debugger.

2.1.4 Trace, Analysis, and Profiling

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 now 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 now included
  - the **AND Filters (versus OR)** option is now 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* on page 2-7 for details of changes to the trace-related CLI commands.

See the *RealView Debugger v3.0 Trace User Guide* for full details of tracing in RealView Debugger.

2.1.5 RealView Simulator Broker support

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

2.1.6 Multi-ICE direct connect

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.

2.1.7 Simulator support

The following simulated targets are now supported:

- An MPCore™ simulated target is now supported in *RealView ARMulator® ISS* (RVISS). However, this simulates only a single processor.
- New *Instruction Set System Model* (ISSM) simulator models are now 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 v3.0 Target Configuration Guide* for more details about configuring these simulated targets.

2.1.8 Changes to the GUI

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 (see *Connecting to a debug target* on page 3-6).
- The **Synch** tab on the older Connection Control window is now a separate Synchronization Control window. To display 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 display 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 now available for all registers.
  - — Registers that have enumerated values appear as list selection boxes.
  - — You can now create your own user-specific register view by copying registers from the other tabs in the Registers pane.

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

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

- The Flash Memory Control dialog box now includes a field that enables you to specify the clock frequency, if supported by your Flash device. See the *RealView Debugger v3.0 Target Configuration Guide* and the *RealView Debugger v3.0 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 v3.0 User Guide* for more details about how to use these features.

2.1.9 Changes to the CLI commands and predefined macros

The changes that have been made to the CLI commands and predefined macros are described in the following sections:

- *Connection CLI commands*
- *OS-aware CLI commands*
- *Trace CLI commands*
- *Other CLI commands* on page 2-8
- *Predefined macros* on page 2-8.

See the *RealView Debugger v3.0 Command Line Reference Guide* for full details.

Connection CLI commands

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

- The `DISCONNECT` command now 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 plugin 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 v3.0 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 architecture.
- 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 TrustZone-enabled target
- the DISASSEMBLE command enables you to show Thumb-2EE disassembly
- the LOAD command enables you to load images to either the Secure or Normal world on a TrustZone-enabled target
- 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.

2.1.10 Updated documentation

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

- The documentation suite now comprises the documents listed in *The RealView Debugger documentation suite* on page 1-8.
- The *RealView Debugger v3.0 User Guide* is now task-based.
- The chapter that describes target connection in the *RealView Debugger v3.0 Target Configuration Guide* has been incorporated into the *RealView Debugger v3.0 User Guide*.
- The *RealView Debugger Extensions User Guide* has been removed. The contents of the *RealView Debugger Extensions User Guide* are now in the following documents:
  - the chapter and appendixes that describe tracing are now in the new document *RealView Debugger v3.0 Trace User Guide*

  - — the chapter that describes OS support is now in the new document *RealView Debugger v3.0 RTOS Guide*
  - — the chapter that describes multiprocessor debugging has been incorporated into the *RealView Debugger v3.0 User Guide*
  - — the chapter that describes DSP support has been incorporated into the *RealView Debugger v3.0 User Guide*.
- The *RealView Debugger Project Management User Guide* is no longer provided.

2.2 Getting more information online

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

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

- **RealView Development Suite v3.0 → RVDS v3.0 Documentation Suite** to view the PDF files.
- **DynaText Documentation** to view the DynaText files.

Note

The DynaText and PDF files contain the same information.

The *RealView Debugger v3.0 User Guide* contains information for developers using RealView Debugger on non-Windows platforms. See the appendix that describes RealView Debugger for Sun Solaris and Red Hat Linux in the *RealView Debugger v3.0 User Guide* for more details.

Chapter 3
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:

3.1 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 *RealView Compilation Tools* (RVCT) supplied with RVDS.

Before you start the tutorial, make a copy of the Dhrystone example, and use your copy during the tutorial. See *Starting the tutorial* on page 3-3.

3.2 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 RVDS examples directory:

   `install_directory``\RVDS\Examples\...\windows\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`
     - `dhrystone.mcp` (the CodeWarrior project file).
   - 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.

3.3 Starting RealView Debugger

To start RealView Debugger, select the following from the Windows **Start** menu:

Programs → ARM → RealView Development Suite v3.0 → RealView Debugger v3.0

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

RealView Debugger uses your login ID as the name of your home directory, which is in the default directory:

install_directory\RVD\Core\...\home

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

The main RealView Debugger window is known as the Code window. The default layout of the Code window is shown in Figure 3-1 on page 3-5.

Note

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

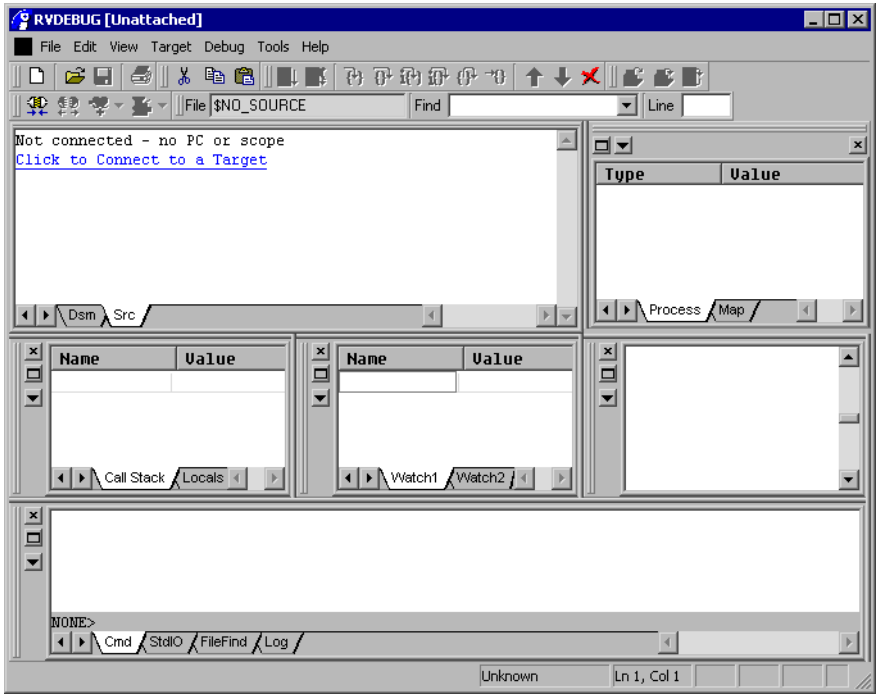

Figure 3-1 Default Code window

3.4 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 Connection Control window. This section contains:

- *How to display the Connection Control window*
- *Elements of the Connection Control window* on page 3-7
- *Making a connection* on page 3-7.

For more details on connecting to a debug target, see the chapter that describes connecting to targets in the *RealView Debugger v3.0 Target Configuration Guide*.

For details on working with multiple target connections, see the *RealView Debugger v3.0 User Guide*.

Note

Be aware that the default connection mode stops the target.

3.4.1 How to display the Connection Control window

To display the Connection Control window, do one of the following:

- select **Target → Connect to Target...** from the Code window main menu
- click the blue hyperlink **Click to Connect to a Target** in the File Editor pane
-  click the **Connection Control** button on the Connect toolbar to display the Connection Control window.

The Connection Control window is displayed, shown in Figure 3-2 (see *Elements of the Connection Control window* on page 3-7 for details).

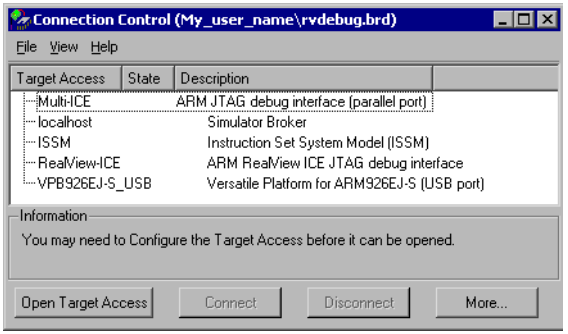

Figure 3-2 Connection Control window

3.4.2 Elements of the Connection Control window

The Connection Control window includes a tree control, which comprises:

Target Access

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

RealView Simulator Broker connections are made through the `localhost` Target Access.

Target processor

When you open a Target Access, the second-level entry shows the target processors that are available, for example, `ARM7TDMI_0`.

The target processor shows the current state of the connection.

Target processor entries usually have a numerical suffix (for example, `_0`). If you are licensed to use multiprocessor debugging, each connection is numbered consecutively, starting at zero. This suffix ensures a unique identification for your connections in case you have two processors the same. For example, if your board contains two ARM7TDMI® processors and an ARM940T™ processor, then the connections might be named `ARM7TDMI_0`, `ARM7TDMI_1`, and `ARM940T_2`.

Note

In multiprocessor debugging mode, a Synchronization Control window is also available. See the chapter that describes debugging multiple targets in the *RealView Debugger v3.0 User Guide*.

3.4.3 Making a connection

This tutorial uses the `localhost` (RVISS) Target Access. 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. In the Connection Control window (see Figure 3-2 on page 3-6), select the Target Access for the type of target to which you want to connect.

   For example, select the `localhost` Target Access to connect to a *RealView ARMulator® ISS* (RVISS) target.

2. Click **Open Target Access** to show the target processors available for the connection.

   For RVISS, the target is called `new_arm`.

3. Select the target processor to which you want to connect.

   For RVISS, select `new_arm`.

4. Click **Connect** to connect to that processor.

With the connection established, your Code window is updated:

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

For RVISS, a new connection is created, called `Simarm_0`, and the connection information in the Code window title bar changes to:

```
@Simarm_0:Sim
```

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

Note

If you are licensed to work in multiprocessor debugging mode, another `new_arm` entry appears for you to make more connections using RVISS.

If you are always debugging code on the same target you can configure RealView Debugger to make the same connection automatically each time it starts. See the chapter that describes connecting to targets in the *RealView Debugger v3.0 User Guide* for details.

3.5 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. This section describes:

- *Loading an image directly*
- *Loading multi-image applications to a single debug target* on page 3-10.

3.5.1 Loading an image directly

To load an image directly to your debug target:

1. Select **Target → Load Image...** from the Code window main menu.

   Alternatively, click the blue hyperlink **Click to Load Image to Target** in the File Editor pane.

   The Select Local File to Load dialog box is displayed.

   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 3-3):

   `install_directory``\RVDS\Examples\...\Tutorial\Debug`

3. Click **Open**.

   The image is loaded to the debug target.

Note

In the Code window **Text Coloring** is enabled by default. If you want to turn on source code line numbering, then select **Edit → Advanced → Show Line Numbers** from the Code window main menu.

When you load an image, the debugger:

- inserts the source filename, for the current context, in the File field of the Find toolbar
- highlights the location of the *Program Counter* (PC) at the entry point with a red box
- moves the text insertion point to the current location of the PC
- updates the Code window panes as appropriate, for example, it updates the process information in the Process Control pane

- displays the load command, used by RealView Debugger to load the image, in the **Cmd** tab in the Output pane.

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, this opens the source file `dhry_1.c` into the **Src** tab when you load the image because it has the context.

3.5.2 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 (see *Loading an image directly* on page 3-9).
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.

For more details about working with multiple images, see the chapter that describes loading images and binaries in the *RealView Debugger v3.0 User Guide*.

3.6 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 **Edit → Advanced → Show Line Numbers** from the Code window main menu to display line numbers in the source tab.

   This is not required but might help you to follow the examples in this tutorial.

2. Click on the **Src** tab in the File Editor pane.

3. Set a default breakpoint at line 149 in `dhry_1.c`, `Proc_5();`. To do this double-click on the line number.

   If the line number is not visible, then double-click inside the gray area at the left of the required statement in the File Editor pane to set the breakpoint.

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

4. Run the image as described in *Running the image* on page 3-12.

See the chapter that describes setting breakpoints in the *RealView Debugger v3.0 User Guide* for more details.

3.7 Running the image

To run an image, either:

- Select **Debug** → **Run** from the 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 pane is updated to show that the image is running
- the **StdIO** tab of the Output pane is selected, and application messages and prompts are displayed
- the CLI prompt in the **Cmd** tab of the Output pane changes to `Run>`.

See the chapter that describes executing images in the *RealView Debugger v3.0 User Guide* for more details.

3.7.1 Continuing the tutorial

If you are running the Dhrystone example described in *Setting a simple breakpoint* on page 3-11, 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 3-11 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 pane, 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 pane.

- Step through the image. For example, select **Debug → Step Into** from the Code window main 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.

For full details on how to use these features, see the *RealView Debugger v3.0 User Guide*.

3.8 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. See *How to explicitly unload an image* for instructions.

You do not have to unload an image from a debug target before loading a new image for execution. Display the Select Local File to Load dialog box and make sure that the **Replace Existing File(s)** check box is selected (see *Loading an image directly* on page 3-9).

3.8.1 How to explicitly unload an image

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

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

   Note

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

2. Right-click on either the `Image` or `Load` entry to display the **Image** context menu.
3. Select **Unload**.

Alternatively, in the Process Control pane unselect 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 pane and select **Delete Entry** from the context menu.

3.9 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.

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

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

- Select **Target → Connect to Target...** from the Code window main menu to display the Connection Control window, and select the required connection, and click **Disconnect**.

  You can also click the **Connection Control** button on the Connect toolbar to display the Connection Control 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 pane contents.

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

Note

It is not necessary to disconnect from a target before you close down RealView Debugger. See *Exiting RealView Debugger* on page 3-16 for more details.

3.10 Exiting RealView Debugger

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

- *Closing down RealView Debugger*
- *Reconnecting to a target*.

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.

3.10.1 Closing down RealView Debugger

To exit RealView Debugger:

1. Select **File** → **Exit** to display the Exit dialog box.
2. 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.

3.10.2 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 Target Access status has changed, for example if your RealView ICE interface has been disconnected or if your Multi-ICE® server has stopped.

If RealView Debugger reconnects successfully, the connection mode specified in the target configuration file is used by default. See the chapter that describes target connection in the *RealView Debugger v3.0 User Guide* for full details on defining the connect mode for your debug target.

3.11 Cleaning up after the tutorial

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

3.12 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 one of the following:

- Chinese
- Japanese.

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

This section includes:

- *Font recommendations*
- *Procedure summary*
- *Configuring the internationalization settings* on page 3-19
- *Configuring the pane views* on page 3-20.

3.12.1 Font recommendations

If you are changing the language to Chinese or Japanese then it is recommended that you also change the font as shown in Table 3-1. For instructions on how to change the font, see *Configuring the internationalization settings* on page 3-19.

Table 3-1 Font recommendations

Language	Font to use
Chinese	`MingLiU`
Japanese	`MSGothic` or `MSMincho`

3.12.2 Procedure summary

To localize the RealView Debugger interface, you must:

1. Configure the internationalization settings.

   See *Configuring the internationalization settings* on page 3-19 for instructions on how to do this.

2. Configure the panes that display normal text to show the text in the correct text encoding.

   See *Configuring the pane views* on page 3-20 for instructions on how to do this.

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 a CodeWarrior project (see the *CodeWarrior IDE Guide* for more information about creating CodeWarrior projects)
- create a makefile (see the *RealView Compilation Tools* (RVCT) documentation for more information about using the ARM compilation tools).

3.12.3 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 the following option from the RealView Debugger main menu:

  Tools → Options...

  The Options window is displayed.

- Individual workspace settings, for individual users. To do this, select the following option from the RealView Debugger main menu:

  File → Workspace → Workspace Options...

  The Workspace Options window is displayed.

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**, **Japanese**, or **Chinese**.
   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. The Font dialog box is displayed.
   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 Chinese and Japanese languages, see *Font recommendations* on page 3-18.
   d. Click **OK** to close the Font dialog box.
6. Select **File** → **Save and Close** to save your changes and close the settings window.
7. Select **File** → **Exit** to exit RealView Debugger.
8. Start RealView Debugger again as described in *Starting RealView Debugger* on page 3-4.

   In the Code window, an `Encoding` toolbar is now displayed, shown in Figure 3-3.

Figure 3-3 Encoding toolbar

3.12.4 Configuring the pane 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 (see *Configuring the internationalization settings* on page 3-19). You can change the format only for the following elements in the Code window:

- File Editor pane, source code tabs, and text searches
- Output pane, all tabs
- individual variables in a Watch pane tab
- individual variables in the **Locals** tab and **Statics** tab of the Call Stack pane.

Changing the encoding format for the File Editor pane and searches

To change the encoding format for the File Editor pane, 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 pane

To change the encoding format for the Output pane:

1. Click the **Pane** menu for the Output pane.
2. Select **Format...** from the **Pane** menu. A List Selection dialog box is displayed.
3. Select the required encoding from the dialog box, for example, **UTF-8**.
4. Click **OK**.

Changing the encoding format for individual Watch pane entries

To change the encoding for an individual Watch pane entry:

1. Right-click on the Watch entry, and select **Format...** from the context menu. A List Selection dialog box is displayed.
2. Select the required encoding from the dialog box, for example, **UTF-8**.
3. Click **OK**.

Changing the encoding format for individual Call Stack pane entries

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

1. Select the required tab in the Call Stack pane, for example **Locals**.
2. Right-click on an entry in the tab, and select **Format...** from the context menu. A List Selection dialog box is displayed.
3. Select the required encoding from the dialog box, for example, **UTF-8**.
4. Click **OK**.

3.13 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.

RealView Debugger stores session details using your:

- *Workspace*
- *Startup file*
- *History file* on page 3-23.

3.13.1 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.

3.13.2 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 pane 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 the chapter that describes configuring workspace settings in the *RealView Debugger v3.0 User Guide*.

3.13.3 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 Open dialog boxes, for example Select File to Open, or Select File to Include Commands from.
- Up to 32 personal favorites such as variables, data values, watches, and breakpoints.

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 non-Windows platforms, the history file is only created if you create and save a favorite, for example a breakpoint or watchpoint. See Appendix B *RealView Debugger for Sun Solaris and Red Hat Linux* in the *RealView Debugger v3.0 User Guide* for details.

Appendix A
About Previous Releases

This appendix describes the major differences between the previous releases of RealView® Debugger, that is v1.8, v1.7, and v1.6.1. It contains the following sections:

- *Changes between RealView Debugger v1.8 and v1.7* on page A-2
- *Changes between RealView Debugger v1.7 and v1.6.1* on page A-9.

A.1 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 contains the following sections:

- *Updated documentation*
- *Trace, Analysis, and Profiling* on page A-3
- *Support for gcc built images* on page A-3
- *Changes to the GUI* on page A-4
- *Changes to the CLI* on page A-8
- *RealView ARMulator ISS support* on page A-8
- *RealMonitor support* on page A-8.

A.1.1 Updated documentation

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* now 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 now 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* on page A-3 for a summary).
- All documents now reflect the enhancements to the RealView Debugger GUI (see *Changes to the GUI* on page A-4 for a summary).
- The RealView Debugger online help is now in HTML format, and is displayed using your default web browser.

A.1.2 Trace, Analysis, and Profiling

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

- tracepoints are now categorized as unconditional or conditional
- the Analysis window has changed (see *Changes to the Analysis window* 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.1.3 Support for gcc built images

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.1.4 Changes to the GUI

The major changes are described in the following sections:

- *Changes to the Code window main menu structure*
- *Toolbar changes* on page A-6
- *Internationalization is now supported* on page A-6
- *Changes to pane views* on page A-6
- *Changes to breakpoints* on page A-7
- *Changes to tracepoints* on page A-7
- *Changes to editing facilities* on page A-7.

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 now available on the new **Target** menu
- connections are now available on the new **Target** menu
- logs and journals are now available on the **Tools** menu
- all workspace-related options are now 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 now 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 now 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** are now 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 **Complex Breakpoints** submenus options are now available from the **Breakpoints** → **Hardware...** submenu
- the **Processor Events** option is now available from the option **Processor Exceptions...**
- the **Include Commands from File...** option is now available on the **Tools** menu.

Tools The following changes have been made to this menu:

- the build-related options are now available on the new **Build** menu
- the workspace options are now 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 now 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 now available through the **RealView Debugger Help** option.

Toolbar changes

The Code window toolbar is now 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 now supported

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

- set the language to English (the default), Japanese, or Chinese
- 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 now 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 now 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 now 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 now only be used to set breakpoints. Therefore, this dialog box is now 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 now 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 now 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 now have more control over source code coloring. You can:
  - set both the foreground text color and the background color for various code elements
  - choose from a list of color schemes, including Visual Studio and CodeWarrior, that determines the default colors that are used
  - set colors for the additional code element identifiers, preprocessor keywords, and operators.
- You can now 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.1.5 Changes to the CLI

The following changes have been made to the CLI:

- Spaces are no longer allowed in connection names. Therefore, the RealView ICE is now 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 now supports the ETB11 port widths.

A.1.6 RealView ARMulator ISS support

Map files are now supported for *RealView ARMulator® ISS* (RVISS) connections through the Simulator Broker interface.

A.1.7 RealMonitor support

You can now 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.2 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 contains the following sections:

- *Updated documentation*
- *Advanced debugging facilities*
- *RealView ARMulator ISS*
- *Trace, Analysis, and Profiling* on page A-10
- *Enhanced RTOS support* on page A-10
- *New GUI elements* on page A-11.

A.2.1 Updated documentation

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

- *RealView Debugger Essentials Guide* for updated information for developers moving to RealView Debugger from 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.2.2 Advanced debugging facilities

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

A.2.3 RealView ARMulator ISS

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

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

Note

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

A.2.4 Trace, Analysis, and Profiling

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

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

Note

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

A.2.5 Enhanced RTOS support

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

Running System Debug

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

Thread-based breakpoints

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

RTOS visualization

This release sees new RTOS visualization features. This gives users an improved threads view in the Process Control pane and provides new menus and tabs in the Resource Viewer 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.2.6 New GUI elements

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

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

RealView Debugger v1.7 also includes:

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