RealView Debugger
Target Configuration Guide

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

Release Information
The following changes have been made to this document.

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

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

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

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

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

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

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

Web Address
http://www.arm.com
## Preface

About this book .................................................................................................................. vi
Feedback ............................................................................................................................. x

### Chapter 1  Introduction

1.1 About target connections and configuration .................................................. 1-2
1.2 Viewing board file properties ........................................................................ 1-5
1.3 Configuration files ............................................................................................. 1-11

### Chapter 2  Connecting to Targets

2.1 Working with the Connection Control window .............................................. 2-2
2.2 Managing connections ..................................................................................... 2-7
2.3 Connecting to a target ..................................................................................... 2-11
2.4 Failing to make a connection ........................................................................ 2-19
2.5 Disconnecting from a target ........................................................................... 2-21

### Chapter 3  Configuring Custom Targets

3.1 About target configuration ............................................................................ 3-2
3.2 The supplied .bcd files .................................................................................. 3-7
3.3 Creating new target descriptions .................................................................. 3-9
3.4 Example descriptions ...................................................................................... 3-24
Preface

This preface introduces the RealView® Debugger Target Configuration Guide. It contains the following sections:

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

RealView Debugger provides a powerful debugging tool for ARM® software projects. This book shows you how to configure RealView Debugger for your chosen debug target. This book also describes how target connections are managed and displayed in RealView Debugger. It contains:

- a description of the target configuration model used by RealView Debugger
- a description of target configuration using the RealView Debugger configuration facility
- a glossary of terms for users new to RealView Debugger.

Intended audience

This book has been written for developers who are using RealView Debugger to debug ARM-based development projects. It assumes that you are an experienced software developer. It 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.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction

This chapter introduces the connection and target configuration system that is used by RealView Debugger. It is recommended that you read this chapter.

Chapter 2 Connecting to Targets

This chapter describes how you connect to your target using the RealView Debugger connection control window. It expands on the introduction in the RealView Debugger v1.7 Essentials Guide, including details of the context menus and connection to targets in specific ways.
Chapter 3 Configuring Custom Targets

This chapter describes how you configure RealView Debugger with details of the memory and registers available on your custom target.

If you are using the ARM Integrator™ or Evaluator development platforms you do not have to configure the target because the required information is included with the product. See Chapter 2 Connecting to Targets for details of how to invoke this configuration.

If you are using another target, the information in this chapter enables you to set up configuration information for RealView Debugger to define the target memory map.

Chapter 4 Configuring Custom Connections

This chapter describes how you create new connection types if the connection types configured into the product are not suitable for your target. It also describes how you install and configure Remote Debug Interface (RDI) target interfaces into the RealView Debugger Connection Control window.

Chapter 5 Working with Remote Targets

This chapter describes how RealView Debugger handles remote connections. It includes information on using RealView Network Broker, and explains how connections to different targets are configured to enable network access.

Appendix and Glossary

Appendix A Configuration Settings Reference

Read this appendix for details on setting options to configure your targets and connections using RealView Debugger.

Glossary

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

Typographical conventions

The following typographical conventions are used in this book:

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

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

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

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

ARM publications

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

* RealView Debugger v1.7 Essentials Guide (ARM DUI 0181)
* RealView Debugger v1.7 User Guide (ARM DUI 0153)
* RealView Debugger v1.7 Project Management User Guide (ARM DUI 0227)
* RealView Debugger v1.7 Command Line Reference Guide (ARM DUI 0175)
* RealView Debugger v1.7 Extensions User Guide (ARM DUI 0174).

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

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


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

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

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

* RealView® ICE User Guide (ARM DUI 0155)
Other publications

For a comprehensive introduction to ARM architecture see:


For a detailed introduction to regular expressions, as used in the RealView Debugger search and pattern matching tools, see:


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


For more information about the JTAG standard, see:


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

http://www dspg com.

Contact information for the MaxSim Simulator from AXYS Design Automation, Inc. is available at http://www.axysdesign 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 email to errata@arm.com giving:
• the document title
• the document number
• the page number(s) to which your comments apply
• a concise explanation of your comments.

General suggestions for additions and improvements are also welcome.
Chapter 1
Introduction

This chapter introduces the connection and target configuration system used by RealView® Debugger. It contains the following sections:

- About target connections and configuration on page 1-2
- Viewing board file properties on page 1-5
- Configuration files on page 1-11.
1.1 About target connections and configuration

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

RealView Debugger uses a board file to access information about the debugging environment and the debug targets available. The board file, therefore, describes how RealView Debugger connects to, and interacts with, your debug targets.

The default board file is set up when you first install RealView Debugger. It is called rvdebug.brd and, along with files that it references, is stored in your default home directory. For details on your home directory, and what it contains, see The home directory on page 1-13.

This section describes target connection and configuration, and introduces the features of RealView Debugger that enable you to define these settings:

- Comparing target connection and target configuration
- Connection Control window on page 1-3
- Connection Properties window on page 1-4.

1.1.1 Comparing target connection and target configuration

RealView Debugger makes a distinction between:

**Target connection**

Describes how the debugger accesses your debug target.

**Target configuration**

Describes your debug target to the debugger, for example how memory is mapped.

Specifically, the board file enables you to specify high-level connection details such as:

- debugger to target connection details, such as interface type and instance, TAP controller positions, and connection interface address
- debugger actions taken when a connection is made, for example running commands and opening projects.

And low-level, target configuration details such as:

- target processor characteristics, for example processor type and endianess
- target peripheral register and memory configuration.
To do this, the default board file contains a series of elements arranged in a hierarchy. At the top level are connection entries or target vehicles that enable RealView Debugger to connect to different targets. As you drill down the connection, you can modify entries, or create new elements, that define the target configuration in more detail.

At the lowest level, RealView Debugger is able to access Extended Target Visibility (ETV) information about your debug target using a special group of settings, the Advanced_information block that is found in all the main groups.

RealView Debugger can also access Board/Chip definition files that contain ETV information about a particular board or chip as supplied by the manufacturer, including peripheral registers and memory regions.

### 1.1.2 Connection Control window

You can use the Connection Control window to make your first connection to a debug target. The RealView Debugger base product includes built-in configuration files to enable you to make a connection without having to modify any configuration details.

![Connection Control window](image)

The Server vehicle in the Connection Control window is also used to connect to DSP simulators, if you installed DSP support and you have the appropriate license. See the chapter describing DSP support in *RealView Debugger v1.7 Extensions User Guide* for more information.
1.1.3 Connection Properties window

The contents of the Connection Control window are defined by elements of the board file. To change target configuration, or to add new debug targets, you change the entries in the board file and this modifies the elements displayed in the Connection Control window. RealView Debugger provides a GUI interface, the Connection Properties window, to make these changes. See Connection Properties window on page 1-5 for more details.

Note

Do not configure the board file when the debugger is connected to a target.

The rest of this chapter describes RealView Debugger target configuration in more detail.

Getting more information

Appendix A Configuration Settings Reference provides a detailed reference that describes all the configuration settings available in the board file. It provides examples of when to use settings and describes how to make changes. This might be useful when you are working through the examples in this chapter and in the rest of this book.
1.2 Viewing board file properties

RealView Debugger provides the Connection Properties window to enable you to examine, and change, target connections and configuration details stored in the board file. This section describes these entries in more detail and contains the following sections:

- Connection Properties window
- Connection entries on page 1-7
- Configuration entries on page 1-8.

1.2.1 Connection Properties window

Select File → Connection → Connection Properties... to display the Connection Properties window, shown in Figure 1-2. This enables you to examine your current configuration settings and edit these settings to set up new connections and change target configuration.

![Connection Properties window](image)

**Figure 1-2 Connection Properties window**

---

**Note**

Do not make changes in the Connection Properties window when the debugger is connected to a target.
The main interface components of the Connection Properties window are:

**Main menu**  This contains:
- **File**  Displays the File menu where you can save the board file after you have made changes.
- **View**  Displays the View menu to toggle the display to show all the settings or only those that have been edited.
- **Help**  Displays the online Help menu.

**Save icon**  Click this icon to save the configuration settings file to disk. The name of the current file is shown as the first entry in the left pane.

**Description**  This field displays a one-line description about an entry selected in the panes below.

**List of Entries and Settings Values**

The left pane of the Connection Properties window, the List of Entries pane, shows configuration entries as a hierarchical tree with node controls. Groups of settings are associated with an icon to explain their function:

- **Red disk**  This is a container disk file. RealView Debugger uses this, for example, to specify an include file.

- **Yellow folder**  This is a parent group containing other groups (rules pages) and/or entries.

- **Rules page**  A rules page is a container for settings values that you can change in the right pane. This icon only appears in the left pane.

An asterisk (*) is placed at the front of an entry to show that it has changed from the default or that it was changed by RealView Debugger.

If you click on an entry in the left pane, a red box is drawn around it and the Description field is updated. At the same time, the right pane, the Settings Values pane, is updated to show the contents of the highlighted group.

--- **Note**

See the RealView Debugger online help topic *Changing Settings* for details on all the entries in the Connection Properties window.
RealView Debugger uses the board file to configure the target and the connection with two types of entry:

- **Connection entries**
- **Configuration entries** on page 1-8.

### 1.2.2 Connection entries

RealView Debugger sets up target connection entries in the board file based on targets it finds during installation, for example where you have installed RealView ICE, Multi-ICE, or RealMonitor, the appropriate DLL is autodetected by RealView Debugger. If you install a new DLL after installing RealView Debugger, for example ARM ADI, the display list is automatically updated for you.

The Connection Control window, shown in Figure 1-3, uses the connection entries in the board file to create a tree of possible connections and the processors that can be reached using them, for example ARM940T_0.

![Figure 1-3 Connection entries in the Connection Control window](image)

To see these connection entries you must install other ARM products because RealView Debugger does not include any connection software of its own. However, the debugger does include connection interface components, or target vehicles, to provide the interface to each target. For example, shown in Figure 1-3:

- the ARM-A-RR vehicle provides the interface to ARM RDI targets
- the Server vehicle is used to connect to simulators such as RVISS
- the ARM-ARM-NW vehicle provides the interface to RealView ICE targets.

Figure 1-3 shows the Connection Control window generated by the board file entries shown in Figure 1-2 on page 1-5. Each connection entry in the board file, for example CONNECTION=Multi-ICE or CONNECTION=RealView ICE, has a corresponding entry in the Connection Control window, for example Multi-ICE and RealView ICE.
You can create your own target connection groups, for example CONNECTION=RVI940T shown in Figure 1-2 on page 1-5. This new connection appears as an entry in the Connection Control window, that is RVI940T, shown in Figure 1-3 on page 1-7. You can create new groups based on existing connection groups or set these up using RealView Debugger defaults. You can also add any groups that are missing when you run RealView Debugger. See Chapter 4 Configuring Custom Connections for details on how to do this.

**Note**

For more details on the entries in the Connection Control window, see Working with the Connection Control window on page 2-2.

### 1.2.3 Configuration entries

Target configuration entries enable you to describe the target architecture to RealView Debugger. This makes it possible for the debugger to present peripheral registers in a more human-readable format, and enables operations involving target memory to take account of the target memory map, for example so that Flash memory can be accessed correctly.

Target configuration information, using BOARD, CHIP, and COMPONENT groups, is used to define a hierarchy, starting from the general board-level and becoming more specific, through whole chips to component modules on a chip. However, RealView Debugger does not distinguish, functionally, between the different group names and you can use them as you require.

**Using .bcd files**

Within the top-level board file, rvdebug.brd, you can have as many BOARD, CHIP, or COMPONENT entries as you require. However, there is a better way to store them.

When RealView Debugger starts up, it searches for files with the extension .bcd and loads them into a group called (*.bcd) Board/Chip Definitions, shown in Figure 1-4 on page 1-9.
Figure 1-4 Viewing .bcd files in the Connection Properties window

Configuration entries in files loaded into this group can be referenced from any connection. For example, you can define a connection that references the file AP.bcd, to access the ARM Integrator/AP board, and the file CM940T.bcd, to access the ARM CM940T processor core module registers and memory map. In this way, the connection defines a hierarchy of configuration details from the board level to the processor level.

This makes the target description independent of the connection used to access it, and makes it easier to reuse target descriptions in different debugging sessions.

Using the Advanced_Information block

Extended target visibility is possible using a special group, the Advanced_Information block that is found in almost all the main groups, shown in Figure 1-2 on page 1-5. For example, there is an Advanced_Information block in the CONNECTION=RealView ICE entry of the ARM-ARM-NW group, and in the BOARD=AP entry of the (*.bcd) Board/Chip Definitions group.

Note

In the board file, both target connection groups, for example CONNECTION=Multi-ICE, and target configuration groups, for example BOARD=AP, contain an Advanced_Information block. Although you can use the Advanced_Information block of the target connection group, it is suggested that you use target configuration groups and then reference these from the connection entry you are using. To ensure that these settings are used, do not change the default settings contained in the target connection group.
The search procedure, the way files are referenced, and the configuration options are described in more detail in Chapter 3 *Configuring Custom Targets.*
1.3 Configuration files

Default configuration files are supplied as part of the RealView Debugger base product that define standard ARM targets such as Integrator/AP. Configuration files are also created when you install additional ARM connection products, for example:

- If you install RealView ICE, configuration files are created for the standard ARM RV-msg DLLs included with this product.
- If you install Multi-ICE v1.4 or above, RealMonitor, or ADI, configuration files are created for the standard ARM RDI DLLs included with these products.
- If you are using a simulator such as RVISS, configuration files are created for these connection types.

1.3.1 What the configuration files contain

The configuration files that RealView Debugger stores in your home directory include the following files relating to this book:

*.brd  Top-level board files. Normally this is rvdebug.brd. This file includes the filenames of the other configuration files, for example .bcd and .jtg files. You might change this file when you save settings in the Connection Properties window (see the examples in Creating new target descriptions on page 3-9).

*.bcd  Board/Chip definition files supplied by hardware manufacturers. Files that contain configuration information for specific targets. By default, these files are used from the default settings directory, \etc. If you change a supplied file or you create your own, you are recommended to store them in your debugger home directory. These files contain per-chip and per-board settings as named configurations. You might change this file when you save settings in the Connection Properties window (see the examples in Creating new target descriptions on page 3-9).

*.cnf  Configuration files for ARM RDI target connections, for example Multi-ICE. You change one of these files when you modify the configuration of an RDI connection using the RDI configuration dialog box. These files are referenced by the .rbe files.
These files are used if you install new products such as RealView ICE. For example, the first time RealView Debugger runs after you install RealView ICE, the information contained in these files is copied into the board file. After this happens, the .prc file is ignored.

*.rbe
This file references the configuration file for ARM RDI targets. This file is where the *.cnf file is named, and it also extends the RDI settings with chip and board specific configuration settings. You might change this file when you save settings in the Connection Properties window.

*.rvc
Configuration files for RV-msg connections. You change the contents of one of these files when you modify the configuration of an RV-msg connection using the RVConfig dialog box (see the example in Configuring a RealView ICE interface unit on page 4-11).

1.3.2 Finding configuration files

This section describes how RealView Debugger directories are created when you install the base product, and how they are used to find the configuration files. It contains the following sections:

- The install directory
- The home directory on page 1-13
- The RealView Debugger search path on page 1-14
- Saving and restoring connection properties on page 1-15.

Note
For information about the other files that are stored in the debugger home directory see Appendix A Configuration Files Reference in RealView Debugger v1.7 Essentials Guide.

For information about differences when using RealView Debugger on non-Windows platforms, see Appendix B RealView Debugger on Sun Solaris and Red Hat Linux in RealView Debugger v1.7 User Guide.

The install directory

RealView Debugger must be able to locate the product installation directory so that it can locate program extensions, data, and configuration files stored there. Use the following settings to do this:

- The default location, for example:
  C:\Program Files\ARM\RVD\Core\1.7\build\win_32-pentium
• The RVDEBUG_INSTALL environment variable:
  set RVDEBUG_INSTALL=D:\ARM\RVD\Core\1.7\build\win_32-pentium
  For information about environment variables in RealView Debugger, see the chapter describing getting started in RealView Debugger v1.7 User Guide.

• The -install command line option:
  -install="D:\ARM\RVD\Core\1.7\build\win_32-pentium"

If the debugger cannot find the install directory, it terminates.

The shortcuts that are installed on the Windows Start → Programs menu might include an -install option to define the directory. If you did not install the debugger in the default location and you create your own shortcuts, or run rvdebug.exe from the command line, you must either include the same -install option or define the RVDEBUG_INSTALL environment variable globally, for example using the Windows Control Panel.

The home directory

The first time you run RealView Debugger after installing on Windows, 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 or copies files into this directory ready for your first debugging session.

The location of the home directory depends on the environment variables and command-line options defined when the debugger is started. RealView Debugger uses the first defined item from the following ordered list:

1. You can use -Home on the command line to specify an explicit path:
   -Home="C:\Documents and Settings\user_name\RVD"
   The home directory is C:\Documents and Settings\user_name\RVD

2. You can use the RVDEBUG_HOME environment variable to use a subdirectory of the Windows home directory:
   set RVDEBUG_HOME=C:\WinNT\Profiles\user_name\Application data\RVD
   For information about environment variables in RealView Debugger, see the chapter describing getting started in RealView Debugger v1.7 User Guide.

3. You can use the -install command line option on its own or together with -user or $USER:
   -install="D:\WinApp\RVD" -user="DevTeam"
   The home directory is D:\WinApp\RVD\home\DevTeam\.
You can use -user or $USER to specify an alternative for DevTeam.

4. You can use the RVDEBUG_INSTALL environment variable on its own or together with -user or $USER:
   set RVDEBUG_INSTALL=D:\WinApp\RVD\n   set USER=DevTeam
   The home directory is D:\WinApp\RVD\home\DevTeam.

5. The default location, for example:
   install_directory\RVD\Core\...\home\user_name\n
   **Note**
   If the debugger home directory location is defined, but the directory does not exist, the debugger creates it and copies the standard set of configuration files there from the default settings directory (\etc).

**RealView Debugger on Sun Solaris and Red Hat Linux**

For details on the debugger home directory for developers using RealView Debugger on Sun Solaris and Red Hat Linux see Appendix B in *RealView Debugger v1.7 User Guide*.

**The RealView Debugger search path**

RealView Debugger searches several directories for board files, including the default file, rvdebug.brd. The search path the debugger uses is:

1. The current working directory, sometimes called the Start In directory.

2. The location defined by the RVDEBUG_SHARE environment variable, if it is set.
   For information about environment variables in RealView Debugger, see the chapter describing getting started in *RealView Debugger v1.7 User Guide*.

3. The RealView Debugger home directory, described in *The home directory* on page 1-13, using the order specified there.

4. The default settings directory, \etc.

RealView Debugger searches all of these directories for workspace files and other configuration files. In particular, this is how *.bcd* files are found. Where two or more files with the same filename are found in more than one of the searched directories, the first file found is loaded and others are ignored.
Saving and restoring connection properties

When you are configuring RealView Debugger, you are recommended to keep backups of known-good configuration information before changing settings. There are backup systems you can use:

- You can rely on the automatic backup that RealView Debugger makes whenever it saves the Connection Properties window, for example rvdebug.brd.bak.
- You can copy specific files, for example rvdebug.brd, or the whole home directory to a backup area.

--- Note ---

It is recommended that you make backups before using the worked examples in the rest of this book.

---

**Using the automatic backup files**

If you edit a board file, or a Board/Chip definition file, RealView Debugger automatically renames the original file by adding a .bak file extension. Any previous backup copy of the file is deleted.

If you want to restore a backup file:

1. Exit the Connection Properties window without saving changes.
2. Delete the current file or files.
3. Rename the backup file to the original filename by deleting .bak from the name.

**Using manual file or directory backups**

For safer backups, you are recommended to make tape or disk copies of the files in another place. The simplest policy is to save the whole directory when you make a backup, but restore individual files when you want to revert changes.

--- Note ---

If you restore the whole directory, then as well as restoring the Connection Properties configuration information, you restore preferences that you might not want to change, for example workspace properties, project properties, and window layout.

---

Creating a directory backup requires you to locate and copy the home directory to a safe place. You do not have to exit the debugger to do this.
Restoring a previous backup file by file requires you to:

1. Locate the backup that you wish to restore from and the debugger home directory that RealView Debugger is using for your session.  
   See The home directory on page 1-13 for more information about the location of your debugger home directory.

2. Determine the files to restore.

3. Copy the backup files to the debugger home directory.

Deciding which files to restore depends on the type of configuration change you have performed. These hints might help:

- If you have changed everything, or you are not sure what to restore, select all the files listed in What the configuration files contain on page 1-11 to restore the Connection Properties window to its original state.

- If you have configured or reconfigured a chip or board using BOARD, CHIP or COMPONENT groups, select appropriate files from the *.bcd set. If you have created new *.bcd files in your debugger home directory, you might also want to delete them from that directory. However, a *.bcd file is not used unless it is explicitly referenced.

- If you have changed RV-msg connection settings, for example by changing items in the RVConfig dialog box, select the *.rvc files for the RealView ICE connections you have reconfigured. These files are saved by default in your RealView ICE installation directory.

- If you have only changed RDI connection settings, for example by changing items in the Multi-ICE configuration dialog box, select the *.cnf files for the RDI connections you have reconfigured.

- If you have changed the linkage between RDI connections and your target, select the *.rbe file for the RDI connections you have reconfigured.
Chapter 2
Connecting to Targets

This chapter describes how to use the Connection Control window to view connection details. It describes how to connect to a target using the default configuration files, and how to disconnect. It contains the following sections:

- Working with the Connection Control window on page 2-2
- Managing connections on page 2-7
- Connecting to a target on page 2-11
- Failing to make a connection on page 2-19
- Disconnecting from a target on page 2-21.
2.1 Working with the Connection Control window

The Connection Control window, shown in Figure 2-1, enables you to make connections, change existing connections, and configure new ones if required.

You can display the Connection Control window in the following ways:

- Click on the blue hyperlink in the File Editor pane, if available.
- Select File → Connection → Connect to Target... from the Code window.
- Click the Connection Control button, in the Connection group on the Actions toolbar. If the window is hidden, click the button twice.

Figure 2-1 shows a Connection Control window where you are licensed to work with multiprocessor debug target systems using different processor families. In this example, the window contains two tabs, Connect and Synch. In single processor debugging mode, these tabs are not visible and the window is used only to make connections.

See the chapter describing working with multiple targets in RealView Debugger v1.7 Extensions User Guide for details on using the Synch tab to synchronize processor operations.

This section describes the Connection Control window in more detail:

- Groups in the Connection Control window on page 2-3
- Using the Connection Control window on page 2-4
- Changing your board file on page 2-6.
2.1.1 Groups in the Connection Control window

Figure 2-2 shows the Connection Control window for the default board file, stored in \home\My_user_name\rvdebug.brd, in the root installation. In this example, some groups have been expanded and a connection has been made (see Expanding and collapsing groups on page 2-4 for details). If you select a group, a box is drawn around it.

![Connection Control window](image)

**Figure 2-2 Groups in the Connection Control window**

The Connection Control window is arranged in two columns or panes, Name and Description. Connection and target details are displayed in the left pane as a hierarchical tree with node controls, + and -. Connection and target details are displayed in the Name column as a hierarchy of entries:

**Target vehicles**

Top-level groups are supported target vehicles as specified by the target configuration settings, for example Server for targets that use the RealView Connection Broker or ARM-ARM-NW for RealView® ICE targets. See Working with target vehicles on page 2-7 for more details on working with top-level groups.

**Access-provider connections**

Second-level groups show the type of vehicle, or the debug target interface used to support the connection, for example RealView ICE, the ARM® JTAG debug tool for embedded systems, or Remote_A, the Angel debug monitor. The ARM RDI second-level groups are also enabled in your board file when you install ARM Developer Suite™ (ADS), ARM ADI, or Multi-ICE®. See Working with access-providers on page 2-8 for more details on working with second-level groups.
Endpoint connections

Third-level entries show the target processors that are made available by the access provider, for example the ARM940T core connected using RealView ICE.

Each endpoint connection is accompanied by a check box to show the current state of the connection. When connected, this check box is checked, shown in Figure 2-2 on page 2-3. Where no connections have been made, the check boxes are unchecked.

RealView Debugger uses icons to help you identify the types of entries in the Name column.

2.1.2 Using the Connection Control window

The Connection Control window shows all the connections available to you as specified in your board file and the configuration files it references. The window title bar shows the location of the board file being used. In these examples, this is the default file, stored in `\home\My_user_name\rvdebug.brd` in the root installation.

Expanding and collapsing groups

Expand and collapse the groups in the Connection Control window by clicking on the plus sign or the minus sign at each node in the Name tree. Figure 2-3 on page 2-5 shows a Connection Control window after groups have been expanded and a connection has been made. If you select a group, a box is drawn around it.

Context menus are available to change the way entries are displayed. To expand the top-level groups, right-click on a group, for example ARM-ARM-NW, and select Expand Vehicles from the context menu. To expand the second-level entries, right-click on an entry, for example RealView ICE, and select Expand. To collapse them again, select Collapse from the context menu, shown in Figure 2-3 on page 2-5.

Note

If you are not connected, you can also expand, or collapse, entries by double-clicking.
Connecting to Targets

You can connect to a target by checking the connection state check box, shown in Figure 2-3. If you collapse connected (checked) entries, RealView Debugger does not complete the collapse request so that a connection is not hidden. Instead, the control is grayed out.

The Connection Control window shows available connections as defined by the target configuration settings. When you first install RealView Debugger, this is based on information about the available target processors as defined in the default configuration files. You can connect to any of the default connections, shown in the Connection Control window, without making any further changes to configuration files.

**Note**

You must configure certain targets before you can make your first connection, for example RealView ICE, Multi-ICE, or RealMonitor. See Chapter 4 Configuring Custom Connections for more details.

Local host simulators are available immediately from the Connection Control window. If you expand the ‘localhost Simulator Broker’ entry, ready to connect to a simulator, RealView Debugger starts RealView Connection Broker in local mode to manage your connection (see Working with Simulator Broker connections on page 4-27 for more details).
2.1.3 Changing your board file

If you work with a variety of targets and connections you might set up and save several board files, so that you can easily switch RealView Debugger from one to another. You can change the board file being used for the current session in two ways:

- right-click on a top-level entry in the Connection Control window, for example ARM-A-RR, and select the option Select Board-File... from the context menu
- change your workspace settings file to start the session with a specified board file, see the chapter on configuring workspaces in RealView Debugger v1.7 User Guide for details.

**Note**

To ensure a consistent target view, do not change the board file when the debugger is connected to a target.
2.2 Managing connections

Use context menus in the Connection Control window to:

- manage the displayed connections
- establish a connection to your chosen debug target
- use a different board file.

There are several context menus that can be displayed, depending on the entry you select:

- Working with target vehicles
- Working with access-providers on page 2-8
- Working with JTAG targets on page 2-10.

2.2.1 Working with target vehicles

Right-click on a top-level entry, for example ARM-A-RR, Server or ARM-ARM-NW, to see the context menu, shown in Figure 2-4.

![Figure 2-4 Target vehicle menu](image)

The options available from this menu enable you to manage the vehicle used to make the connection. What options are available, and how you use these options, depends on the vehicle you are using:

**Collapse All**  Collapses the hierarchical tree to display only the top-level entries.

**Expand Vehicles**  Expands the hierarchical tree to display the contents of top-level groups, and any previously expanded second-level groups.

**Connection Properties...**  Displays the Connection Properties window to amend current configuration details or to add target configurations (see Viewing board file properties on page 1-5 for details).

**Add/Remove/Edit Devices...**  Displays the RDI Target List window, used to add or remove RDI target DLLs from the ARM-A-RR vehicle, or to edit the configuration of one of these targets.
Connecting to Targets

See Adding and configuring RDI targets on page 4-14 for full details on using this option.

Note
This is only appropriate for the ARM-A-RR vehicle. Do not use this menu option to add RV-msg connections.

Select Board-File...
Displays the Select Board-File to Read dialog box, where you can specify a new board file for target configuration in this session.
If you change the board file in this way, the new board is then used for all connections available with the chosen vehicle, that is local Multi-ICE connections and any remote connections that you might specify.

Note
To ensure that configuration information is maintained, do not change the active board file if:
• the Connection Properties window is open
• you are connected to a debug target.

If there are any connections established, for any vehicle, the menu includes the option:

Disconnect All
Disconnects all connected targets, using the default disconnect mode, and collapses the tree to display the top-level and second-level entries.

2.2.2 Working with access-providers
Right-click on a second-level entry, for example RealView ICE or Multi-ICE, to see the access-provider context menu, shown in Figure 2-5.

Figure 2-5 Access-provider menu
The options available from this menu enable you to manage the debug interface used to support the connection. What options are available, and how you use these options, depends on the interface you are using:

**Expand**
Expands the hierarchical tree to display the contents of the second-level group. This then changes to **Collapse**.

When you expand, RealView Debugger initializes and queries the selected interface. This might:

- cause a short delay
- result in error messages being displayed if a previously configured connection no longer operates because, for example, the JTAG cable is not connected or where a RealView ICE configuration file cannot be located
- result in the connection configuration dialog box being displayed for the interface software, for example Multi-ICE.

**Connection Properties...**
Displays the Connection Properties window to amend current configuration details or to add target configurations (see Viewing board file properties on page 1-5 for details).

**Add/Remove/Edit Devices...**
Displays the RDI Target List window, used to add or remove RDI target DLLs from the ARM-A-RR vehicle, or to edit target configuration files.

See Adding and configuring RDI targets on page 4-14 for full details on using this option.

**Note**
This is only appropriate for the ARM-A-RR vehicle. Do not use this menu option to add RV-msg connections.

**Configure Device Info...**
Enables you to configure the debug target, for example by displaying the RVConfig dialog box where you can configure RealView ICE debug targets.

If you want to configure Connection Broker connections, for example RealView ARMulator® ISS (RVISS), you must:

1. Expand the second-level localhost entry.
2. Right-click on the simulator you are using, for example new_ARM.
3. Select **Configure Device Info...** from the context menu.
2.2.3 Working with JTAG targets

The JTAG target context menu is available for connections that use On-Chip Debugging (OCD) JTAG-based connections, such as Multi-ICE direct connect.

Note

When installing RealView Debugger, you must choose the Custom option and install support for DSP. Do this to ensure that the required JTAG files are available to enable connection using Multi-ICE direct connect.

Where this type of connection is available, right-click on a second-level entry, for example ARMOAK_MICE, to see the context menu, shown in Figure 2-6.

The options available from this menu are:

Expand

Expands the hierarchical tree to display the contents of the second-level group. This then changes to Collapse.

Connection Properties...

Displays the Connection Properties window to amend current configuration details or to add target configurations (see Viewing board file properties on page 1-5 for details).

Test JTAG...

Enables you to test that a JTAG connection exists to the devices specified for this emulator. This is useful when troubleshooting the connection setup.

For details on using these menu options to configure different targets, see Chapter 4 Configuring Custom Connections.
2.3 Connecting to a target

RealView Debugger offers different ways to connect to your debug target:

- Using the Connection Control window
- Setting connect mode on page 2-13
- Connecting to multiple targets on page 2-17
- Including the connection in the workspace on page 2-17
- Using CLI commands on page 2-17.

If you are already connected to a target processor in single processor debugging mode, making a new connection gives the option to disconnect the existing connection. The auto-disconnect does not occur until the new connection is successfully established. However, because you are in single processor debugging mode, you cannot make a second concurrent connection and so the auto-disconnect is not completed (see Using auto-disconnect on page 2-22 for details).

--- Note ---
You must configure certain debug targets before making your first connection, for example RealView ICE or Multi-ICE. If you try to connect where the target has not been configured, RealView Debugger displays a dialog box to indicate that the connection has failed. See Failing to make a connection on page 2-19 for more details.

2.3.1 Using the Connection Control window

Select File → Connection → Connect to Target... from the Code window main menu, or click the Connection Control button, to display the Connection Control window where you can connect to your debug target.

You can connect to a target in the following ways:

- Double-click on an unconnected connection entry.
- Select the check box for a required connection entry so that it is checked.
- Right-click on a connection entry and select Connect from the Connection context menu, shown in Figure 2-7 on page 2-12.
- Right-click on a connection entry and select Connect (Defining Mode)... from the Connection context menu, shown in Figure 2-7 on page 2-12.
Connecting to Targets

Note

You must use this option if you do not want the processor to be stopped when you connect to a target. For more information on defining the connection mode, see Setting connect mode on page 2-13.

For example, to connect to the default ARM7TDMI® core using RVISS:

1. Display the Connection Control window.
2. Expand the top-level Server vehicle.
3. Expand the second-level localhost entry.
4. Right-click the processor connection, for example the default new_ARM, to see the Connection menu, shown in Figure 2-7.

![Connection menu](image)

5. Select Connect.

Note

When you connect to a multiprocessor JTAG target, for example a RealView ICE target with two or more processors, the order the processors appear in is neither alphabetical nor based on the TAP number. The order is chosen by the debugger.

If the chosen vehicle supports a target that has not been configured, a dialog box is displayed to indicate that the connection has failed. This enables you to configure the target first and then connect. See Failing to make a connection on page 2-19 for more details.

RealView Debugger connects to the specified target using the default connection mode. You can, however, specify the connection mode to use, see Setting connect mode on page 2-13.
Other options on the **Connection** menu are:

**Configure Device Info...**
Select this option to display the configuration dialog box for the target. You must configure certain targets before you can make your first connection, for example RealView ICE, Multi-ICE, or RealMonitor. See Chapter 4 *Configuring Custom Connections* for more details.

**Connection Properties...**
Select this option to display the Connection Properties window described in *Viewing board file properties* on page 1-5.

---
**Note**

Be aware that the default connection behavior for RealView ICE differs from that for Multi-ICE. You can override this to define the connection behavior (see *Setting connect mode* for details).

### 2.3.2 Setting connect mode

When you connect to a target, RealView Debugger attempts to establish the connection using the default connect mode, that is **No Reset and Stop**. This mode is used when you connect from the Connection Control window in any of the following ways:

- Double-click on an unconnected connection entry.
- Select the check box for a required connection entry so that it is checked.
- Right-click on a connection entry and select **Connect** from the **Connection** context menu, shown in Figure 2-7 on page 2-12.

Before connecting, RealView Debugger checks to see if a user-defined connect mode has been specified in your board file (see *User-defined connect mode* on page 2-14 for details). If such a setting is found, it overrides the default connect mode and is used instead. If there is no connect mode specified in your target configuration file, RealView Debugger proceeds to connect using the default connect mode.

However, the connect mode that is actually used depends on the target hardware, the target vehicle, and the associated interface software that manages the connection. In some cases, the interface cannot complete the connection using the mode requested, for example where your RealView ICE unit configuration conflicts with the connect mode. In this case, the RealView ICE unit determines the connect mode and makes the connection. Similarly, some ARM processors require a reset before you can connect, for
example XScale™, whereas other cores have no such restriction. RealView Debugger displays a warning message to say that the requested connect mode cannot be honored and tells you what connection mode was used instead.

--- Note ---
This behavior also applies to disconnect mode, for example mode substitution might occur when you try to disconnect from a running target. See Setting disconnect mode on page 2-23 for more details.

RealView Debugger provides a mechanism to enable you to override both the default connect mode and any user-defined settings in your board file. In this way, you can control the way a processor starts for individual connections. To do this you must select the option Connect (Defining Mode)... from the Connection context menu in the Connection Control window (shown in Figure 2-7 on page 2-12). See Connect (Defining Mode)... on page 2-15 for details on how to use this option.

User-defined connect mode

Before connecting, RealView Debugger checks to see if a user-defined connect mode has been specified by the Connect_mode setting in the Advanced_Information block in your board file, or in any .bcd file linked to the connection. If such a setting is found, it becomes the default connect mode for this connection and is used if you connect in any of the following ways:

- Double-click on an unconnected connection entry.
- Select the check box for a required connection entry so that it is checked.
- Right-click on a connection entry and select Connect from the Connection context menu, shown in Figure 2-7 on page 2-12.

--- Note ---
Be aware that the default connection behavior for RealView ICE differs from that for Multi-ICE. Set connect mode in your target configuration file to ensure that you connect to your chosen target in the correct way.

For more information about setting connect mode in your board file, see:

- About target configuration on page 3-2 in Chapter 3 Configuring Custom Targets
- the example Specifying connect and disconnect mode on page 3-32 in Chapter 3 Configuring Custom Targets
Connect (Defining Mode)...

RealView Debugger enables you to control the way a processor starts when you connect and so override any connection mode settings stored elsewhere. This is useful when debugging multiprocessor debug target systems or multithreaded applications, but can also be used when debugging a single processor target system, for example using RealMonitor.

To define a connection mode using the option Connect (Defining Mode)...

1. Right-click on a processor connection entry, for example ARM940T_0, using RealView ICE, to display the Connection menu, shown in Figure 2-7 on page 2-12.
2. Select Connect (Defining Mode)… to display the Connect Mode list selection box shown in Figure 2-8.

![Figure 2-8 Connect Mode selection box](image)

The state options shown depend on the interface software making the connection, but are always one or more from:

**No Reset and Stop**

Does not submit a processor reset but explicitly halts any process currently running by issuing a Stop command.

**No Reset and No Stop**

Does not submit a processor reset or halt any process currently running.

**Reset and Stop**

Submits a processor reset and explicitly halts any process currently running by issuing a Stop command.

- the description of the Advanced Information block in Appendix A Configuration Settings Reference.
Reset and No Stop

Submits a processor reset but does not halt any process currently running.

Highlight the required state and click **OK**.

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

---

**Default state**

The selection box, shown in Figure 2-8 on page 2-15, offers a series of state options to make the connection. The options shown depend on the interface software that manages the target connection and so vary depending on the connection entry you select in the Connection Control window. In this example, the selection box contains an entry **No Reset and Stop (default)**. This is either the:

- default connect mode chosen by the debugger (see *Setting connect mode* on page 2-13 for details)
- user-defined connect mode specified in the target configuration file (see *User-defined connect mode* on page 2-14 for details).

Any user-defined connect mode takes precedence over the default connect mode chosen by the debugger.

If you click **Cancel**, RealView Debugger establishes the connection using the connect mode shown as the default. Where this mode is not supported by your target, a different connect mode is used and you are warned about the substitution.

--- Note
This selection box is also displayed if you specify a prompt as your user-defined connect mode. In this case, there is no default entry (see *Using a prompt* for details).

---

**Using a prompt**

You can specify a user-defined connect mode to display a prompt to enable you to choose the connection mode for each connection. In this case, RealView Debugger displays the Connect Mode selection box shown in Figure 2-8 on page 2-15. However, where the prompt setting is defined, the state options offered are all supported by the target vehicle and so do not include a default option.
Note

If a prompt is specified in your board file, or in any .bcd file linked to the connection, it takes priority over any other user-defined connect mode setting. This prompt-first rule holds true regardless of where the setting is in the configuration hierarchy.

For more information about using a prompt, see:

- the example Specifying connect and disconnect mode on page 3-32 in Chapter 3 Configuring Custom Targets
- the description of the Advanced Information block in Appendix A Configuration Settings Reference.

2.3.3 Connecting to multiple targets

If you are licensed to work in multiprocessor debugging mode, you can make multiple target connections. In single processor debugging mode, however, you can make only one connection at a time, and making a second connection fails.

For details on working with multiprocessor connections see the multiprocessing chapter in RealView Debugger v1.7 Extensions User Guide.

As supplied in the base product, the default board file rvdebug.brd enables you to connect to one or more preconfigured debug targets on your local workstation. You can connect to these targets without making any changes to this board file.

For details on configuring different types of target, see Chapter 4 Configuring Custom Connections.

2.3.4 Including the connection in the workspace

If you exit the debugger with an active connection, a record of the connection details is kept in the active workspace. The next time that workspace is active when the debugger starts, the debugger attempts to set up the previous connection again. See the chapter describing configuring workspaces in RealView Debugger v1.7 User Guide for more details.

2.3.5 Using CLI commands

The CONNECT and RUN commands can be used to make a connection to your debug target.

You can use the CLI CONNECT command to make numbered connections, for example:

```
connect,route 1
```
Connecting to Targets

connect 6

where the connection id is used to identify the target.

You can also use the CONNECT command to specify the connect mode, for example:
connect, reset, halt 12

You can connect to remote connections in the same way.

**Making named connections**

You can also make named connections, for example:

connect @new_ARM
connect @new_OAK@Remote_Debug
connect @new_ARM@Remote_MICE

Specify the route name as defined in the board file, that is as it appears in the Connection Control window. Use the access-provider to avoid ambiguity, for example:

connect @new_ARM@localhost
connect @new_ARM@Remote_ARM_Debug

See the description of the CONNECT command in *RealView Debugger v1.7 Command Line Reference Guide* for full details on connecting to targets this way.
2.4 Failing to make a connection

When you click inside a check box, in the Connection Control window, RealView Debugger might fail to connect to the chosen debug target. This might be due to one of the following reasons:

- You do not have a valid license to use the debug target. RealView Debugger displays a message if you do not have a valid license.
- The debug target is not installed or the connection is disabled.
- The target hardware is in use by another user.
- The connection has been left open by software that exited incorrectly.
- The target has not been configured, or a configuration file cannot be located.
- The target hardware is not powered up ready for use.
- The target is on a scan chain that has been claimed for use by something else.
- The target hardware is not connected.

If RealView Debugger attempts to make a connection and fails, it normally displays a list selection box to offer possible actions, shown in Figure 2-9. Some endpoint connections, such as Multi-ICE, might also display their own dialog boxes or messages.

![Figure 2-9 Failing to make a connection](image)

The message displayed at the top of the selection box indicates the type of problem encountered by RealView Debugger. What message is displayed, and what options are available, depends on the interface you are using. The options are:

**Retry...** If you have identified the cause of the failure and corrected it, for example you have connected a board or switched on power, you can select this option and click **OK** to connect.
Connecting to Targets

Edit Board file...

You can select this option and click **OK** to close the list selection box and display the Connection Properties window where you can edit your target configuration details (see **Viewing board file properties** on page 1-5 for details).

Save the new settings and then close the Connection Properties window before trying to make the connection using the Connection Control window.

Configure Device Info...

Select this option to display the configuration dialog box for the target.

You must configure certain targets before you can make your first connection, for example RealView ICE, Multi-ICE, or RealMonitor. See Chapter 4 Configuring Custom Connections for more details.

Click **Cancel** to close the message box and abandon the connection.

### 2.4.1 Troubleshooting

This section helps you to identify and fix connection problems you might encounter:

- The selection box, shown in Figure 2-9 on page 2-19, might include the entry **Display list of possible problems**.... RealView Debugger might display this option if there are known problems with solutions to apply. Selecting the option displays a message box containing a list of possible causes for the failure to connect. The text describes ways to fix the problem.

  This list provides only suggestions and might not be applicable to your debug target.

- If your working versions of configuration files are accidentally erased, or become corrupted, RealView Debugger might be unable to use them. See **Troubleshooting** on page 3-57 for information describing how to recover from this situation.

- If you are using RealView ICE and see a message asking you to **kill all other connections**.... ensure that the required connection is available before terminating other connections. See **RealView ICE User Guide** for more details.

- If you are using Multi-ICE and see a message asking you to **reconnect to the server**, you must disconnect from all processors using the Multi-ICE connection and then reconnect them. See **Multi-ICE User Guide** for more details.
2.5 Disconnecting from a target

There are several ways to disconnect when working with a target. Choosing the most appropriate method depends on:

- the number and attachment of Code windows
- which window has the focus when the disconnection option is used
- the state of the currently connected processor, and process if running
- the required state of the processor, or process, following disconnection.

Code windows are not closed on disconnecting but their contents might change depending on the data they contain. For example any loaded images are unloaded and so associated source files close and entries displayed in a Register, Memory, or Process Control pane are cleared whereas entries in the Watch pane remain unchanged. This behavior depends on the update options you set for the window and the disconnect state of the target processor.

For details on disconnecting during multiprocessor debugging sessions see the multiprocessing chapter in RealView Debugger v1.7 Extensions User Guide.

If you are working with projects, any open projects do not close if you disconnect from a debug target. Even where a project is bound to the connection, it does not close if you disconnect. However, it is unbound and its details are no longer shown in the Process Control pane.

For details on working with projects see the chapter describing project binding in RealView Debugger v1.7 Project Management User Guide.

RealView Debugger provides different ways to disconnect from a target:

- Using auto-disconnect on page 2-22
- Using the File menu on page 2-22
- Using the Connection Control window on page 2-22
- Setting disconnect mode on page 2-23
- Disconnecting by exiting on page 2-28
- Using the CLI on page 2-30.

--- Note ---

Be aware that the default disconnection behavior for RealView ICE differs from that for Multi-ICE. You can override this to define the disconnection behavior, using the Advanced_Information block in your board file, or in any .bed file linked to the connection. See Setting disconnect mode on page 2-23 for details.
2.5.1 Using auto-disconnect

In single processor debugging mode, if you are already connected to a debug target processor, making a new connection gives you the option to disconnect the existing connection. The auto-disconnect does not occur until the new connection is successfully established.

However, if the target is running, RealView Debugger warns you and asks for confirmation to auto-disconnect before checking that you have the multiprocessor extension to enable you to make multiple connections. Because, you are in single processor debugging mode, you cannot make a second concurrent connection and so the auto-disconnect is not completed and the target state is unchanged.

2.5.2 Using the File menu

If you are connected to a single debug target processor, you can disconnect from the current connection, either:

- Select File → Connection → Disconnect from the Code window main menu.
- Click the Disconnect button from the Connection group on the Actions toolbar.

This has the following results:

- the current connection is terminated immediately
- any windows attached to the current connection are unattached
- title bars and Color Boxes for all unattached windows are updated.

To close any unwanted windows, select File → Close Window from the main menu.

2.5.3 Using the Connection Control window

Select File → Connection → Connect to Target... from the Code window main menu, or click the Connection Control button, to display the Connection Control window where you can disconnect from your debug target.

You can disconnect from a target in the following ways:

- Double-click on a connected entry.
- Select the check box for a required entry so that it is unchecked.
- Right-click on a connection entry and select Disconnect from the Disconnection context menu, shown in Figure 2-10 on page 2-23.
- Right-click on a connection entry and select Disconnect (Defining Mode)... from the Disconnection context menu, shown in Figure 2-10 on page 2-23.
Connecting to Targets

This option is also available from the **Connection** submenu of the **File** menu.

--- **Note** ---

You must use this option if you do not want the processor to be stopped when you disconnect from a target. For more information on defining the disconnection mode, see *Setting disconnect mode*.

![Disconnection menu](image)

**Figure 2-10 Disconnection menu**

Using any of these methods immediately terminates the connection and updates the Code window display and the active connections list. This has the following results:

- the specified connection is terminated immediately
- any windows attached to the current connection are unattached
- title bars and Color Boxes for all unattached windows are updated.

RealView Debugger disconnects from the specified target using the default disconnection mode. You can, however, specify the disconnection mode to use, see *Setting disconnect mode*.

### 2.5.4 Setting disconnect mode

When you disconnect from a target, RealView Debugger attempts to disconnect using the default disconnect mode, that is **As-is with Debug**. This mode is used when you disconnect from the Connection Control window in any of the following ways:

- Double-click on a connected connection entry.
- Select the check box for a required connection entry so that it is unchecked.
- Right-click on a connection entry and select **Disconnect** from the **Disconnection** context menu, shown in Figure 2-10.
This default disconnect mode is also used when you disconnect from the Code window using either:

- Select File → Connection → Disconnect from the Code window main menu.
- Click the Disconnect button from the Connection group on the Actions toolbar.

Before disconnecting, RealView Debugger checks to see if a user-defined disconnect mode has been specified in your board file (see User-defined disconnect mode for details). If such a setting is found, it overrides the default disconnect mode and is used instead. If there is no disconnect mode specified in your target configuration file, RealView Debugger proceeds to disconnect using the default disconnect mode.

However, the disconnect mode that is actually used depends on the target hardware, the target vehicle, and the associated interface software that manages the connection. In some cases, the interface cannot complete the disconnection using the mode requested, for example where your RealView ICE unit configuration conflicts with the disconnect mode. In this case, the RealView ICE unit determines the disconnect mode and disconnects. RealView Debugger displays a warning message to say that the requested disconnect mode cannot be honored and tells you what disconnection mode was used instead.

---

**Note**

This behavior also applies to connect mode, for example mode substitution might occur when you try to connect to a running target. See Setting connect mode on page 2-13 for more details.

---

RealView Debugger provides a mechanism to enable you to override both the default disconnect mode and any user-defined settings in your board file. In this way, you can control the way a processor stops when you disconnect. To do this you must select the option Disconnect (Defining Mode)... from the Disconnection context menu in the Connection Control window (shown in Figure 2-10 on page 2-23) or from the File menu from the Code window main menu. See Disconnect (Defining Mode)... on page 2-25 for details on how to use this option.

### User-defined disconnect mode

Before disconnecting, RealView Debugger checks to see if a user-defined disconnect mode has been specified by the disconnect_mode setting in the Advanced_Information block in your board file, or in any .bcd file linked to the connection. If such a setting is found, it becomes the default disconnect mode for this connection and is used if you disconnect in any of the following ways:

- Double-click on a connected connection entry.
- Select the check box for a required connection entry so that it is unchecked.
Right-click on a connection entry and select **Disconnect** from the **Disconnection** context menu, shown in Figure 2-10 on page 2-23.

Select **File → Connection → Disconnect** from the Code window main menu.

Click the **Disconnect** button from the Connection group on the Actions toolbar.

---

**Note**

Be aware that the default disconnection behavior for RealView ICE differs from that for Multi-ICE. Set disconnect mode in your target configuration file to ensure that you disconnect from your chosen target in the correct way.

For more information about setting disconnect mode in your board file, see:

- *About target configuration* on page 3-2 in Chapter 3 *Configuring Custom Targets*
- the example *Specifying connect and disconnect mode* on page 3-32 in Chapter 3 *Configuring Custom Targets*
- the description of the *Advanced Information* block in Appendix A *Configuration Settings Reference*.

---

**Disconnect (Defining Mode)...**

RealView Debugger enables you to control the way a processor stops when you disconnect and so override any disconnection mode settings stored elsewhere. This is useful when debugging multiprocessor debug target systems or multithreaded applications, but can also be used when debugging a single processor target system, for example to download your application and leave it running without the debugger connected.

To set the disconnect mode use either:

- **File → Connection → Disconnect (Defining Mode)...**
- the option **Disconnect (Defining Mode)...** from the Connection Control window **Disconnection** menu, shown in Figure 2-10 on page 2-23.

To define a disconnect mode using the option **Disconnect (Defining Mode)...**:

1. Right-click on a connected connection entry, for example **ARM940T_0**, using RealView ICE, to display the **Disconnection** menu, shown in Figure 2-10 on page 2-23.

2. Select **Disconnect (Defining Mode)...** to display the Disconnect Mode list selection box shown in Figure 2-11 on page 2-26.
The state options shown depend on the interface software managing the connection, but are always one or more from:

**As-is without Debug**

Leaves the target in its current state, whether stopped or running, and removes any debugging controls such as software breakpoints.

If this leaves the processor running, any defined breakpoints are disabled. This means the program does not enter debug state after the debugger has disconnected.

**As-is with Debug**

Leaves the target in its current state, whether stopped or running, and maintains any debugging controls such as software breakpoints.

If this leaves the processor running, any defined breakpoints are still active. This means the program might enter debug state after the debugger has disconnected, depending on the code paths the program takes.

Highlight the required state and click **OK**. This has the following results:

- the current connection is disconnected
- the command is reflected in the **Cmd** tab of the Output pane
- the toolbar state group is set to **Unknown**
- any windows attached to the current connection are unattached
- title bars and Color Boxes for all unattached windows are updated.

The state options specify that the target is left in the current state, that is running or stopped. There is no option to stop the processor before disconnecting. To ensure that the processor is left in a particular state, you should stop (or start) the processor before disconnecting.
Remember that the disconnect mode that is actually used depends on the target hardware, the target vehicle, and the associated interface software that manages the connection. In some cases, the interface cannot complete the disconnection using the mode requested, for example where your RealView ICE unit configuration conflicts with the disconnect mode. In this case, the RealView ICE unit determines the disconnect mode and disconnects. RealView Debugger displays a warning message to say that the requested disconnect mode cannot be honored and tells you what mode was used instead.

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

Default state
The selection box, shown in Figure 2-11 on page 2-26, offers one or more state options to disconnect from your target. The options shown depend on the interface software that manages the target connection and so vary depending on the connection entry you select in the Connection Control window. In this example, the selection box contains an entry As-is with Debug (default). This is either the:

- default disconnect mode chosen by the debugger (see Setting disconnect mode on page 2-23 for details)
- user-defined disconnect mode specified in the target configuration file (see User-defined disconnect mode on page 2-24 for details).

Any user-defined disconnect mode takes precedence over the default disconnect mode chosen by the debugger.

If you click Cancel, RealView Debugger disconnects using the disconnect mode shown as the default. Where this mode is not supported by your target, a different disconnect mode is used and you are warned about the substitution.

Note
This selection box is also displayed if you specify a prompt as your user-defined disconnect mode. In this case, there is no default entry (see Using a prompt on page 2-28 for details).
Using a prompt

You can specify a user-defined disconnect mode to display a prompt to enable you to choose the disconnection mode for each connection. In this case, RealView Debugger displays the Disconnect Mode selection box shown in Figure 2-11 on page 2-26. However, where the prompt setting is defined, the state options offered are all supported by the target vehicle and so do not include a default option.

Note

If a prompt is specified in your board file, or in any .bcd file linked to the connection, it takes priority over any other user-defined disconnect mode setting. This prompt-first rule holds true regardless of where the setting is in the configuration hierarchy.

For more information about using a prompt, see:

- the example Specifying connect and disconnect mode on page 3-32 in Chapter 3 Configuring Custom Targets
- the description of the Advanced_Info block in Appendix A Configuration Settings Reference.

2.5.5 Disconnecting by exiting

By default, exiting the debugger with a connection causes details of the connection to be stored in the current workspace. See the chapter describing configuring workspaces in RealView Debugger v1.7 User Guide for more details.

When you exit, the debugger prompts you to ensure you want to disconnect, as described in Disconnection confirmation.

When RealView Debugger starts up with a workspace that includes stored connection information, it tries to reconnect. If this fails, you are prompted for the next action, as described in Reconnecting stored connections on page 2-29.

Disconnection confirmation

If you exit the debugger with active connections, the debugger asks whether these connections can be disconnected, shown in Figure 2-12 on page 2-29.
If you do not disconnect, the Target Vehicle Server (TVS) maintains the connection until another debugger session requires it. Therefore, if you load and run an image on your target, stop it, exit from the debugger without disconnecting, and then rerun the debugger, it will still be stopped in the same place when the debugger redispays the connection.

If you disconnect when exiting the debugger, TVS disconnects that connection, using the disconnection mode defined in the Advanced Information setting Disconnect_mode in the connection properties for that connection. If this leaves the TVS with no connections, it exits as well.

**Reconnecting stored connections**

If, when you restart the debugger, the connection stored in the workspace is no longer available, you are prompted to retry or reconfigure it, shown in Figure 2-13. Select **Configure Device Information...** from the list and click **OK** to reconfigure the connection (see Chapter 4 Configuring Custom Connections for more details). Click **Cancel** to abort the connection attempt.

---

**Figure 2-12 Disconnect confirmation**

If you disconnect when exiting the debugger, TVS disconnects that connection, using the disconnection mode defined in the Advanced Information setting Disconnect_mode in the connection properties for that connection. If this leaves the TVS with no connections, it exits as well.

**Figure 2-13 Disconnect reconfiguration or retry**
2.5.6 Using the CLI

You can disconnect from a debug target using the CLI command DISCONNECT. This also enables you to specify the disconnection mode using a dialog box. See RealView Debugger v1.7 Command Line Reference Guide for more information.
Chapter 3
Configuring Custom Targets

This chapter describes in detail the debug target configuration model used by RealView® Debugger. Read this chapter to find out how to describe your debug target to the debugger. It contains the following sections:

- About target configuration on page 3-2
- The supplied .bcd files on page 3-7
- Creating new target descriptions on page 3-9
- Example descriptions on page 3-24.
3.1 About target configuration

This section describes target configuration in more detail in readiness for the examples in the rest of this chapter. This section describes:

- Target configuration settings
- Default configuration files
- How configuration files are linked together
- Board file contents

3.1.1 Target configuration settings

RealView Debugger assembles configuration settings to describe the debug environment and all the debug targets available in the current debugging session. These settings serve two main purposes. They:

- describe your debug targets in a way that enables RealView Debugger to find out all the information it requires to establish a connection
- enable you to configure the Extended Target Visibility (ETV) features of your debug targets, and make this information accessible to RealView Debugger.

Using internal configuration settings in this way means that you can change your debug target connection, or connect to multiple debug targets, without leaving your RealView Debugger session.

3.1.2 Default configuration files

As introduced in Chapter 1 *Introduction*, the debug target configuration settings are maintained through the use of a hierarchy of configuration files:

- Board file
- RV-msg configuration files
- JTAG configuration files
- RDI configuration files
- Board/Chip definition files

Note

Some default configuration files are supplied as part of the RealView Debugger installation, see *Configuration files* on page 1-11 for details.
Board file

RealView Debugger uses a board file to access information about the debugging environment and the debug targets that are available. You can use RealView Debugger with the default board file that is installed for you. This is called rvdebug.brd and is copied into your home directory, from the default settings directory \etc, when you first use RealView Debugger after installation. This means that if you damage your personal board file, you only have to delete it from your home directory and a new copy of the original default board file is placed there when you next run the debugger.

The board file defines the debug target configuration settings for the current session. For each available target, it describes the type of target, the simulator or emulator being used, and any custom connection information.

RealView Debugger must have a board file to make connections. If you work with a variety of targets and connections, you might set up, and save, several board files so that you can easily switch the debugger from one to another. You can use the default board file as a basis for any number of further copies, each edited for a particular purpose.

You can use a text editor to display or print the contents of a board file, and all associated configuration files, but it is recommended that you never edit these files with a text editor or word processor. Use only the Connection Properties window to make changes to a board file, or to create a new one.

RV-msg configuration files

These files are used to define the configuration settings for RV-msg connections, such as RealView ICE. You change the contents of one of these files when you modify the configuration of an RV-msg connection using the RVConfig dialog box.

Each interface unit is defined using a .rvc file, for example rvi_940T_tst.rvc. However, you can specify different .rvc files to configure custom targets.

When you install RealView ICE, these XML files are created in your RVI\RVConfig directory ready for use. RealView Debugger searches for these files using the default search path, see The RealView Debugger search path on page 1-14 for details. However, you can specify a full pathname in your board file to use a different location, for example \RVI\test_targets\rvi_920T_tst.rvc.

Note

Do not edit these files manually. See the chapter describing configuring a RealView ICE connection in RealView ICE User Guide for full details on how to use the RVConfig dialog box.

---
JTAG configuration files

JTAG files are used for built-in emulators such as ARM® Multi-ICE® direct connect. These files define the JTAG (Joint Test Action Group) boundary scan architecture for your target and so describe the number and types of hardware devices in the scan chain that are available for connection.

Note
When installing RealView Debugger, you must choose the Custom option and install support for DSP. Do this to ensure that the required JTAG files are available to enable connection using Multi-ICE direct connect.

JTAG files provide access, on the local workstation, to an emulator for each architecture that RealView Debugger supports, as specified in the installation. Each emulator is defined using a .jtg file named processor.jtg, for example arm.jtg. These files are created in the default settings directory \etc at installation.

Whenever RealView Debugger reads a .brd file, it also searches for any of these related files and reads them. In this way, the information held in the JTAG files becomes part of the configuration settings for this session. You can add or remove JTAG files if necessary, without having to edit the .brd file. By default, new .jtg files are stored in \etc, but you can specify a different location in your .brd file.

If you are using an emulation scan chain that corresponds to the devices defined in an available .jtg file, you can refer to that file and specify the I/O port used by the emulator, if necessary. If you plan to use different debug target systems, you must create a .jtg file that defines the devices on your target. Do this using the Connection Properties window.

Note
When working with RDI targets, JTAG files are replaced by .cnf configuration files.

RDI configuration files

When you are working with RDI targets, such as Multi-ICE and Remote_A, special configuration files are generated by the RDI configuration utilities. These files make up the RealView Debugger configuration settings specific to RDI targets.

The RDI configuration files consist of:

.rbe There is one .rbe file for each RDI target available for connection.
These files store the target configuration settings you make in one debugging session so that they can be automatically used again in any subsequent sessions.

--- Note ---

You must not edit these files manually. Instead, use the RDI configuration utilities provided as part of the RealView Debugger base product, as described in Chapter 4 Configuring Custom Connections.

---

**Board/Chip definition files**

Board/Chip definition files contain ETV information about a particular board or chip as supplied by the manufacturer, including peripheral registers and memory regions. Board/Chip definition files are also supplied as part of support plugins to enable awareness in RealView Debugger, for example RTOS. The files are usually stored in one location so that they can be referred to from as many places as necessary, but only a single copy requires maintenance.

Each board or chip is defined using a file named `processor_name.bcd`, for example `CM920T_ETM.bcd` or `CM966E5.bcd`. By default, `.bcd` files are stored in \etc, but you can specify a different location in your board file.

In general, you do not have to edit these files. However, where changes are required, use the Connection Properties window to make the necessary changes.

---

**3.1.3 How configuration files are linked together**

The board file might reference several other configuration files, for example the RDI definitions, `*.rbe`, and the Board/Chip definition files, `*.bcd`, to form the complete configuration. This relationship is shown in Figure 3-1 on page 3-6.
The RV-msg, JTAG, and RDI configuration files contain the remaining information required to configure a specific debug target. These files are not structured in the same way as the board files. They use the format required by the debug target that they are used to configure, for example, the RDI configurations are structured as Toolconf files.

### 3.1.4 Board file contents

Appendix A *Configuration Settings Reference* provides a detailed reference that describes all the configuration settings available in the board file. It provides examples of when to use settings and describes how to make changes. This might be useful when you are working through the examples in this chapter.
3.2 The supplied .bcd files

This section details the target board descriptions that are supplied as part of the RealView Debugger base product. These descriptions can be referenced from a specified target without further modification. You do this by linking the file to the connection, see Linking a board, chip, or component to a connection on page 3-14 for more details.

The target descriptions are stored in files with the extension .bcd, in the default settings directory \etc. These Board/Chip definition files include details of the location and format of the registers and memory available on the described target boards.

Note

If you upgrade to a later version of RealView Debugger you are provided with a new, and possibly different, version of these files. It is recommended, therefore, that you do not modify these files so that you can upgrade easily. See Creating new target descriptions on page 3-9 for details of creating your own configurations.

The supplied descriptions include:

- AP.bcd  
  A description of the ARM Integrator/AP registers and that part of the core module memory map that is decoded by the motherboard. This description is also suitable for use with Integrator core modules.

- Eval7T.bcd  
  A description of the ARM Evaluator-7T registers and memory map, including a description of the KS32C50100 processor internal registers.

- CM7TDML.bcd  
  A description of the ARM CM7TDML processor core module registers and memory map.

- CM740T.bcd  
  A description of the ARM CM740T processor core module registers and memory map.

- CM920T.bcd  
  A description of the ARM CM920T processor core module registers and memory map.

- CM920T_ETM.bcd  
  A description of the ARM CM920T-ETM processor core module registers and memory map.

- CM940T.bcd  
  A description of the ARM CM940T processor core module registers and memory map.
Configuring Custom Targets

CM10200.bcd A description of the ARM CM10200 processor core module registers and memory map.

CP.bcd A description of the ARM Integrator/CP registers and that part of the core module memory map that is decoded by the motherboard. This description is also suitable for use with Integrator core modules.

Use the supplied target descriptions by referencing them from the connection you use to communicate with your target. For example, if you are using an ARM CM920T processor core module with RealView ICE, modify the setting BoardChip_name in the CONNECTION=RealView ICE group to reference the CM920T.bcd file.

If you are using an Integrator/AP, Integrator/CM, or Integrator/CP motherboard with a core module, you can combine the platform and processor core module descriptions by using multiple BoardChip_name settings in the CONNECTION group, shown in the example in Figure 3-2.

![Figure 3-2 Referencing two .bcd files in the Connection Properties window](image)

For further details on doing this, see Linking a board, chip, or component to a connection on page 3-14.
3.3 Creating new target descriptions

This section explains how to create variations of existing configurations and new target descriptions. Creating new target descriptions, rather than modifying existing entries, provides certain advantages, for example:

- if you define the addresses of I/O registers, and the bit fields within them, the debugger can display tabs in the Register pane enabling GUI access to these values
- if you define a memory map, the debugger can check that memory is used as it should be, including refusing to load programs where there is no memory, and automatically invoking Flash memory programming routines.

If you want to create and use a new target description, you must:

1. Create a new .bcd file to store the configuration. The recommended way to do this is to base the new file on an existing file stored in the default settings directory, see Creating a new .bcd file on page 3-10 for details.

2. Create and name a BOARD, CHIP, or COMPONENT group for the configuration. Which group you use depends on the type of hardware you are describing.

3. Define the CONNECTION entry to specify the access method used to connect to the new target.

4. Link the new target definition to the CONNECTION group that the target uses.

The rest of this section describes how to configure the new target, that is:

- Creating a new .bcd file on page 3-10.
- Creating and naming a board, chip, or component on page 3-12.
- Linking a board, chip, or component to a connection on page 3-14.

For information about how these different configuration groups interact, that is configuration settings, the board file, and any linked Board/Chip definition files, see Managing configuration settings on page 3-22.

For examples that describe how to create new CONNECTION entries (as required by step 3 above), see Example descriptions on page 3-24.

3.3.1 Using the examples

The examples in the rest of this chapter, modify the board file stored in your RealView Debugger home directory. By default, this is called rvdebug.brd. Target configuration files might also be stored in this directory, for example .rvc files or .cnf files.
It is recommended that you back up this directory before starting the examples, so that you can restore your original configuration later. For details see:

- *Configuration files* on page 1-11 for instructions on making backups of your configuration.
- *Restoring your .brd file* on page 3-57 for instructions on restoring a default configuration.
- *Troubleshooting* on page 3-57 for instructions on recovering from an incorrectly configured debugger home directory, whether or not you have a backup.

**Note**

Remember, when you are following these examples, do not configure the board file when the debugger is connected to a target.

### 3.3.2 Creating a new .bcd file

To create a new *.bcd file, copy one of the existing files within RealView Debugger or using the tools provided by your operating system, for example, using Windows Explorer or the `cp` command on Sun Solaris.

To copy a *.bcd file from RealView Debugger use the Connection Properties window:

1. Select **File → Connection → Connection Properties...** to display the Connection Properties window.
2. Expand the group (*.bcd) Board/Chip Definitions by clicking on the plus sign. This displays the current list of target descriptions, in the left pane.
3. Right-click on the name of the *.bcd file to copy. For example, right-click on the entry `...\AP.bcd`.
4. Select **Save As...** from the context menu, to display the dialog box shown in Figure 3-3 on page 3-11.
By default, *.bcd files are saved in your default settings directory \etc.
In this example, save the new file in your RealView Debugger home directory.

5. Enter the new filename, for example AM.bcd. You must use the .bcd file extension when saving the file in your home directory.

6. Click Save.
   The dialog box closes and the new name is displayed in the *.bcd list. It replaces the initial filename.

7. Select File → Save Changes, then File → Reset to display the updated list of target description files.

8. Expand the group (*.bcd) Board/Chip Definitions to show the current list of target descriptions. The list includes the new .bcd file, that is AM.bcd.

9. Delete the contents of the new .bcd file group to avoid problems caused by old or inappropriate settings:
   a. Expand the new .bcd file group. Right-click on the contents, for example, *BOARD=AP, to display the context menu shown in Figure 3-4 on page 3-12.
3.3.3 Creating and naming a board, chip, or component

To configure your target, within the *.bcd file that you created in Creating a new .bcd file on page 3-10, you must create a BOARD, a CHIP, or a COMPONENT group. RealView Debugger uses these groups in the same way regardless of which type you create. However, it is recommended that you use them as follows so that it is clear what the group describes:

**BOARD**

Use this to define target boards as a whole, for example, the Evaluator-7T, or the Integrator/AP motherboard, implementing memory maps and including simple peripheral components.
CHIP

Use this to define significant devices on a target board. For example, in the supplied target descriptions, the Evaluator-7T (\Eval7T.bcd) references the ARM-based KS32C50100 (CHIP=KS32C50100) to define the processor and ASIC components of that device.

It is recommended that you use a CHIP entry where you might use the device on more than one board, or where the device is in itself complex.

COMPONENT

Other components not covered by the above.

To create a group in a .bcd file:

1. Right-click on the name of the *.bcd file. For example, right-click on the entry ...\AM.bcd, shown in Figure 3-5.

![Figure 3-5 Adding a new group to a .bcd file](image)

2. Select Make New Group... to display the Group Type/Name selector dialog box.

3. Select the type of group you want to use from BOARD, CHIP, or COMPONENT.

4. In the Group Name data field change the name from new to something suitable for your target, using only alphanumeric characters, underscore _, and dash -. This example shows a CHIP called S5471KT.

5. Click OK to create the group. This collapses the AM.bcd entry so that the new group is hidden.

6. Expand the .bcd file group to see the new CHIP=S5471KT group, shown in Figure 3-6 on page 3-14.
7. Select **File → Save and Close** to save your changes and close the Connection Properties window.

Use the method described here to set up as many configuration groups as you require. It is recommended that you do this before creating, or modifying, any CONNECTION groups, as described in *Example descriptions* on page 3-24.

### 3.3.4 Linking a board, chip, or component to a connection

The configuration group you create in a *.bcd* file is only used if:

- the *.bcd* file is referenced from a connection
- the identification string, `id_match`, specified in the `Advanced_Information` group, matches the hardware ID string returned by the target.

If more than one linked configuration group matches the target hardware, the group with the longest match is used.

This section describes how you make these links in your board file. These examples assume that you are linking configuration groups to an existing connection, in this case using RealView ICE.

If this is the first time you have used this vehicle, ensure that you configure the debug interface before trying to connect (see *Configuring a RealView ICE interface unit* on page 4-11 for details).
There are several cases to consider, presented here in order of increasing complexity:

- Linking one board group to one connection
- Linking several board groups to one connection on page 3-17
- Linking one or more board groups to another board group on page 3-18
- Linking one or more board groups to multiple processor connections on page 3-21.

**Linking one board group to one connection**

This configuration is shown in Figure 3-7.

![Figure 3-7 Linking one connection to one board](image)

To set up this linking configuration you must first ensure that the *.bcd file exists and contains the required BOARD, CHIP, or COMPONENT groups. Then reference this file from your board file:

1. In the Connection Properties window, expand the connection that you are using, for example CONNECTION=RealView ICE. Double-click on the entry in the List of Entries pane, the left pane, to see the contents in the Settings Values pane, the right pane.

2. Right-click on the BoardChip_name, in the right pane, to display the context menu shown in Figure 3-8 on page 3-16.
This context menu does not show the names of the .bcd files found by RealView Debugger. Rather, the menu includes entries derived from the contents of the .bcd files. It shows all BOARD, CHIP and COMPONENT groups in the order they were located. This illustrates how you can share target descriptions between different connections.

In this example, the list:

- does not include the option AM because the AM.bcd entry, created in Creating a new .bcd file on page 3-10, does not specify BOARD=AM
- includes the option S5471KT because the AM.bcd entry, created in Creating and naming a board, chip, or component on page 3-12, specifies CHIP=S5471KT.

3. Select the name of the required group, for example S5471KT. A new entry is displayed in the right pane with an asterisk * beside it, for example *BoardChip_name S5471KT.
4. Select **File → Save and Close** to save your changes and close the Connection Properties window.

When you connect to your RealView ICE target, the configuration defined in your board group is applied to the connection. This gives you visibility of your custom hardware.

**Linking several board groups to one connection**

You might want to link several groups to a single connection if the groups represent different, possibly optional, parts of the same target. For example, the Integrator/AP motherboard definition `BOARD=AP` and an Integrator core module definition such as `BOARD=CM940T`. This kind of layout is shown in tree form in Figure 3-9.

![Figure 3-9 Linking one connection to two boards](image)

When you reference multiple boards, RealView Debugger merges the settings from each matching group in a breadth-first search of the group tree. Therefore the complete configuration is the combined configurations of all of the matching groups. If the same setting is specified in more than one group, the specification in the group that is listed first in the `CONNECTION` is used, for example `BoardChip_name=AP` in Figure 3-9.

To set up this linking configuration you must first ensure that the `.bcd` files exist and contain the required `BOARD`, `CHIP`, or `COMPONENT` groups. Then reference these files from your board file:

1. In the Connection Properties window, expand the connection that you are using, for example `CONNECTION=RealView ICE`.

2. Right-click on the `BoardChip_name`, in the right pane, to display the context menu shown in Figure 3-8 on page 3-16.

   In this example, the list includes the `CHIP` called `S5471KT` that was created in *Creating and naming a board, chip, or component* on page 3-12.

3. Select the name of the required group, for example `CM940T`. A new entry is displayed in the right pane with an asterisk * beside it, for example `*BoardChip_name CM940T`. 
4. Right-click on the BoardChip_name that does not have an asterisk to see the context menu, shown in Figure 3-10.

5. Select the name of the required group, for example AP. A new entry is displayed in the right pane with an asterisk * beside it, for example *BoardChip_name AP.

6. Select File → Save and Close to save your changes and close the Connection Properties window.

When you connect to your RealView ICE target, the configuration defined in your board groups is applied to the connection. This gives you visibility of all your custom hardware.

Use the method described here to set up as many configuration groups as you require.

**Changing the order of board groups**

This procedure describes adding board CM940T before board AP, so that the structure shown in Figure 3-9 on page 3-17 is recreated. New boards are always added at the top of the list and this gives their settings priority over the settings in boards lower down the list.

If you want to reorder the boards in the BoardChip_name list to give the settings a different priority, right-click on any BoardChip_name entry to display the context menu and select Manage List..., shown in Figure 3-10. Use the Settings: List Manager dialog box to reorder the board groups.

**Linking one or more board groups to another board group**

You might want to link several groups together so that you can share descriptions or simplify each part of a description. For example, the description of the ARM Evaluator-7T provided in Eval7T.bcd is split into a description of the board, Evaluator7T, and a description of the processor on the board, KS32C50100. This is shown in tree form in Figure 3-11 on page 3-19.
Groups can contain `BoardChip_name` references to other groups, so that you can build multi-layered descriptions. For example, if you were building a simple Ethernet router, you might use the network interface on the KS32C50100 with a second network interface provided by an AMD LANCE. If you set this up as a board file, you might get the structure shown in Figure 3-12.

**Figure 3-11 Linking one board into another board**

**Figure 3-12 Linking one board into other boards**

--- **Note** ---

You are not required to split your board up into distinct `CHIP` descriptions. You could create one `BOARD` description containing all of the required information. However, splitting your board up into distinct `CHIP` descriptions enables you share descriptions or reuse a description for another development project.
To create the structure shown in Figure 3-12 on page 3-19 you must first ensure that the `.bcd` files exist:

1. Create a new `.bcd` file, called `ether.bcd` as described in Creating a new `.bcd file` on page 3-10.
2. Create a new `BOARD` group in `ether.bcd` file, called `EtherRouter`, as described in Creating and naming a board, chip, or component on page 3-12.
3. Create a new `.bcd` file, called `AMDLANCE.bcd` as described in Creating a new `.bcd file` on page 3-10.
4. Create a new `CHIP` group in the `AMDLANCE.bcd` file, called `AMDLANCE`, as described in Creating and naming a board, chip, or component on page 3-12.

Splitting out the target descriptions in this way makes it easier to reuse configuration details on different connections. Remember to save your changes and reset the board file contents.

To reference these files from your board file:

1. In the Connection Properties window, expand the connection that you are using, for example `CONNECTION=RealView ICE`.
2. Right-click on the `BoardChip_name` entry, in the right pane, to display the context menu shown in Figure 3-8 on page 3-16.
   In this example, the list includes the CHIP called `S5471KT`, the BOARD called `EtherRouter`, and the CHIP called `AMDLANCE`.
3. Select the name of the required group, for example `EtherRouter`. A new entry is displayed in the right pane with an asterisk * beside it, shown in Figure 3-13.

4. In the Connection Properties window expand the group (`*.bcd`) Board/Chip Definitions to show the current list of target descriptions. The list includes the new `.bcd` files, that is `ether.bcd` and `AMDLANCE.bcd`.
5. Expand the `ether.bcd` entry to see the `BOARD=EtherRouter` entry. Expand this group to see the settings in the right pane, including `BoardChip_name`.
6. Right-click on the `BoardChip_name` entry to see the context menu, shown in Figure 3-8 on page 3-16.
7. Select the name of a board group, for example KS32C50100. A new entry is added to the right pane with an asterisk * beside it. For example, BoardChip_name KS32C50100.

8. Right-click on the BoardChip_name that does not have an asterisk again to see the context menu.

9. Select the name of the other board group, for example AMDLANCE. A new entry is added to the right pane with an asterisk * beside it, shown in Figure 3-14.

10. Select File → Save and Close to save your changes and close the Connection Properties window.

When you connect to your RealView ICE target, the configuration defined in your board groups is applied to the connection. This gives you visibility of all your custom hardware.

Use the method described here to set up as many configuration groups as you require.

**Linking one or more board groups to multiple processor connections**

If you want to use RealView Debugger to debug a multiprocessor target, where some of the processor configurations are different, you do so by defining multiple Advanced_Information groups using names that match the processor name. These then appear in the Connection Control window.

For example, if you have a single Integrator CM920T, RealView ICE names the connection ARM920T_0. The _0 in the name indicates that this processor is on the first TAP position, that is position 0. If, in any BOARD, CHIP or COMPONENT, you create an Advanced_Information group called ARM920T_0, the entries in that group only apply to that processor.

If you have two CM920T boards connected to an Integrator motherboard, RealView ICE names them ARM920T_0 and ARM920T_1. If you create two Advanced_Information groups called ARM920T_0 and ARM920T_1, shown in Figure 3-15 on page 3-22, you can configure each board independently. Using the Default group, you can also have Advanced_Information that applies to both processors linked to the connection.
Figure 3-15 Configuring a two processor target

See Chapter 4 Configuring Custom Connections for more information about managing connections.

For more information about connecting RealView Debugger to multiprocessor targets, see the multiprocessing chapter in RealView Debugger v1.7 Extensions User Guide.

3.3.5 Managing configuration settings

You configure your debug target by amending board file entries using the Connection Properties window. This enables you to specify connection behavior, target visibility, image loading parameters, project loading and binding behavior, and disconnect options.

RealView Debugger provides great flexibility in how to configure these settings so that you can control your debug target and any custom hardware that you are using. This means that some settings can be defined in the top-level board file so that they apply to a class of connections, for example CONNECTION=RealView ICE, or on a per-board (or per-chip) basis using groups in one or more linked Board/Chip definition files, for example CHIP=S5471KT in the file AM.bcd.

Where settings conflict, priority depends on the type of setting, whether it has changed from the default, and its location in the configuration hierarchy. When you reference multiple boards, RealView Debugger merges the settings to generate a combined configuration from all the matching groups. If the same setting is specified in more than one group, the specification in the group that is listed first in the CONNECTION is used (see Linking several board groups to one connection on page 3-17).
To ensure that settings defined in one or more linked .bcd file are used to assemble the target configuration, do not change the default settings contained in the target connection group. For example, if you specify top of memory in a linked .bcd file, you must check that the same entry is blank (the default) in the top-level board file:

1. Select File → Connection → Connection Properties... to display the Connection Properties window.

2. Expand the following entries in turn:
   a. CONNECTION=MP3Player (change as required)
   b. Advanced_Information
   c. Default
   d. ARM_config

3. Ensure that Top_memory is blank.
   If the setting contains an entry, right-click to display the context menu. Because the setting has been configured, the menu now offers more options. Select Reset to Empty to create a blank setting.

The interaction between settings in the top-level board file and any linked .bcd files might be important if you are using .bcd files to enable RTOS awareness in RealView Debugger.

——— Note ————
The configuration settings Connect_mode and Disconnect_mode are a special case. See Specifying connect and disconnect mode on page 3-32 for details on using these entries.
3.4 Example descriptions

These examples describe how to amend board file entries, using the Connection Properties window, to configure your debug target. They assume you are familiar with the procedures described in the online help topic Changing Settings.

In these examples, it is assumed that you are starting with the default board file as installed with the base product. Depending on the type of installation you choose, and your other ARM products, the contents of this file might be different from the one shown here. However, the methods described can be applied to your board file.

In these examples, board file entries are created and renamed. The names used are for illustration only and you can change them as you require. However, it is recommended that you avoid duplicates.

--- Note ---

- Do not configure the board file when the debugger is connected to a target.
- See Configuration files on page 1-11 for instructions on making backups of your configuration before you start.

The examples are described in the following sections:

- Setting up an Integrator board and core module
- Specifying connect and disconnect mode on page 3-32
- Configuring a memory map on page 3-35
- Setting up a custom register on page 3-38
- Setting up memory blocks on page 3-42
- Setting top of memory and stack heap values on page 3-45
- Using RealMonitor on page 3-48
- Flash programming on page 3-53.

This section also includes:

- Restoring your .brd file on page 3-57 for instructions on restoring your factory settings
- Troubleshooting on page 3-57 for instructions on recovering from an incorrectly configured debugger home directory, whether or not you have a backup.

3.4.1 Setting up an Integrator board and core module

This example demonstrates how to use the Connection Properties window to create a specific Integrator/AP and core module target configuration. It shows how to use a predefined Board/Chip definition file, with extension .bcd, to set up your target.
After you set up your target, the example also demonstrates how you can connect to it using RealView ICE with the Connection Control window, and verify that RealView Debugger can connect to the target.

The example is split into the following sections, which must be executed in this sequence:

1. Setting up the hardware and debug interface
2. Configuring the new target on page 3-26
3. Connecting to the new target on page 3-30
4. Viewing the new target definition on page 3-30.

**Setting up the hardware and debug interface**

The first stage is to set up the hardware and configure the RealView ICE unit:

1. Ensure that your Integrator/AP and core module are connected and switched on. This example uses the ARM940T processor, but you can use any core module supported by the Integrator/AP.

2. Ensure that you have RealView ICE installed, and that the RealView ICE unit is connected and configured for use with RealView Debugger. If you have not configured the RealView ICE unit, do so now.

--- Note ---

- RealView ICE version 1.0.1 works with RealView Debugger with some limitations. The examples assumes that you are using RealView ICE version 1.1. See the release notes accompanying the software for the latest details.
- See *RealView ICE User Guide* for more details on configuring RealView ICE.

---

**Using Multi-ICE**

If you are using Multi-ICE to complete the examples, ensure that you have Multi-ICE installed, and that the Multi-ICE server is running on the workstation connected to the target. If you have not yet configured the target with Multi-ICE, do so now.

See *Multi-ICE User Guide* for more details on configuring Multi-ICE.
Configuring the new target

The next stage is to configure the new target:

1. Start RealView Debugger without connecting to a target.
2. Select File → Connection → Connect to Target... to display the Connection Control window.
3. Right-click on the RealView ICE entry and select Connection Properties... from the context menu to display the Connection Properties window.
4. Right-click on the CONNECTION=RealView ICE entry and select Make Copy... from the context menu shown in Figure 3-16.

5. Use the Group Type/Name selector dialog box, shown in Figure 3-17 on page 3-27, to name the new connection, for example MP3Player.
Configuring Custom Targets

6. Click **OK**. The new entry, CONNECTION=MP3Player, is added to the List of Entries in the Connection Properties window.

7. Right-click on the Configuration setting for the new connection to specify the rvi.rvc configuration file, as shown in Figure 3-18.

8. Select **File → Save and Close** to save your changes. The new target is added to the Connection Control window.

9. Right-click on the new MP3Player connection and select **Connection Properties...** from the context menu. This displays the Connection Properties window showing the contents of the new connection, shown in Figure 3-19 on page 3-28.
10. Right-click on Description in the right pane to display the context menu. Select Edit Value and enter a short description for the new connection, for example Integrator/AP with ARM940 for MP3 product.

11. Select File → Save and Close to save your changes. The new target description is added to the Connection Control window.

**Linking board groups to the connection**

The next stage is to link board groups to the new connection. This is not necessary but gives extended target visibility and enables you to view register contents and manipulate memory.

**Note**

The connection created in this example is used in other examples in the rest of this chapter. If you do not link the board groups, the contents of the Register pane will differ from those shown here.

To link board groups:

1. Right-click on the MP3Player entry and select Connection Properties... from the context menu to display the Connection Properties window.

2. Right-click on BoardChip_name in the right pane to display the context menu.

3. Select AP to select the Integrator/AP description, shown in Figure 3-20 on page 3-29. A new entry +BoardChip_name AP is added to the pane.
Figure 3-20 Configuring the new connection

4. Right-click on BoardChip_name (not on +BoardChip_name). The context menu is displayed again.

5. Click on CM940T to select the ARM940T core module description. The Connection Properties window now shows two BoardChip_name settings, shown in Figure 3-21.

Figure 3-21 Board groups linked to the new connection
Target configuration settings were copied from the source connection. However, depending on the target hardware, you might have to configure other settings, for example to enable semihosting or to set stack size. See Configuring a memory map on page 3-35 for details. If required, you can do this now for the new connection.

6. Select File → Save and Close to save your changes and close the Connection Properties window.

Connecting to the new target

The next stage is to connect to the new target board and core module:

1. Select File → Connection → Connect to Target... to display the Connection Control window.

2. Click on the icon next to the new entry, MP3Player, to expand it.

   The entry expands and the relevant processor name entry is displayed, shown in Figure 3-22.

3. Click on the processor entry under the new MP3Player entry to connect to the target. RealView Debugger retrieves information specific to the target.

Viewing the new target definition

To view details about the new target hardware:

1. In the Code window, select View → Pane Views → Registers to display the Register pane. Two new tabs are included at the bottom of the pane, AP and CM940T.
2. Click on the **AP** tab. RealView Debugger shows the abstraction of the hardware information specific to the Integrator/AP board, shown in Figure 3-23.

![Figure 3-23 AP tab in the Register pane](image)

This tab view enables you to modify your Integrator/AP board features, such as the memory mapped peripherals.

3. To illustrate how RealView Debugger communicates directly with your Integrator/AP board, right-click on the text **OFF** directly beneath the **L2** entry in the Register pane, and select **ON** from the context menu. The relevant LED display on your Integrator/AP board is turned on.

4. Select the **CM940T** tab to see the abstraction of the hardware specific to the core module. The **PRESENT** status of the Motherboard indicates that the core module is connected to the Integrator/AP board.

   For more details on the Register pane, see the section on working with registers in the chapter describing debug views in *RealView Debugger v1.7 User Guide*.

5. In the Output pane at the bottom of the Code window, click on the **Cmd** tab. The display includes the line `Advanced_info searched...BOARD=AP, BOARD=CM940T` indicating that RealView Debugger is using the Integrator/AP board file. As a result, the memory map now contains the definitions required to use the Flash memory on the Integrator. See *Flash programming* on page 3-53 for details.
3.4.2 Specifying connect and disconnect mode

If you want to specify how RealView Debugger connects to (or disconnects from) a target processor, you must configure this in your board file. These definitions are contained in the Advanced_Information group for the target processor.

--- Note ---

This example uses the connection created in Setting up an Integrator board and core module on page 3-24. In this example, the S5471KT board group (stored in AM.bcd) is also linked to demonstrate how to manage conflicting settings (see Managing configuration settings on page 3-22).

The configuration settings Connect_mode and Disconnect_mode are a special case when used to configure a debug target:

- If a prompt is specified in your board file, or in any .bcd file linked to the connection, it takes priority over any other user-defined setting. This prompt-first rule holds true regardless of where the setting is in the configuration hierarchy.

- If a (non-prompt) user-defined setting is specified in your board file and in any .bcd file linked to the connection, the board file setting takes priority.

- A blank entry in the top-level Advanced_Information block ensures that any setting in a linked Board/Chip definition file is used instead. This might be important if you are using .bcd files to enable RTOS awareness in RealView Debugger.

The tasks described in this example illustrate both rules.

--- Note ---

The connect (or disconnect) mode that is actually used depends on the target hardware, the target vehicle, and the associated interface software that manages the connection. If you are using RealView ICE, the unit configuration determines the connect mode and makes the connection. Therefore, the unit configuration might override any settings that you specify in your board file. See Setting connect mode on page 2-13 for more details.

To configure connect mode for any hardware debug target:

1. Ensure that RealView Debugger is not connected to a target.

2. Select File → Connection → Connect to Target... to display the Connection Control window.

3. Right-click on the MP3Player entry and select Connection Properties... from the context menu.
4. Expand the following entries in turn:
   a. **CONNECTION=MP3Player (change as required)**
   b. Advanced Information
   c. Default

5. Right-click on the **Connect_mode** entry, to display the context menu, shown in Figure 3-24.

![Figure 3-24 Viewing the Connect_mode options](image)

The connection options are fixed, shown in Figure 3-24, and so might include options that are not supported by your target vehicle. If you specify such an option, the debugger prompts you to select an appropriate connection mode when you try to connect. See *Setting connect mode* on page 2-13 for details on how RealView Debugger connects to a target.

6. Select **no_reset_and_no_stop** so that the target processor is not reset when you connect. The running state of the target is unchanged.

7. Select **File → Save and Close** to save your changes and close the Connection Properties window.

To configure disconnect mode for any hardware debug target:

1. Ensure that RealView Debugger is not connected to a target.

2. Select **File → Connection → Connect to Target...** to display the Connection Control window.

3. Right-click on the **MP3Player** entry and select **Connection Properties...** from the context menu.
4. Expand the following entries in turn:
   a. (*.bcd) Board/Chip Definitions
   b. (...\AM.bcd
   c. CHIP=S5471KT
   d. Advanced_Information
   e. Default

5. Right-click on the Disconnect_mode entry, to display the context menu, shown in Figure 3-25.

![Figure 3-25 Viewing the Disconnect_mode options](image)

The disconnection options are fixed, shown in Figure 3-25, and so might include options that are not supported by your target vehicle. If you specify such an option, the debugger prompts you to select an appropriate disconnection mode when you try to disconnect. See Setting disconnect mode on page 2-23 for details on how RealView Debugger disconnects from a target.

6. For this example, select prompt so that RealView Debugger displays a prompt when you disconnect. This enables you to choose what disconnection mode to use. See Using a prompt on page 2-28 for details on using a disconnect prompt in this way.

   **Note**

   This setting is stored in the Board/Chip definition file and so is used if you link this file to any other connection.
7. You can override the `Disconnect_mode` setting by specifying the same setting higher in the configuration hierarchy (see Managing configuration settings on page 3-22). To ensure that the setting in the CHIP=S5471KT group is used, check for a blank entry:
   a. Expand `CONNECTION=MP3Player` *(change as required)*
   b. Expand `Advanced_Information`
   c. Expand `Default`
   d. Ensure that `Disconnect_mode` is blank
   If the setting contains an entry, right-click to display the context menu shown in Figure 3-25 on page 3-34. Because the setting has been configured, the menu now offers more options. Select **Reset to Empty** to create a blank setting.

   **Note**
   If a prompt is specified in your board file, or in any `.bcd` file linked to the connection, it takes priority over any other user-defined disconnect mode setting. This prompt-first rule holds true regardless of where the setting is in the configuration hierarchy.

8. Select **File → Save and Close** to save your changes and close the Connection Properties window.

9. To test your changes:
   a. Connect to your target and load an image, for example dhrystone.axf.
   b. Run your image.
   c. Disconnect from your target to view the Disconnect Mode prompt.
   See Disconnect (Defining Mode)... on page 2-25 for details on using these options.

   **Note**
   This behavior also applies to connect mode, that is, a blank entry in the top-level `Advanced_Information` block ensures that any setting in a linked Board/Chip definition file is used instead. This might be important if you are using `.bcd` files to enable RTOS awareness in RealView Debugger.

### 3.4.3 Configuring a memory map

If you want to set up a memory map that is used automatically when you connect to a target processor, you must configure this in your board file. The memory definition is contained in the `Advanced_Information` group for the target processor.
Configuring Custom Targets

--- Note ---
This example uses the connection created in Setting up an Integrator board and core module on page 3-24. However, in this example, the AP, AM, and CM940T board groups are not linked to ensure a default memory mapping.

To configure a memory map for any hardware debug target:

1. Ensure that RealView Debugger is not connected to a target.
2. Select File → Connection → Connect to Target... to display the Connection Control window.
3. Right-click on the MP3Player entry and select Connection Properties... from the context menu.
4. Expand the following entries in turn:
   a. CONNECTION=MP3Player (change as required)
   b. Advanced Information
   c. Default
   d. Memory_block
5. Right-click on the Default entry, under Memory_block, to display the context menu, shown in Figure 3-26.

![Figure 3-26 Viewing the default Memory_block settings](image-url)
6. Select **Make Copy...** to describe the memory map for the chosen target. Give this entry a suitable name, for example SSRAM, and click **Create**.

7. Click on the new SSRAM entry in the left pane to display it in the right pane.

8. Set the value of Start, in the right pane, to 0x0.

9. Set the value of Length to 0x20000.

10. Set the value of Description to **Static RAM** as shown in Figure 3-27.

11. Select **File → Save and Close** to save your changes and close the Connection Properties window.

12. Connect to your target and load an image, for example dhrystone.axf.

13. Select **View → Pane Views → Memory Map** to view the new memory map, shown in Figure 3-28, before loading an image.

![Figure 3-27 Viewing the contents of the new group](image)

![Figure 3-28 New memory map in the Process Control pane](image)
3.4.4 Setting up a custom register

This example describes the steps to follow to specify a register MYREG, that appears as a new tab in the Register pane. It also describes how to set up named bit fields in this register. To set up the custom register, you must make changes in the Memory_block, Register, Register_enum, and Register_Window groups.

--- Note ---
This example uses the connection created in Setting up an Integrator board and core module on page 3-24. However, in this example, the AP, AM, and CM940T board groups are not linked to ensure a default registers view.

---

In this example:

- The memory region used for the new register is called REGS and is addressed from 0x10000000 - 0x107FFFFF.
- The custom register, named MYREG, has an offset of 0x20 from the base of the memory block.
- The custom register, MYREG, has four bit fields. These are used as indicators of the state of the register and are named INDICATORS. They are labeled IND1, IND2, IND3, and IND4.

The example is split into the following sections, which must be executed in this sequence:

1. Setting up the configuration
2. Creating enumerations for the register values on page 3-39
3. Creating the register descriptions on page 3-40
4. Creating the register tab on page 3-41
5. Displaying the register on page 3-42.

Setting up the configuration

In this stage, you set up a memory group that provides the base address for the new registers:

1. Ensure that RealView Debugger is not connected to a target.
2. Select File → Connection → Connect to Target... to display the Connection Control window.
3. Right-click on the MP3Player entry and select Connection Properties... from the context menu.
4. Expand the following entries in turn:
   a. CONNECTION=MP3Player (change as required)
   b. Advanced Information
   c. Default

5. Expand the Memory_block group.

6. Rename the Default entry under Memory_block to REGS.

7. Click on the REGS entry, in the left pane, to display the group contents.

8. Set the value of Start, in the right pane, to 0x10000000.

9. Set the value of Length to 0x800000.

10. Set the value of Description to I/O Registers, (shown in Figure 3-29).

![Figure 3-29 Configuring REGS](image)

**Creating enumerations for the register values**

In this stage you set up enumerations, or names for specific values, that are used when the register value is displayed:

1. Expand Register_enum in the left pane.

2. Right-click on Register_enum and select Make New... to create a new group. Name this E_SWITCH.

3. Click on the E_SWITCH entry, in the left pane, to display the group contents.

4. Set the value of Names, in the right pane, to On, Off.

5. Rename the Default entry under Register_enum to E_ENABLE.

6. Click on the E_ENABLE entry, in the left pane, to display the group contents.

7. Set the value of the Names entry, in the right pane, to Disable, Enable, shown in Figure 3-30 on page 3-40.
Creating the register descriptions

In this stage you create descriptions of each register:

1. Expand the Register group.
2. Rename the Default entry under Register to Newreg.
3. Expand the Newreg group.
4. Set the value of Base to REGS.
5. Set the value of Start to 0x20, that is the offset from the base address of the REGS memory block.
6. Expand the Bit_fields group, to set up the four bit fields.
7. Rename the Default entry under Bit_fields to IND1.
8. Use Make Copy... on IND1. The dialog suggests the name IND2. Click Create.
9. Use Make Copy... on IND2. The dialog suggests the name IND3. Click Create.
10. Use Make Copy... on IND3. The dialog suggests the name IND4. Click Create.
11. Click IND1, in the left pane and set these values, shown in Figure 3-31 on page 3-41:
    • Position=0 (this is the default)
    • Size=4
    • Enum=E_ENABLE.
12. Click on the IND2 entry and set these values:
   - Position=4
   - Size=4
   - Enum=E_SWITCH.

13. Click on the IND3 entry and set these values:
   - Position=8
   - Size=4.

14. Click on the IND4 entry and set these values:
   - Position=12
   - Size=4.

Creating the register tab

In this stage you create a Register_Window group to display the new register in the Register pane:

1. Expand the Register_Window group.
2. Rename the Default entry under Register_Window to MYREG. This is the name of the new tab in the Register pane.
3. Click on MYREG, in the left pane.
4. Set the Line entry, to _INDICATORS. Literals entered in Line must be preceded by an underscore.
5. Use Make New... on $Line to create a new $Line entry.
6. Set the new $Line to IND1,IND2,IND3,IND4.

The Connection Properties window looks like Figure 3-32 on page 3-42.
All board file entries are now complete.

7. Select File $\rightarrow$ Save and Close to save your changes and close the Connection Properties window.

### Displaying the register

In the last stage, display the new register in the Register pane:

1. Connect to your target.
2. Select View $\rightarrow$ Pane Views $\rightarrow$ Registers to view the new tab, MYREG, shown in Figure 3-33.

### 3.4.5 Setting up memory blocks

This example uses the connection created in Setting up an Integrator board and core module on page 3-24. However, the AP, AM, and CM940T board groups are not linked to ensure a default memory mapping.

This example assumes that you have worked through Setting up a custom register on page 3-38, because it uses some of the configuration details set up in that example and saved in the board file.

This example describes how to set up two memory blocks that are activated at different times according to the value of a register. It uses the Newreg register created in Setting up a custom register on page 3-38. This is displayed in the MYREG tab in the Register pane.
This example also describes setting a memory rule to specify how the memory is used. When Newreg is zero, MEM2 is activated. Otherwise, MEM1 is used. The example is split into these sections, which must be executed in this sequence:

1. *Defining the memory blocks*
2. *Defining the memory rules* on page 3-44.

**Defining the memory blocks**

The first stage is to define the two `Memory_block`s named MEM1 and MEM2:

1. Ensure that RealView Debugger is not connected to a target.
2. Select **File** → **Connection** → **Connect to Target...** to display the Connection Control window.
3. Right-click on the `MP3Player` entry and select **Connection Properties...** from the context menu.
4. Expand the following entries in turn:
   a. `CONNECTION=MP3Player (change as required)`
   b. `Advanced_information`
   c. `Default`
   d. `Memory_block`
5. Right-click on the `REGS` entry, in the left pane, and select **Make New...** from the context menu to see the prompt shown in Figure 3-34.

![Figure 3-34 Creating a new memory block](image)

6. Enter a new name for this entry, for example `MEM1`, and click **Create**.
7. Click on the MEM1 entry, in the left pane.
8. Set the value of Start to 0x0 (this is the default).
9. Set the value of Length to 0x80000.
10. Set the value of Description to Fast Static RAM.
11. Set the value of Access entry to RAM (this is the default).
12. Use Make Copy... on MEM1 to create a new group, MEM2.
13. Click on the MEM2 entry, in the left pane, to display the settings values.
14. Set the value of Access to ROM.
15. Set the value of Description to Slow Boot ROM.

Defining the memory rules

The second stage is to define the rules that control which memory block is used:

1. Expand the Map_rule group.

   The map rule defines which memory block to use. In this example, MEM2 is activated if Newreg is set to zero. Otherwise MEM1 is used.

2. Click on the Default entry.
3. Set the value of Register to Newreg (use the context menu).

   Do not change the settings for Mask or Value.
4. Set the value of On_equal to MEM1, shown in Figure 3-35.

   ![Figure 3-35 Creating a map rule](image)

   Figure 3-35 Creating a map rule

5. Rename the Default entry of Map_rule to RULE1.
6. Use Make Copy... on RULE1, to create a new group, RULE2.
7. Click on RULE2 and set the value of Value to 1.
8. Set the value of `on_equal` to `MEM2`, shown in Figure 3-36.

![Figure 3-36 Settings for the second map rule](image)

All board file entries are now complete.

9. Select **File → Save and Close** to save your changes and close the Connection Properties window.

10. Connect to your target.

11. In the Code window, select **View → Pane Views → Registers** to view the `MYREG` tab.

12. Toggle the register value to activate the memory rule and so specify the memory block.

13. Select **View → Pane Views → Memory Map** to view the `Map` tab, shown in Figure 3-37.

![Figure 3-37 New memory block in the Map tab](image)

### 3.4.6 Setting top of memory and stack heap values

This example demonstrates how you can set permanent top of memory and stack heap values for a given target using your board file. It shows how to do this by updating the information in the `Advanced Information` block. After you have defined the settings, they are used whenever you connect to the target with RealView Debugger.
Configuring Custom Targets

You can set the value of top_of_memory in a BOARD group used to configure the target or in the CONNECTION entry you use to connect to the target. The method described here applies to both. This example uses an Integrator/AP board with an ARM940T core module, but the procedure for amending the settings is the same for any target.

Note

This example uses the connection created in Setting up an Integrator board and core module on page 3-24. However, the AP, AM, and CM940T board groups are not linked to ensure a default memory mapping. If you do link the board groups, the contents of the Register pane will differ from those shown here because the value of top_of_memory is defined in the linked CM940T board group. This value overrides any value that you define in the CONNECTION entry.

The top_of_memory variable is used to enable the semihosting mechanism to return the top of stack. You can create your own settings to specify the bottom of the stack address, the size of the stack, the bottom of the heap address, and the size of the heap. If you do not set these values manually, RealView Debugger uses the target-dependent defaults. If your application is scatterloaded, you must define the stack and heap limits.

Note

The value of top_of_memory must be higher than the sum of the program base address, program code size, and program data size. If set incorrectly, the program might crash because of stack corruption or because the program overwrites its own code.

There is no requirement that the top of memory address is at the true top of memory. A C or assembler program can use memory at higher addresses.

To set the top of memory and stack heap values in the CONNECTION:

1. Ensure that RealView Debugger is not connected to a target.
2. Select File → Connection → Connect to Target... to display the Connection Control window.
3. Right-click on the MP3Player entry and select Connection Properties... from the context menu.
4. Expand the following entries in turn:
   a. CONNECTION=MP3Player (change as required)
   b. Advanced Information
   c. Default
   d. ARM_config
5. Set the value of top_of_memory, in the right pane, as required. For example, set it to 0x40000 if your target has 256KB of RAM starting at location 0.

--- **Note** ---

Be sure to specify a value that is supported by your debug target.

When you load a program, the debugger sanity-checks top_of_memory by checking that the words just below top_of_memory are writable. It issues a warning if they are not. However, your program might require much more RAM than the debugger checks for.

6. Double-click on the Stack_Heap group, in the right pane, to display the contents, shown in Figure 3-38.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack bottom</td>
<td>&lt;above heap&gt;</td>
</tr>
<tr>
<td>Stack size</td>
<td>0x2096</td>
</tr>
<tr>
<td>Heap base</td>
<td>0x4096</td>
</tr>
<tr>
<td>Heap size</td>
<td>0x4096</td>
</tr>
</tbody>
</table>

**Figure 3-38 Settings in the Stack_Heap group**

This shows the currently selected size and location of the stack and heap. A blank or zero Heap_base value is modified by the ARM C library runtime code, setting it to the address of the end of program data space, shown in Figure 3-39.

**Figure 3-39 Relating top_of_memory to single section program layout**

7. If you require more control over the stack and heap location for a semihosted program, set the values as required.
If your application requires control over the stack and heap location, or if the application is scatterloaded, the application must include a user-defined function, __user_initial_stackheap, that defines the stack and heap limits.

8. Select **File → Save and Close** to save your changes and close the Connection Properties window.

Whenever you load a program compiled with the standard ARM C library to this target, the top of memory, stack, and heap values you have set are used.

The default value of `top_of_memory` for ARM processors is 0x20000. For details on how this variable is relevant to ARM targets, see the internal variable descriptions sections in the user guide that accompanies your interface, for example *RealView ICE User Guide*.

### 3.4.7 Using RealMonitor

This example describes how to set up RealView Debugger and run a RealMonitor-integrated application. It demonstrates how to use the Connection Properties window to access the Multi-ICE server with RMHost to run the RealMonitor LEDs demonstration supplied with the *ARM Firmware Suite* (AFS). For more information see:

- *Multi-ICE User Guide*
- *ARM Firmware Suite User Guide*
- *ARM RMHost User Guide*
- *ARM RMTarget Integration Guide.*

To debug a RealMonitor-integrated application, you must connect with the RMHost controller while the program is running. Therefore, the example is split into sections, which must be executed in this sequence:

1. *Setting up the hardware and Multi-ICE server*
2. *Configuring the targets* on page 3-49
3. *Connecting and running the image* on page 3-50
4. *Configuring RealMonitor* on page 3-51
5. *Connecting and running the RealMonitor image* on page 3-52.

**Setting up the hardware and Multi-ICE server**

The first stage is to set up the hardware and configure the Multi-ICE server software:

1. Ensure that your Integrator/AP and core module are connected and switched on. This example uses the ARM7TDMI® core mounted on the CM7TDMI board, but you can use any core module supported by the Integrator/AP.
2. Configure the Multi-ICE server to work with RMHost.

**Note**

If you are using the Multi-ICE server with RMHost, you must ensure that it is not autoconfigured because this causes the target to be reset and disrupts any running program. See *ARM RMHost User Guide* for more details on configuring Multi-ICE server with RMHost.

**Configuring the targets**

The next stage is to configure the new target:

1. Start RealView Debugger without connecting to a target.
2. Select **File → Connection → Connect to Target...** to display the Connection Control window, shown in Figure 3-40.

![RealMonitor in the Connection Control window](image)

Figure 3-40 RealMonitor in the Connection Control window

Where you have installed RealMonitor, the RealMonitor.dll is autodetected by RealView Debugger and appears as a target in the Connection Control window. If it is not visible, see *Adding and configuring RDI targets* on page 4-14 for details on how to add this DLL to your list of RDI targets.

3. Right-click on the Multi-ICE connection and select **Connection Properties...** from the context menu. This displays the Connection Properties window with the connection expanded, shown in Figure 3-41 on page 3-50.
Configuring Custom Targets

Figure 3-41 Multi-ICE connection properties

4. Set the BoardChip_name in the right pane, as described in Setting up an Integrator board and core module on page 3-24, for example:
   - BoardChip_name=AP
   - BoardChip_name=CM7TDI.

5. Expand the following entries in turn:
   a. (*.rbe) ARM RDI Configuration Entries
   b. ...\rm.rbe
   c. CONNECTION=RealMonitor

6. Set the BoardChip_name in the right pane, for example:
   - BoardChip_name=AP
   - BoardChip_name=CM7TDI.

7. Select File → Save and Close to save your changes and close the Connection Properties window.

Connecting and running the image

The next stage is to connect to Multi-ICE and run the image:

1. Select File → Connection → Connect to Target... to display the Connection Control window.

2. Click on the icon next to the entry, Multi-ICE, to expand it.
   The entry expands and the relevant processor name entry is displayed.
3. Click on the processor entry under the Multi-ICE entry to connect to the target. RealView Debugger retrieves information specific to the target.

4. Select File → Load Image... to display the Load File to Target dialog box where you can locate the LEDs demonstration for your target, for example install_directory\AFSv1_4\Demos\Integrator7M\standalone\LEDs.axf.

5. Specify the location of the source, for example install_directory\AFSv1_4\Source\RealMonitor\Demos\Sources\Native\entry.s.

6. Select Debug → Execution Control → Go (Start Execution) to execute the image.

   The LEDs demonstration runs:
   
   - a foreground task to scroll text across the 15-segment alphanumeric LEDs on the Integrator board in an endless loop
   - a background task to cycle the three colored LEDs in a UK traffic light sequence.

   ——— Note ———
   See ARM RMTarget Integration Guide for full details about this demonstration.

7. Select File → Connection → Disconnect (Defining Mode)... and disconnect from the Multi-ICE target but leave the image running. See Setting disconnect mode on page 2-23 for full details on disconnecting this way.

8. Click OK to close the Disconnection Mode selection box.

Configuring RealMonitor

The next stage is to configure RealMonitor to use Multi-ICE:

1. Select File → Connection → Connect to Target... to display the Connection Control window.

2. Click on the icon next to the entry, RealMonitor, to expand it.

   The entry expands and the relevant processor name entry is displayed.

3. Right-click on the RealMonitor entry and select Configure Device Info... from the context menu. Use the configuration dialog box to configure RealMonitor to use Multi-ICE. Ensure that the JTAG Controller settings points to the Multi-ICE DLL. See Configuring ARM RealMonitor on page 4-26 for details.

4. Click OK to close the RealMonitor configuration dialog box.
Connecting and running the RealMonitor image

The next stage is to connect to the target and load the RealMonitor-integrated image:

1. Click on the icon next to the entry, RealMonitor, to expand it.
   The entry expands and the relevant processor name entry is displayed.

2. Right-click on the processor entry and select Connect (Defining Mode)... to connect to the Multi-ICE target without resetting the target. Select No-Reset and No-Halt Target from the Connection Mode selection box. See Setting connect mode on page 2-13 for details on connecting this way.

   If you select an option that is not supported by your target processor, a warning is displayed to show that RealView Debugger has not completed the request.

3. Click OK to close the Connection Mode selection box.

4. Select File → Load Image... to display the Load File to Target dialog box where you can locate the LEDs demonstration for your target.

   **Note**
   Ensure that you select the Symbols Only check box and unselect all other check boxes.

5. To get the context, you have to stop and restart the image:
   a. Select Debug → Execution Control → Stop Execution to stop the image.
   b. Select Debug → Execution Control → Go (Start Execution).

   Specify the location of the source when requested.

You are now ready to start debugging the image using RealView Debugger, for example by changing the country element of the user_state structure to cycle the traffic light LEDs in the US sequence, shown in Figure 3-42 on page 3-53.
3.4.8 Flash programming

Before you can use RealView Debugger to control a Flash memory chip on your target, you must:

- describe the Flash memory chip in a memory map entry, in a manner similar to that described in Configuring a memory map on page 3-35
- ensure that you have a correctly configured Flash MEthod (FME) file.

FME files include:
- code that enables writing to the Flash device
- code to perform write and erase operations
- information describing the way the Flash is configured on the bus.

The following example describes how to use the ARM Integrator FME file to program Flash memory on the Integrator/AP board. If you have another target board with a standard AMD, ATMEL, or Intel Flash device you must create a board-specific assembler file and link that file to create an FME file before you can program the Flash memory. If you are using another type of Flash memory, you must also create the Flash programming routines.

The board-specific assembler and Flash memory programming files are installed as part of the base product in the directory:

\install_directory\RVD\Core\...\flash\examples
Files are collected in subdirectories based on the target Flash device. The board-specific files have names starting with \_b\_, for example \_examples\aeb1\b_aeb1.s. The Flash memory files have names starting with \_f\_, for example \_examples\at91eb01\f_intel_arm.s.

RealView Debugger projects to create FME files from these sources are also provided, for example in:

\install_directory\RVD\Core\...\flash\examples\IntegratorAP\IntegratorAP.prj.

**Programming an image to the Integrator/AP Flash target**

This example describes how to use the predefined Integrator/AP Flash configuration to write an image to the Flash memory on the Integrator/AP board.

--- **Note** ---

If you program the Flash on an Integrator using this release of RealView Debugger, you bypass the AFS Flash library system information blocks. These blocks are used by the AFS Flash Library and are stored at the end of each image written to Flash. If you rely on these blocks to keep track of what is in the Flash memory of your target, keep a record of the state and recreate it after trying the example.

---

The example is split into these sections, which must be executed in this sequence:
1. *Defining the new target*
2. *Programming the image into Flash* on page 3-56.

**Defining the new target**

This example uses the connection created in *Setting up an Integrator board and core module* on page 3-24. However, the AP and O940T board groups are not linked before you start to ensure a default memory mapping.

To configure the Flash target:
1. Ensure that RealView Debugger is not connected to a target.
2. Select *File* → *Connection* → *Connect to Target*... to display the Connection Control window.
3. Right-click on the *MP3Player* entry and select *Connection Properties*... from the context menu to display the Connection Properties window.
4. Right-click on *BoardChip_name* in the right pane to display the context menu.
5. Select AP to select the Integrator/AP description, shown in Figure 3-20 on page 3-29. A new entry *BoardChip_name AP* is added to the pane.

6. Select File → Save and Close to save your changes and close the Connection Properties window.

7. Connect to your target.

8. Click on the Cmd tab in the Output pane, shown in Figure 3-43.

9. Select View → Pane Views → Memory Map to view the new memory map, shown in Figure 3-44, before loading an image.
Programming the image into Flash

To program the image, you request RealView Debugger to write to the Flash memory region that you have defined by using the Integrator/AP board file. The Integrator Flash starts at memory address 0x24000000, so to write an image to flash:

1. If necessary, create an image file compiled to run with code at 0x24000000 and that has data in RAM.
   This example uses the dhrystone project, located in your examples directory: install_directory\RVDS\Examples\...
   Open the project and rebuild using modified linker options. Set the Link_Advanced values in the BUILD group using Ro_base = 0x24000000 and Rw_base = 0x8000.

2. Click File \ Load Image... and select the image file.

3. Click Open in the Load File to Target dialog box. The Flash Memory Control dialog box appears, shown in Figure 3-45.

4. Click Write to program the image into Flash.
   You must wait for the Flash routine to load and run, that is wait until the last block has been written before continuing (shown in Figure 3-46 on page 3-57).
5. Click Close to close the Flash Memory Control dialog box.

**Use Current values...**

The option *Use Current values for Unspecified data in block* should only be used if you are updating part of the Flash block and you want to retain the current values in the rest of the block. This check box is unselected by default. If you select this option, RealView Debugger reads and then writes the entire block. This might take some time to complete.

### 3.4.9 Restoring your .brd file

If you have completed these examples and you want to return to the factory settings:

1. Exit RealView Debugger.
2. Delete your RealView Debugger home directory.

When you restart RealView Debugger it creates a new default configuration for you.

### 3.4.10 Troubleshooting

If your working versions of configuration files are accidentally erased, or become corrupted, RealView Debugger might be unable to use them. In this case, making a connection to your chosen target is not possible.

You can do one the following:

- If you have made a backup of your configuration, restore it as described in *Saving and restoring connection properties* on page 1-15.

- If it is acceptable to lose all of the configuration settings, program preferences, workspaces and other information that is stored in the debugger home directory, you can delete it:
  1. Exit RealView Debugger.
  2. Locate the home directory the debugger is using.
See *The home directory* on page 1-13 for more details.

3. Use Windows Explorer to rename or delete the home directory. You might want to move or rename it before deleting so that if you make a mistake you can recover selected files.

4. Restart RealView Debugger. It will create a new, default copy of the debugger home directory as it starts up.

- If there are configuration items that you wish to try to keep:
  1. Exit RealView Debugger.
  2. Using Windows Explorer, display the home directory the debugger is using. See *The home directory* on page 1-13 for more details.
  3. Using a second Windows Explorer window, locate the RealView Debugger installation directory. See *The install directory* on page 1-12 for more details.
  4. Use the hints given in *Using manual file or directory backups* on page 1-15 to copy files from the default settings directory `\etc` to your debugger home directory. Some of the `*.cnf` files have no default in `\etc`, and are recreated as required. If you believe it is causing problems, delete the version in your home directory and let the debugger recreate it when you next connect.
  5. Restart RealView Debugger.
Chapter 4
Configuring Custom Connections

This chapter describes how you can configure the connection that RealView® Debugger makes to your target. It includes information on the board file groups CONNECTION and DEVICE, and explains how connections to different targets are configured. It contains the following sections:

- Working with connection properties on page 4-2
- Working with RealView ICE targets on page 4-8
- Working with RDI targets on page 4-14
- Working with Simulator Broker connections on page 4-27
- Working with JTAG files on page 4-36.
4.1 Working with connection properties

As described in About target connections and configuration on page 1-2, you configure the way that RealView Debugger connects to, and interacts with, your debug target using connection properties contained in board file entries. A debug target might be a simulator, an emulator, or an evaluation board installed on your host workstation. Using connection entries enables you to configure:

- debugger to target connection details, such as interface type and instance, TAP controller positions, and connection interface address
- debugger actions taken when a connection is made, for example running commands and opening projects.

This section describes how to work with connection entries. It contains the following sections:

- Connection entries in the Connection Control window on page 4-3
- Enabling or disabling a board file entry on page 4-4
- Restoring board file entry defaults on page 4-5.

In the examples in the rest of this chapter, you are changing your board file. This is stored in your RealView Debugger home directory. Target configuration files might also be stored in this directory, for example .rvc files or .cnf files.

It is recommended that you back up this directory before starting the examples described in this chapter, so that you can restore your original configuration later. For details see:

- Configuration files on page 1-11 for instructions on making backups of your configuration
- Restoring your .brd file on page 3-57 for instructions on restoring a default configuration
- Troubleshooting on page 3-57 for instructions on recovering from an incorrectly configured debugger home directory, whether or not you have a backup.

Note

When you are following these examples, do not configure the board file when the debugger is connected to a target.

There are descriptions of the general layout and controls of the RealView Debugger settings windows, including the Connection Properties window, in the RealView Debugger online help topic Changing Settings. This chapter assumes you are familiar with the procedures described in this help topic.
4.1.1 Connection entries in the Connection Control window

Board file entries that are enabled form the basis of the information displayed in the Connection Control window, shown in Figure 4-1.

Figure 4-1 Connection properties entries in the Connection Control window

In Figure 4-1 the entry CONNECTION=ARM_MICE has been disabled in the board file. Therefore, this entry does not appear in the Connection Control window. Other connection entries are enabled and so are visible, for example CONNECTION=Multi-ICE, CONNECTION=RealView ICE, and the local host simulator using RealView Connection Broker.

You can enable and disable any entry in the board file if required. When you view the Connection Properties window, disabled entries are grayed out in the left pane, the List of Entries pane. Disabled entries can be edited in the same way as enabled entries and then enabled when available for connection. See Enabling or disabling a board file entry on page 4-4 for details on how to do this.

If you make changes to values in the Connection Properties window, an asterisk is added to each entry, in the left or the right pane, to show that the defaults have changed. You can restore the default settings and so cancel any changes. See Restoring board file entry defaults on page 4-5 for details on how to do this.
When you display the Connection Properties window, the left pane shows the top-level entries specifying the supported vehicles, for example ARM RDI Configuration Entries. Board file entries might have duplicate names because entries are uniquely identified through the combination of three elements:

- the CONNECTION entry
- the name of the manufacturer of the simulator, emulator, or board
- the host workstation I/O device address, IP address (for remote connections), or ID, of the emulator or board.

For example, assume that you have a target board named MP3Player that you want to use with two different emulators. The board file entry name for each is MP3Player to reflect the target. However, the entries are differentiated by the type of connection (emulator type), and I/O device connection addresses.

--- Note ---
You can create your own custom entries in this hierarchy using other types of entry, for example BOARD, CHIP, COMPONENT, or DEVICE. However, if you are creating custom, lower-level entries, it is recommended that you avoid duplicate names.

---
For full information on the contents and values contained in different types of board file entries, both default and custom, see Appendix A Configuration Settings Reference.

4.1.2 Enabling or disabling a board file entry

To disable a board file entry so that the target it represents is no longer offered for selection in the Connection Control window:

1. Start RealView Debugger without connecting to a target.
2. Select File → Connection → Connection Properties... to display the Connection Properties window.
   
   Enabled entries in the left pane are displayed in regular type, and those that are disabled are grayed out.

3. Expand the connection that you are using, for example RVBROKER=localhost.
   It becomes the selected entry and its contents are displayed in the right pane, shown in Figure 4-2 on page 4-5.
Figure 4-2 Enabling or disabling a board file entry

4. Right-click on the Disabled entry, in the right pane, and select True from the menu.

5. Select File → Save and Close to save your changes and close the Connection Properties window.

6. Select File → Connection → Connect to Target.... to display the Connection Control window where this entry is no longer available.

4.1.3 Restoring board file entry defaults

An entry starts with an asterisk when it has been edited. For group entries, this might mean that a value lower down in the hierarchy has been edited.

In Enabling or disabling a board file entry on page 4-4, the RVBR0KER=localhost entry was disabled. This means that it now contains a custom setting, that is the default setting has been changed. To restore the default values for this entry:

1. Select File → Connection → Connection Properties... to display the Connection Properties window.

2. Click on the RVBR0KER=localhost entry, in the left pane. It becomes the selected entry and its contents are displayed in the right pane, shown in Figure 4-3 on page 4-6.
3. Right-click on the *Disabled entry, in the right pane, and select Reset to Default from the context menu.
   This sets the value of this setting to False as defined in the default board file. The asterisk is removed.

4. Select File → Save and Close to save your changes and close the Connection Properties window.
   The original contents of the Connection Control window are restored.

Where entries contain user-information values, these can be customized in a similar way:

1. Select File → Connection → Connection Properties... to display the Connection Properties window.

2. Click on the RVBROKER=localhost entry, in the left pane. It becomes the selected entry and its contents are displayed in the right pane.

3. Right-click on the Description entry, in the right pane, and select Edit Value from the context menu.
   The text defined by this entry appears in the Description in the Connection Control window, shown in Figure 4-1 on page 4-3. Enter a new description, for example RealView ARMulator ISS and press Enter.

4. Select File → Close Window to close the Connection Properties window without saving this change. This generates a dialog box warning that contents have changed and giving you the option of saving them. Do not save this change.
This example demonstrates:

- the Description setting has been changed in the default board file, that is Simulator Broker is not the default setting for this value.
- because the default setting has been changed, an asterisk marks the entry.
- RealView Debugger warns you if you have not saved changes you have made to the board file settings.
4.2 Working with RealView ICE targets

When you install the RealView ICE software, it adds entries to your RealView Debugger installation. This updates the default board file so that an ARM-ARM-NW target vehicle appears in the Connection Control window to provide the interface to RealView ICE targets, shown in Figure 4-4.

![RealView ICE targets in the Connection Control window](image)

**Note**

If the RealView ICE target vehicle is not visible in the Connection Properties window, you must add it before continuing with this section. See the chapter describing configuring a RealView ICE connection in *RealView ICE User Guide* for full details on how to do this.

If you expand the ARM-ARM-NW vehicle, you can see the access-provider connection defined automatically, that is RealView ICE, shown in Figure 4-4. You can then configure this connection by specifying an appropriate .rvc file, ready to make a connection. However, it is recommended that you use this default connection as the basis for new connections that you create.

This section describes:
- Creating a new RealView ICE connection on page 4-9
- Configuring a RealView ICE interface unit on page 4-11.

**Note**

You can use RealView ICE to connect to a target that incorporates DSP hardware with a suitable JTAG configuration, see Configuring a new DSP connection on page 4-39 for details. However, at this stage of development, it is not possible to connect to DSP cores using RealView ICE.
4.2.1 Creating a new RealView ICE connection

This example defines a new RealView ICE connection. You can do this by creating a new connection entry or by copying an existing entry.

--- Note ---

If you copy a connection entry, certain configuration details are not updated until you close the Connection Properties window. This might mean that some asterisks are not immediately visible in the copy.

---

This example creates a new connection entry. It assumes that a correctly configured .rvc file exists for the new target and this has been saved in the default RealView ICE installation directory. If you do not have this file, you can follow the example but you must also follow the instructions in *Configuring a RealView ICE interface unit* on page 4-11 before you can connect to the new target.

To define the new connection:

1. Select **File → Connection → Connection Properties...** to display the Connection Properties window.
2. Right-click on the **CONNECTION=RealView ICE** entry, in the left pane.
3. Select **Make New...** from the context menu.
4. This displays the Group Type/Name selector dialog box, shown in Figure 4-5.

---

![Group Type/Name selector](figure4-5.png)

**Figure 4-5 Specifying a new CONNECTION group**

Leave the type of the new entry unchanged as **CONNECTION**.

In the Group Name data field change the name from **RealView ICE** to something suitable for your target, using only alphanumeric characters, underscore _, and dash -, for example **New_RealView_ICE_2**.

This can be a descriptive name or the name of the new .rvc file that you are going to select, but without the .rvc extension.
5. Click **OK** to confirm your settings and to close the Group Type/Name selector dialog box.

   The new entry appears in the left pane of the Connection Properties window. It is automatically selected, and its details are displayed in the right pane. These details are the default for a new CONNECTION and you must change at least the Connect_with/Manufacturer, the Configuration filename and target Description. The next steps explain how to make these changes.

6. In the right pane of the Connection Properties window, right-click on the **Configuration** entry and select **Edit as Filename** from the context menu.

   The Enter New Filename dialog box is displayed to enable you to locate the required .rvc file, for example **RealViewICE2.rvc**.

7. Click **Save** to confirm your entries and to close the Enter New Filename dialog box.

   The new pathname is displayed in the right pane.

8. In the right pane of the Connection Properties window, right-click on the **Description** field and select **Edit Value** from the context menu.

   Type **RealView ICE connection to test hardware board** in the entry area and press Enter.

   This is the description displayed in the Connection Control window and Connection Properties window to identify the new target.

9. In the right pane of the Connection Properties window, right-click on the **Connect_with** entry and select **Explore** from the context menu.

10. In the right pane of the Connection Properties window, right-click on the **Manufacturer** entry and select the required connection type from the context menu, that is **ARM-ARM-NW**.

    If you do not specify this setting, the new connection appears in the Connection Control window but, when you try to connect, RealView Debugger prompts for the connection type.

11. If you want to carry out trace operations on the new target, you must also define the analyzer settings. If you copy an existing setting, this is set for you.

    Expand the following entries in turn:

    a. **CONNECTION=New_RealView_ICE_2**
    b. **Advanced_Information**
    c. **Default**
    d. **Logic_Analyzer**
Right-click on Vendor and select, for example, ARM from the context menu.

12. If you want to configure your target, for example by referencing a .bcd file, make these changes now.

13. Select File → Save and Close to save your changes and close the Connection Properties window.

Your new RealView ICE target is now displayed in the Connection Control window.

4.2.2 Configuring a RealView ICE interface unit

Use RVConfig to configure a new RealView ICE interface unit:

1. Right-click on the new RealView ICE connection, shown in Figure 4-6.

![Figure 4-6 Configuring the new RealView ICE connection](image)

2. Select the option Configure Device Info... to display the RVConfig dialog box shown in Figure 4-7 on page 4-12.
3. Click **Browse** to locate the RealView ICE unit, if you do not know the correct address.

4. Click **Connect** to connect the RVConfig dialog box to the RealView ICE unit.

5. Click **AutoConfigure Scan Chain** to detect all supported ARM® processors and add them to the scan chain.

6. Select the device, for example ARM940T, in the left pane, to configure it using the supplied template. For example, you might want to set the **Code Sequence Code Address** or **Code Sequence Code Size**, to specify target memory available to RealView ICE.

7. Select **Advanced**, in the left pane, to configure advanced settings, for example, you might want to click the **Reset On Connect (Default)** so that the target hardware is reset when you connect.

If you are using an Integrator™ board, as in this example, it is recommended that you use the settings shown in Figure 4-8 on page 4-13.

![Figure 4-7 Configuring a RealView ICE interface unit](image-url)
Configuring Custom Connections

Figure 4-8 Recommended settings for an ARM Integrator board

8. Select **File** → **Save** to save your changes.

9. Select **File** → **Exit** to close the RVConfig dialog box.

You can now connect to your new target in the usual way.

**Configuring RealView ICE units for different targets**

You might have to use different configuration settings, from those shown in this example, depending on your debugging environment. Remember the following when specifying these settings:

- Autoconfiguration does have side effects and might be intrusive. Where this is not acceptable, you should configure the target manually.

- You must use different settings if your application uses interrupts to set up pre-initialization code on the target, for example if you are using *Running System Debug* (RSD). See *RealView Debugger v1.7 Extensions User Guide* for details.

- The RealView ICE scan chain configuration lists devices in ascending order of TAP ID. This is the opposite order to that used by Multi-ICE®.

For full details on using RVConfig, and on how to add new devices to the scan chain, see the chapter describing configuring a RealView ICE connection in *RealView ICE User Guide*. 
4.3 Working with RDI targets

This section describes how to use the Connection Control window to add and configure RDI targets. The settings defined in your configuration files control the available targets and the emulators and simulators offered. Therefore, your installation might vary from the examples shown in this section.

--- Note ---

RDI connections cannot be made if you are using Sun Solaris or Red Hat Linux.

---

The RealView Debugger base product includes RDI configuration files to simulate the ARM7TDMI® core using RealView ARMulator® ISS (RVISS). You can change this by reconfiguring RVISS, described in Configuring RealView ARMulator ISS on page 4-18.

When you first try to connect to another RDI target, for example Multi-ICE, you must configure the target first, described in Configuring ARM Multi-ICE on page 4-22.

To add a new RDI target to the RealView Debugger configuration settings, you must:

1. add the required DLL to specify the device
2. configure the new target.

See the following sections for details on these operations:
- Adding and configuring RDI targets
- Configuring RealView ARMulator ISS on page 4-18
- Configuring Remote_A on page 4-21
- Configuring ARM Multi-ICE on page 4-22
- Configuring ARM Agilent Debug Interface on page 4-25
- Configuring ARM RealMonitor on page 4-26.

4.3.1 Adding and configuring RDI targets

RDI targets are automatically installed for ARM products such as Multi-ICE. RealView Debugger also includes DLLs to enable you to add an Angel debug target. The debugger provides the RDI Target List dialog box where you can add more RDI targets or configure targets detected automatically.

Adding RDI targets

If you have a third-party RDI component that uses RDI 1.5.1, you can use the following procedure to include it.
To add an RDI target to the RDI Target List dialog box:

1. Start RealView Debugger without connecting to a target.
2. Select **File → Connection → Connect to Target...** to display the Connection Control window.
3. Right-click on the ARM RDI vehicle, ARM-A-RR, to display the context menu.
4. Select the option **Add/Remove/Edit Devices...** to display the RDI Target List configuration dialog box shown in Figure 4-9.

![Figure 4-9 RDI Target List dialog box](image)

The entries in the display list show the autodetected targets currently available to you. If you install a new DLL outside RealView Debugger, for example ARM ADI, the display list is automatically updated for you.

The check boxes show which devices are enabled. Disabling an entry removes it from the list of available connections shown in the Connection Control window. The entry is not, however, removed from the RDI Target List and so can be re-enabled when required.

If you do want to add a target yourself, click **Add DLL...** to display the Select RDI DLL dialog box where you can locate the required DLL and add it to the list.

Any entry in the display list can be duplicated so that specific processor configurations are available in the Connection Control window. Highlight the entry to be copied and click **Duplicate...** to display the Create New RDI Target dialog box where the new target can be specified, shown in Figure 4-10 on page 4-16.
Enter:

**Short Name**  The target name as shown in the RDI Target List dialog box, the Connection Control window, and in the Connection Properties window, for example `CONNECTION=Multi-ICE_copy`.

**Description**  The target description as shown in the RDI Target List dialog box, the Connection Control window, and in the Connection Properties window, for example `Description=ARM JTAG debug interface (parallel port)`.

Click **OK** to confirm your entries and to add the target to the display list.

Any DLLs added to the list manually, or new entries created by duplicating an existing target, must be configured before you can connect. Highlight the entry in the display list and click **Configure...** to display the configuration dialog box for the chosen entry. This is described in detail in *Configuring RDI targets* on page 4-17.

You can use the RDI Target List dialog box to remove RDI targets from the list. Highlight the entry to be deleted and click **Remove**.

You can also use the RDI Target List dialog box to reset all entries to the installation defaults by clicking **Reset list**.

**Note**
When working with the RDI Target List dialog box:

- The Angel debug monitor, `Remote_A`, is autodetected if you install ADS. When you install this DLL as part of RealView Debugger, you must add it to the list yourself, from its location in the `\lib` directory.

- Autodetected targets cannot be removed from the display list.

- If you use **Reset list** to reset list entries, any settings changed, or targets added to the list, since installation are also removed. There is no undo.
Click **Close** to close the RDI Target List dialog box and return to the Connection Control window.

**Configuring RDI targets**

RealView Debugger does not support connecting to multiple RDI targets at the same time. You can, however, configure your target connections in advance and then connect to each in turn.

Within a single Multi-ICE connection, you can connect to multiple processors provided they are all on the same scan chain. You must have the appropriate RealView Debugger multiprocessor license to make multiple connections.

You can configure your RDI target either:

- from the Connection Control window. Highlight the entry, for example the second-level entry **Multi-ICE**, and select **Configure Device Info...** from the RDI Target menu
- from the RDI Target List dialog box, shown in Figure 4-9 on page 4-15. Highlight the entry in the display list and click **Configure**.

**Note**

If you cannot see your RDI target in the Connection Control window then you must add it using the RDI Target List dialog box. See *Adding RDI targets* on page 4-14 for details on how to do this.

The rest of this section contains basic configuration information for the following ARM RDI interface software, with details of changes that result from the use of RealView Debugger:

- **Configuring RealView ARMulator ISS** on page 4-18
- **Configuring Remote_A** on page 4-21
- **Configuring ARM Multi-ICE** on page 4-22
- **Configuring ARM Agilent Debug Interface** on page 4-25
- **Configuring ARM RealMonitor** on page 4-26.

These instructions do not replace the original manuals.
4.3.2 Configuring RealView ARMulator ISS

RVISS is the ARM processor simulator supplied with RealView Developer Suite (RVDS). This section describes how to configure the RDI interface to RVISS.

--- Note ---
RVISS also provides a RealView Connection Broker interface. See Working with Simulator Broker connections on page 4-27 for details on configuring this interface.

For the ARMulator RDI target entry, the configuration dialog box is shown in Figure 4-11.

![ARMulator Configuration dialog box](image-url)
The ARMulator Configuration dialog box (Figure 4-11 on page 4-18) enables you to examine and change the following settings:

**Processor**  
Use the drop-down list to specify which ARM processor you want RVISS to simulate.  
The list of processors includes all available variants including, for example, ARM7TDMI-ETM or ARM920T-ETM.

**Clock**  
Choose between simulating a processor clock running at a speed that you can specify, or executing instructions in real-time by setting this value to 0. You can use units of Hz, KHz, MHz and GHz, for example 50MHz.  
Changing this value does not affect the real time taken to run a program. Instead, it affects the values that the semihosting time() functions return to the program.

**Options**  
Enable this to include a simulation of the Floating Point Accelerator (FPA) coprocessor included in the ARM7500FE processor.

**Debug Endian**  
Select the byte order of the modeled system. This setting:
- sets RealView Debugger to work with the appropriate byte order
- sets the byte order of models that do not have a CP15 coprocessor
- sets the byte order of models that do have a CP15 coprocessor if the Start target Endian option is set to **Debug Endian**.

**Start target Endian**  
Select the way in which the byte order of RVISS models that have a CP15 coprocessor is determined:
- Select the **Debug Endian** radio button to instruct the model to use the byte order set in the Debug Endian group.
- Select the **Hardware Endian** radio button to instruct the model to simulate the behavior of real hardware.
The possible combinations of Debug Endian and Start target Endian are shown in Table 4-1.

<table>
<thead>
<tr>
<th>Usage</th>
<th>Debug Endian</th>
<th>Start target Endian</th>
</tr>
</thead>
<tbody>
<tr>
<td>A target that is always little-endian. This is the default.</td>
<td>Little</td>
<td>Debug Endian</td>
</tr>
<tr>
<td>A target that is always big-endian</td>
<td>Big</td>
<td>Debug Endian</td>
</tr>
<tr>
<td>A big-endian target where the code and the processor core start in little-endian mode, and switch to big-endian in the initialization code</td>
<td>Big</td>
<td>Hardware Endian</td>
</tr>
</tbody>
</table>

**Memory Map File**

Specify a memory map file for use with RVISS.

A map file that is specified in this way is not available to RealView Debugger. Use the `Memory_block` configuration item to specify the memory map to RealView Debugger. See *Configuring a memory map* on page 3-35 for an example explaining how to do this.

**Floating Point Coprocessor**

Use the drop-down list to specify the Vector Floating Point (VFP) coprocessor included with some ARM CPUs. The default is No_FPU.

**MMU/PU Initialization**

If you are simulating a processor with an active *Memory Management Unit* (MMU), specify DEFAULT_PAGETABLES, otherwise select NO_PAGETABLES. See *ARM Architecture Reference Manual* for more information.

See *RealView ARMulator ISS User Guide* for more information on how these settings apply to RVISS.
4.3.3 Configuring Remote_A

To enable RealView Debugger to communicate with an Angel debug target, use Remote_A, supplied with RealView Debugger.

--- Note ---
Remote_A.dll is installed as part of RealView Developer Suite, you must add it to the RDI Target List yourself, from its location in the \lib directory (see Adding RDI targets on page 4-14 for details).

For the Remote_A RDI target entry, the configuration dialog box is shown in Figure 4-12.

![Figure 4-12 Remote_A connection dialog box](image)

The Remote_A connection dialog box enables you to examine and change the following settings:

**Remote connection driver**

Click Select... to see a list of available drivers. This includes Serial, Serial/Parallel, and Ethernet drivers. Select one if you want to use it instead of the current driver.

To change the settings of the currently selected driver, click Configure.... This displays the dialog box appropriate to the chosen driver.
Heartbeat  Ensures reliable transmission by sending heartbeat messages. Any errors are more easily detected when known messages are expected regularly.

Endian  These radio buttons specify that the target is operating in little-endian or big-endian mode.

This setting is only used if you are connected to an EmbeddedICE® Interface Unit.

Channel Viewers  

RealView Debugger does not support channel viewers.

If you add Remote_A to the RDI Target List dialog box yourself, you must configure the target before you can connect. When you first display the Remote_A connection dialog box, all settings are empty. Use Select... to choose the driver DLL. Configure the other settings as required.

4.3.4 Configuring ARM Multi-ICE  

You can use the Multi-ICE interface unit in two ways:

- If you are only connecting to ARM processors, use the RDI Multi-ICE DLL and the Multi-ICE server.

- If you want to connect to a DSP, use the Multi-ICE interface unit with no Multi-ICE server and the ARM-ARM-PP Multi-ICE direct connect vehicle.

See Working with JTAG files on page 4-36 for general information about configuring Multi-ICE direct connect.

See RealView Debugger v1.7 Extensions User Guide for more information about connecting to DSP processors.

For the Multi-ICE RDI target entry, the configuration dialog box is shown in Figure 4-13 on page 4-23.
Figure 4-13 Multi-ICE DLL configuration dialog box

The ARM Multi-ICE dialog box, shown in Figure 4-13, contains the tabs:

**Connect**

This tab contains the **This computer...** and **Another computer...** buttons that enable you to select the Windows workstation that is running the Multi-ICE server, and the **Connection name** data field that enables you to identify each processor connection.

**Processor Settings**

This tab contains any processor-specific settings. See *Multi-ICE User Guide* for details of these.

**Advanced**

This tab contains the target endianess and interface settings. Use the radio buttons in the Target Settings group as shown in Table 4-2.

<table>
<thead>
<tr>
<th><strong>Usage</strong></th>
<th><strong>Selection</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>A target that is always little-endian. This is the default.</td>
<td>Little-endian</td>
</tr>
<tr>
<td>A target that is always big-endian.</td>
<td>Big-endian</td>
</tr>
</tbody>
</table>
Disable the Read-ahead cache if you are accessing read-sensitive memory with the debugger.

**Trace**
Use this tab to enable and configure your Trace Capture tool. Where enabled, select the required Trace Capture DLL from the list, or use Add... to locate a new DLL.

**About**
Displays information about the version numbers of the Multi-ICE DLL and RealView Debugger.

**Configuring multiple targets using Multi-ICE**

If you connect to a Multi-ICE server that is configured with more than one ARM processor, the configuration dialog box includes a side-panel showing icons for each of these processors. This side-panel is not available if you connect to a single processor, shown in Figure 4-13 on page 4-23.

Selecting a processor in the side-panel enables you to independently configure properties for that processor using the **Processor Settings** and **Advanced** tabs. For example, you can set the processor setting **Cache clean code address**, differently on two processors:

1. Click **Processor Settings** tab.
2. Select the first processor in the side-panel.
3. Change the value of **Cache clean code address**.
4. Select the second processor in the side-panel.
5. Change the value of **Cache clean code address**.

With RealView Debugger all the available processors are configured into the Connection Control window and you use that window to connect and disconnect from each processor as required.

**Connecting to multiple targets using Multi-ICE**

It is recommended that you turn off the cache mechanism in Multi-ICE when debugging multiple processors:

1. Select **File → Connection → Connect to Target**... to display the Connection Control window.
2. Right-click on the **Multi-ICE** entry and select **Configure Device Info**... from the context menu.

   The Multi-ICE DLL configuration dialog box is displayed.
3. Click the Advanced tab.

4. Ensure that the Start-up with cache enabled check box is not selected, and click OK.

More information about configuring Multi-ICE is given in Multi-ICE User Guide and in the online help available from the dialog box.

Note
• Releases of Multi-ICE before 1.4 are not compatible with RealView Debugger.
• RealView Debugger supports DCC semihosting with Multi-ICE. When this mode is used, the target processor is not stopped while semihosting takes place.
• RealView Debugger does not support multiple simultaneous connections to Multi-ICE.

4.3.5 Configuring ARM Agilent Debug Interface

The ARM Agilent Debug Interface (ADI) configuration dialog box enables you to change the following settings:

Network details
The network (Ethernet) address of the Agilent JTAG probe.

JTAG Frequency
The frequency of the IEEE1149.1 TCK signal. A frequency of 10MHz is suitable for most ARM processor cores.

Device Configuration
The Specify Devices... button displays the Specify Devices dialog box, and enables you to select the processor with which to connect. RealView Debugger does not support multiple simultaneous connections with ARM ADI Version 1.0.

See ARM Agilent Debug Interface User Guide for more information.
4.3.6 Configuring ARM RealMonitor

The ARM RealMonitor configuration dialog box enables you to change the following settings:

**JTAG Controller**

The RDI-compliant JTAG controller DLL, for example your Multi-ICE DLL.

**RDI Module Server**

This enables you to view special target registers in RealView Debugger:

- **Use RDI Module Server**
  
  Ticked by default, click to disable the RDI Module Server. If disabled, this grays out the second option in this group.

- **Fetch module information from target**
  
  Ticked by default, this provides the module server with information about the target system. This might be embedded in RMTarget. If you unselect this option then you must provide:
  
  - the target processor you are using
  - the target board you are using.

  See [ARM RMTarget Integration Guide](#) for more information.

**Configure**

Click to configure the JTAG controller you have selected, for example the Multi-ICE server, shown in Figure 4-13 on page 4-23.

See [Multi-ICE User Guide](#) or the documentation that accompanies your JTAG unit for more details.

See [ARM RMHost User Guide](#) for more information.
4.4 Working with Simulator Broker connections

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

For more details on using RealView Connection Broker see:
- the chapter describing getting started in RealView Debugger v1.7 User Guide.
- Chapter 5 Working with Remote Targets.

Operating as RealView Simulator Broker, RealView Connection Broker runs on your local workstation and enables you to access targets on the local workstation. This means that local host simulators are available immediately from the Connection Control window. If you expand the Simulator Broker entry, ready to connect to a local host simulator, RealView Debugger starts RealView Connection Broker in local mode to manage your connections.

The RealView Debugger base product includes support for simulators to emulate the instruction sets of different processors and their supporting architecture. Where you have installed the software, and you have the appropriate licenses, you can use the default board file to make Simulator Broker connections to:

- RealView ARMulator ISS
- Motorola M56621 DSP simulator
- supported DSP simulators, for example the MaxSim System-on-Chip (SoC) simulator from AXYS Design Automation Inc.

This section describes:
- Connecting to RealView ARMulator ISS
- Configuring RealView ARMulator ISS on page 4-28
- Configuring a new simulator connection on page 4-31
- Using other simulators on page 4-35.

4.4.1 Connecting to RealView ARMulator ISS

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

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. See Configuring a new simulator connection on page 4-31 for details.
The Simulator Broker connection to RVISS enables the debugger to connect to a target using a network connection. RealView Debugger must provide the Simulator Broker interface to RVISS. RVISS models communicate with Simulator Broker through an intermediate interface called SimRdi_Manager.

Simulator Broker connections to RVISS support the following features:

- instruction and data trace
- hardware breakpoints
- multiple instances of RVISS.

**Note**

RVISS is the only way to connect to simulated ARM cores with RealView Debugger on Sun Solaris or Red Hat Linux.

To connect to RVISS:

1. Select **File → Connection → Connect to Target...** to display the Connection Control window.
2. Expand the entry **Server Connection Broker**.
3. Expand the entry **localhost Simulator Broker**.
4. Double-click on the **new_ARM** entry to connect to the simulator.

The RealView Connection Broker interface of RVISS enables you to connect to multiple targets at the same time. For details see the chapter describing multiprocessor support in *RealView Debugger v1.7 Extensions User Guide*.

### 4.4.2 Configuring RealView ARMulator ISS

To configure your RVISS connections from the Connection Control window:

1. Right-click on the required entry, **new_ARM**.
2. Select **Configure Device Info...** from the context menu to display the configuration dialog box is shown in Figure 4-14 on page 4-29.
Configuring Custom Connections

The ARMulator Configuration dialog box enables you to examine and change the following settings:

**Processor**

Use the list to specify which ARM processor you want RVISS to simulate.

The list of processors includes all available variants currently supported on Simulator Broker connections.

**Clock**

Choose between simulating a processor clock running at a speed that you can specify, or executing instructions in real-time.

Specify speed in Hz, for example 50000Hz.

Changing this value does not affect the real time taken to run a program. Instead, it affects the values that the semihosting time() functions return to the program.

**Debug Endian**

Select the byte order of the modeled system. This setting:

- sets RealView Debugger to work with the appropriate byte order
- sets the byte order of models that do not have a CP15 coprocessor
- sets the byte order of models that do have a CP15 coprocessor if the Start target Endian option is set to **Debug Endian**.
Start Endian

Select the way in which the byte order of RVISS models that have a CP15 coprocessor is determined:

- Select the **Debug Endian** radio button to instruct the model to use the byte order set in the Debug Endian group.
- Select the **Hardware Endian** radio button to instruct the model to simulate the behavior of real hardware.

The possible combinations of Debug Endian and Start target Endian are shown in Table 4-3.

<table>
<thead>
<tr>
<th>Usage</th>
<th>Debug Endian</th>
<th>Start target Endian</th>
</tr>
</thead>
<tbody>
<tr>
<td>A target that is always little-endian. This is the default.</td>
<td>Little</td>
<td>Debug Endian</td>
</tr>
<tr>
<td>A target that is always big-endian</td>
<td>Big</td>
<td>Debug Endian</td>
</tr>
<tr>
<td>A big-endian target where the code and the processor core start in little-endian mode, and switch to big-endian in the initialization code</td>
<td>Big</td>
<td>Hardware Endian</td>
</tr>
</tbody>
</table>

Floating Point

Use the radio buttons to specify the Vector Floating Point (VFP) coprocessor included with some ARM CPUs. The default is **No FPU**.

MMU/PU Initialization

If you are simulating a processor with an active Memory Management Unit (MMU), specify Default Page-Tables. See ARM Architecture Reference Manual for more information.

Be aware of the following features when using this dialog box:

- Some cores are not available when you are connected using RVISS, for example StrongARM™, ARM8 and the simulated *Embedded Trace Macrocell™* (ETM) functionality.
- Use the Additional Modules control group to define Floating Point Accelerator (FPA) and Vector Floating Point (VFP) options (VFPv1 is no longer available).
- If you choose **Real-time**, do not enter a value for Speed.
4.4.3 Configuring a new simulator connection

Simulator Broker connections to local host simulators are configured with custom settings by default. However, you might want to create new connections to simulate other peripherals and custom memory systems. For example, during the early stages of your development project, you might want to simulate an RTOS-enabled connection to reference a vendor-supplied board file. This enables accurate benchmarking of ARM-targeted software although performance is slower than with real hardware.

This example creates a new local host connection using RVISS. It uses the default configuration file for ARM simulators, default.auc, created in your RealView Debugger home directory when you installed the base product. This simulates an ARM7TDMI core. It is recommended that you back up this file before starting the example, so that you can restore your original configuration later. If you want to create different simulator connections, for example to simulate an ARM940T core, make copies of this file using appropriate names.

For details on using RealView Connection Broker in remote mode, and how to create a connection to a remote host, see Chapter 5 Working with Remote Targets.

The example is split into the following sections:

- Creating the configuration file
- Creating the new connection on page 4-32
- Viewing details of the new connection on page 4-34.

Creating the configuration file

To create a configuration for the new connection:

1. Locate the file default.auc in the directory that RealView Debugger is using for your session.
   
   See The home directory on page 1-13 for more information about the location of your debugger home directory.

2. Copy the file.

3. Rename the copy, for example Tst_SIM_Broker_ARM940T.auc.

It is not necessary but it is recommended that you create a new .auc file when you create a new simulator connection.
Creating the new connection

To define the new connection:

1. Select File → Connection → Connection Properties... to display the Connection Properties window.
2. Right-click on the ...\rvdebug.brd entry, in the left pane, to display the context menu, shown in Figure 4-15.

![Figure 4-15 Making a new group in the board file](image)

3. Select Make New Group... from the context menu, to display the Group Type/Name selector dialog box, shown in Figure 4-16.

![Figure 4-16 Specifying a new CONNECTION group](image)

4. Define the new group:
   - Leave the type of the new entry unchanged as CONNECTION.
In the Group Name data field change the name from new to something suitable for your target, using only alphanumeric characters, underscore _, and dash -, for example New_Test_SIM.
This can be a descriptive name or the name of the .auc file that you are going to select, but without the .auc extension.

5. Click OK to confirm your settings and to close the Group Type/Name selector dialog box.
The new entry appears in the left pane of the Connection Properties window. It is automatically selected, and its details are displayed in the right pane. These details are the default for a new CONNECTION and you must change at least the Connect_with/Manufacturer, the Configuration filename and target Description. These changes are described in the following steps.

6. In the right pane of the Connection Properties window, right-click on the Configuration entry and select Edit as Filename from the context menu.
The Enter New Filename dialog box is displayed to enable you to locate the required .auc file, for example default.auc or Tst_SIM_Broker_ARM940T.auc, the processor-specific file created in Creating the configuration file on page 4-31.
If RealView Debugger cannot locate the specified file, you are offered the option to create the copy now or to define a new file for this connection.

7. Click Save to confirm your entries and to close the Enter New Filename dialog box.
The new pathname is displayed in the right pane.

8. In the right pane of the Connection Properties window, right-click on the new *Configuration entry and select Edit Configuration-File Contents... from the context menu.
Use the configuration dialog box, shown in Figure 4-14 on page 4-29, to configure your new RVISS connection (see Configuring RealView ARMulator ISS on page 4-28 for details).
In this example, specify that the ARM940T core is simulated.
Click OK to save your changes and close the configuration dialog box.

9. In the right pane of the Connection Properties window, right-click on the Description field, and select Edit Value from the context menu.
Type ARM940T + RVISS to new test hardware in the entry area and press Enter.
This is the description displayed in the Connection Control window and Connection Properties window to identify the new target.
10. In the right pane of the Connection Properties window, right-click on the Connect_with entry and select **Explore** from the context menu.

11. In the right pane of the Connection Properties window, right-click on the Manufacturer entry and select the required connection type from the context menu, that is **ARM-A-SW**.

   If you do not specify this setting, the new connection appears in the Connection Control window but, when you try to connect, RealView Debugger prompts for the connection type.

12. If you want to configure your target, for example by defining ARM configuration settings such as semihosting, make these changes now.

   Expand the following entries in turn:
   a. CONNECTION=New_Test_SIM
   b. Advanced_Information
   c. Default
   d. ARM_config

13. Select **File → Save and Close** to save your changes and close the Connection Properties window.

   Your new simulator is now displayed in the Connection Control window where you can connect in the usual way, shown in Figure 4-17.

![Connection Control (My_user_name\rvdebug.hro)](image)

**Figure 4-17 Connecting to a new simulator**

**Viewing details of the new connection**

When you connect to the new simulator, the Output pane shows details about the connection to RVISS, shown in Figure 4-18 on page 4-35.
Connecting to the new simulator also creates a new entry in the Connection Control window under `localhost Simulator Broker`, shown in Figure 4-17 on page 4-34. If you try to click on this entry to connect, this fails. The Output pane displays details of the error, for example:

```
Error: Failed on remote SIM/EMU/EVM. In use by my_user_name
```

This is because Simulator Broker views this as a remote connection. When you disconnect from the new connection, this entry also disappears.

### 4.4.4 Using other simulators

Other simulators, including the MaxSim simulator, from AXYS Design Automation Inc., or the Motorola M56621 DSP simulator, might be available from the RealView Debugger Connection Control window if the software has been installed and the appropriate licenses have been obtained.

If you do not see an appropriate entry, you have not included DSP support in the RealView Debugger installation options. Choose a Custom installation to install DSP support if you want to use the Oak or TeakLite DSP or the Motorola M56621 DSP. If you do see the appropriate entry in the Connection Control window but cannot connect, check your licenses.

See the chapter describing DSP support in *RealView Debugger v1.7 Extensions User Guide* for details on using DSPs with RealView Debugger.
4.5 Working with JTAG files

Multi-ICE direct connect is an On-Chip Debugging (OCD) based emulator that uses JTAG files to define the devices on the JTAG scan chain and their order. This information might be supplied by the manufacturer or might be configured after installation. RealView Debugger detects the JTAG files and uses them to complete the configuration settings. This enables the debugger to access emulator targets on the local host for each supported processor.

Each emulation is specified in a .jtg file named after the supported processor, for example arm.jtg or arm_oak.jtg. By default, supported .jtg files are stored in the default settings directory \etc when you install RealView Debugger.

This section describes:
- Editing JTAG files
- Configuring a new DSP connection on page 4-39.

4.5.1 Editing JTAG files

To access the .jtg file editor for a Multi-ICE direct connect connection:

1. Start RealView Debugger without connecting to a target.
2. Right-click on the chosen entry in the Connection Control window to display the context menu, shown in Figure 4-19.

3. Select Connection Properties... to display the Connection Properties window.
4. Right-click on the required .jtg file and select Edit Configuration-File Contents... from the context menu, shown in Figure 4-20 on page 4-37.
This displays the Device JTAG-File Editor dialog box, shown in Figure 4-21.

Use this dialog box to amend the current device list, or to add new devices to the stored scan chain. The dialog box contains the following controls:

Name: Enter the name of a new device to be added to the configuration. This name identifies the new processor in the Connection Control window.
Type: Click on the down arrow to see a list of predefined processor types, for example ARM.

--- Note ---
The choices displayed in the drop-down list are determined by the vehicle defined for this connection, as set by the Connect_with/Manufacturer (see Appendix A Configuration Settings Reference).

On Connect:
Click on the down arrow to see a list of processor reset options when making a connection to the chosen device, for example Reset.

Bypass: This is used to ignore a particular TAP controller, preventing the debugger from connecting to it. Insert in the text field the number of bits in the Scan Chain Select Register (SCSR) for the bypassed device. For ARM CPUs this depends on CPU model, but is normally either 4 or 5.
The bypass option is not available for some vehicle types.

Extra: If your configuration file contains non-generic, target-specific entries, use this field to specify these parameters.

Create New When you have entered the device details, click here to add the new device to the top of the display list.

Device List This shows all the devices currently configured in the JTAG file. As you create a new device it is added to the top of the list.
The list is ordered as shown in Figure 4-22. That is, the top list entry corresponds to the device that has its Test Data In (TDI) connected to the host interface Test Data Out (TDO) pin and the bottom list entry has its TDO connected to the TDI pin of the host interface.

![Figure 4-22 JTAG chain ordering](image-url)
Move Up  Use this to change the order of entries in the display list. Highlight an entry and click **Move Up** to move it up towards the top of the list by one place.

Move Down Use this to change the order of entries in the display list. Highlight an entry and click **Move Down** to move it down towards the bottom of the list by one place.

Copy  Click to copy an existing entry and add it to the device list.

Edit  Click to edit a device definition and so rename it in the device list. Editing a device also means that you can change the text given in the **Description** column in the Connection Control window.

There is a limit on the number of characters that can be displayed in the Description column.

Remove  Click to remove a highlighted entry from the device list.

File: Use this data field to specify the full pathname to the .jtg file that defines the device list.

OK  Click to confirm your entries and so update the specified .jtg file. This closes the Device JTAG-File Editor dialog box.

Cancel  Click to close the Device JTAG-File Editor dialog box without making any changes to the device list.

Help  Click to get online help text about this dialog box.

**Viewing changes**

You must save the amended board file entries to update the contents of the Connection Control window. After confirming entries in the Device JTAG-File Editor dialog box, select **File → Save and Close** to close the Connection Properties window.

If the chosen entry, for example **ARM_MICE**, was expanded in the Connection Control window, saving the updated board file settings collapses the second-level entry.

Expand the entry to see the new or updated devices available for connection.

### 4.5.2 Configuring a new DSP connection

This example defines a new DSP connection. You can do this by creating a new connection entry or by copying an existing entry.
Note
If you copy a connection entry, certain configuration details are not updated until you close the Connection Properties window. This might mean that some asterisks are not immediately visible in the copy.

This example creates a new Oak DSP connection. The example assumes that a correctly configured .jtg file exists for the new target and this has been saved in the default settings directory \etc.

Note
You can use RealView ICE to connect to a target that incorporates DSP hardware with a suitable JTAG configuration file. For example, suppose that your target contains an ARM core and a DSP core. You can use RealView ICE to connect to the ARM core if the JTAG configuration file for the SoC specifies that the DSP TAP controller is ignored, that is, by setting it to bypass. Doing this makes the DSP invisible to the debugger but enables you to connect to the ARM core.

To define the new target:

1. Select File → Connection → Connection Properties... to display the Connection Properties window.
2. Right-click on the ...vdebug.brd entry, in the left pane.
3. Select Make New Group... from the context menu, shown in Figure 4-23.

Figure 4-23 Configuring a new target
4. This displays the Group Type/Name selector dialog box, shown in Figure 4-24 on page 4-41.

![Group Type/Name selector dialog box](image)

Figure 4-24 Specifying a CONNECTION group

Leave the type of the new entry unchanged as CONNECTION, but replace the default name new with a meaningful name for the new entry, for example New_OAK_DSP. This can be a descriptive name or the name of the new .jtg file that you are going to select, but without the .jtg extension.

5. Click OK to confirm your settings and to close the Group Type/Name selector dialog box.

The new entry appears in the left pane of the Connection Properties window. It is automatically selected, and its details are displayed in the right pane. These details are the default for a new CONNECTION and you must change at least the Connect_with/Manufacturer, the Configuration filename and target Description.

6. In the right pane of the Connection Properties window, right-click on the Configuration entry and select Edit as Filename from the context menu.

The Enter New Filename dialog box is displayed to enable you to locate the required .jtg file, for example \etc\New_OAK_DSP.jtg.

7. Click Save to confirm your entries and to close the Enter New Filename dialog box.

The new pathname is displayed in the right pane.

8. In the right pane of the Connection Properties window, right-click on the Description field, and select Edit Value from the context menu.

Type New_Oak_DSP in the entry area and press Enter.

This is the description displayed in the Connection Control window and Connection Properties window to identify the new target.

9. In the right pane of the Connection Properties window, right-click on the Connect_with entry and select Explore from the context menu.
10. In the right pane of the Connection Properties window, right-click on the Manufacturer entry and select the required connection type from the context menu, for example **ARM-ARM-PP**.

   If you do not specify this setting, the new connection appears in the Connection Control window but, when you try to connect, RealView Debugger prompts for the connection type.

11. Select **File → Save and Close** to save your changes and close the Connection Properties window.

   Your new target is now displayed in the Connection Control window.

--- **Note** ---

RealView Debugger DSP support is separately licensed. You must obtain a license from your ARM distributor to use this feature and connect to the new target.
Chapter 5
Working with Remote Targets

This chapter describes how RealView® Debugger handles remote connections. It includes information on the board file group RVBROKER, and explains how connections to different targets are configured to enable network access. It contains the following sections:

• Working with RealView Network Broker on page 5-2
• Connecting to a remote host simulator on page 5-5
• Connecting to a remote host emulator on page 5-10
• Using a hosts file on page 5-15
• Disconnecting remote connections on page 5-21.
5.1 Working with RealView Network Broker

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

For more details on using RealView Connection Broker see:
- the chapter describing getting started in RealView Debugger v1.7 User Guide.
- Chapter 4 Configuring Custom Connections.

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

This section describes how to work with remote connection entries. It contains the following sections:
- Accessing remote targets
- Specifying a remote host simulator on page 5-3
- Using the examples on page 5-4.

5.1.1 Accessing remote targets

RealView Debugger enables you to debug software running on targets connected to remote hosts, that is where hardware is connected to workstations accessed across the network. To do this, you must use the appropriate configuration dialog to locate the target:

- If you are using RealView® ICE, enter the address of the remote unit in the RVConfig dialog box. See Configuring a RealView ICE interface unit on page 4-11 for details.
  If you do not know these details, use the RealView ICE Config IP software to scan for all available units on your network. See the chapter describing configuring RealView ICE networking in RealView ICE User Guide for details on how to do this.
- If you are using Multi-ICE® on Windows, click Another computer... on the Multi-ICE DLL configuration dialog box to locate all workstations on your network that are running the Multi-ICE server software. See Configuring ARM Multi-ICE on page 4-22 for details.
RealView Network Broker enables you to connect to remote simulators and OCD-based emulators such as Multi-ICE direct connect. To do this, target configuration files are held on the local workstation and pushed across the network connection to the remote host. RealView Network Broker must be running on the remote workstation to provide access to the local simulators and emulators.

--- Note ---

Multi-ICE direct connect is the only way to connect to Multi-ICE where you are working on Sun Solaris or Red Hat Linux.

5.1.2 Specifying a remote host simulator

To access a remote host simulator or emulator using RealView Network Broker you must use the Connection Properties window to define the location of the remote workstation in your connection entries and target configuration settings in your board file, shown in Figure 5-1.

There are two ways to do this:

**RVBROKER**

This entry specifies the location of a host that supports simulators. The Connection Control window uses this entry to define remote targets. See Connecting to a remote host simulator on page 5-5 for details.

**rvbroker.brd**

This entry enables the inclusion of a special hosts file containing a list of all network nodes that can run, or are running, RealView Network Broker. See Using a hosts file on page 5-15 for details.
You can combine both types of entry in the same board file.

5.1.3 Using the examples

In the examples in the rest of this chapter, you are changing your board file. This is stored in your RealView Debugger home directory. Target configuration files might also be stored in this directory, for example .rvc files or .cnf files.

It is recommended that you back up this directory before starting the examples described in this chapter, so that you can restore your original configuration later. For details see:

- Configuration files on page 1-11 for instructions on making backups of your configuration
- Restoring your .brd file on page 3-57 for instructions on restoring a default configuration
- Troubleshooting on page 3-57 for instructions on recovering from an incorrectly configured debugger home directory, whether or not you have a backup.

If you are using a hosts file this might be stored in the default settings directory (\etc) or in your home directory. See Using a hosts file on page 5-15 for details.

--- Note ---

When you are following these examples, do not configure the board file when the debugger is connected to a target.

---

There are descriptions of the general layout and controls of the RealView Debugger settings windows, including the Connection Properties window, in the RealView Debugger online help topic Changing Settings. This chapter assumes you are familiar with the procedures described in this help topic.
5.2 Connecting to a remote host simulator

To connect to a remote host simulator, you must configure a new target description in the board file stored on the local workstation. This enables you to access a remote host simulator across the network.

First, you must start the remote connection broker, that is RealView Network Broker, on the remote workstation. See the chapter describing getting started in RealView Debugger v1.7 User Guide for details on how to do this.

Note
RealView Debugger does not have to be running on the remote workstation. Only RealView Network Broker is required to make the simulators and emulators visible across the network.

The rest of this section describes how to configure and connect to a remote simulator. It contains the following sections:

- Configuring the new target
- Connecting to the new target on page 5-8
- Viewing remote simulator connections on page 5-9.

5.2.1 Configuring the new target

To configure the new target on the local workstation:

1. Start RealView Debugger without connecting to a target.
2. Select File → Connection → Connection Properties... to display the Connection Properties window.
3. Right-click on the RVBROKER=localhost entry and select Make Copy... from the context menu, shown in Figure 5-2 on page 5-6.
Working with Remote Targets

4. Use the Group Type/Name selector dialog box, shown in Figure 5-3, to name the new connection, for example Remote_Debug.

5. Click OK. The new entry, RVBROKER=Remote_Debug, is added to the List of Entries in the Connection Properties window.

6. Select File → Save to save your changes. The new target is added to the Connection Properties window and the right pane is updated with the contents of the new group, shown in Figure 5-4 on page 5-7.
7. Right-click on Remote in the right pane, shown in Figure 5-4, and select Explore from the context menu.

8. Right-click on Hostname in the right pane to display the context menu. Select Edit Value and enter the hostname of the remote workstation, for example Hostname=armpc41, shown in Figure 5-5.

Specify the full domain name where necessary, for example Hostname=armpc41.ournet.arm.com. You can use the IP address, for example Hostname=192.168.2.212.

It is not necessary to specify the host name of the remote workstation if the RVBROKER group name is also the host name of the remote workstation on your network, for example RVBROKER=PC41. Where the Hostname entry is filled in, this is used and the group name is irrelevant.

9. Right-click on Description in the right pane, shown in Figure 5-4, to display the context menu. Select Edit Value and enter a short description for the new connection, for example Remote sim on network test pc 41.
10. Select **File → Save and Close** to save your changes and close the Connection Properties window.

You can create multiple RVBROKER groups if required. Use unique names to identify the different remote hosts.

### 5.2.2 Connecting to the new target

To connect to the remote target.

1. Display the Connection Control window to see the new remote connection, shown in Figure 5-6.

![Remote target in the Connection Control window](image)

2. Click on the icon next to the new entry, Remote_Debug, to expand it.

   This shows the available simulators made visible by the remote connection broker.

   This entry is empty if there are no remote simulators available on the specified workstation.

3. Click on the entry under the new Connection Broker entry to connect to the target.

   RealView Debugger retrieves information specific to the target.
5.2.3 Viewing remote simulator connections

When you connect to a remote target, a new entry appears in the Connection Control window, for example RSimARM_1, as shown in Figure 5-7.

![Connection Control window](image)

Figure 5-7 Remote connections in the Connection Control window

Here we can see two connections to remote RealView ARMulator® ISS targets. In this example, the IP address has been used to specify the host workstation. Other remote connections are visible depending on the type of installation at the remote workstation. Where you have the appropriate licenses, you can connect to these simulators in the usual way.

**Note**

Ending a debugging session on the remote workstation, and closing down RealView Debugger, does not terminate the remote connection broker. RealView Network Broker must be shut down explicitly when it is no longer required.
5.3 Connecting to a remote host emulator

RealView Debugger enables you to connect to OCD-based emulators running on remote hosts across a network. To do this you must:

- configure the target description on the remote workstation so that it is visible to users across the network
- create a new CONNECTION entry in the board file stored on the local workstation and use this to specify the hostname of the remote workstation.

The rest of this section describes how to configure and connect to a remote emulator. It contains the following sections:

- Configuring the remote target
- Configuring the new target on page 5-11
- Connecting to the new target on page 5-13.

Note

When installing RealView Debugger, you must choose the Custom option and install support for DSP. Do this to ensure that the required JTAG files are available to enable connection using Multi-ICE direct connect.

5.3.1 Configuring the remote target

To configure the target on the remote workstation:

1. Start RealView Debugger without connecting to a target.
2. Select File → Connection → Connection Properties... to display the Connection Properties window.
3. In the Connection Properties window, expand the connection that you are using:
   a. (*.rbe) ARM RDI Configuration Entries
   b. ...\multiice.rbe
4. Click on CONNECTION=Multi-ICE so that it is highlighted with a red border. The right pane displays the current settings. Configure the following entries:
   - Expand Connect_with and set Manufacturer=ARM-A-RR.
   - Set Configuration=multiice.cnf.
   - Set Shared=True.
   - Link any required board groups, for example, right-click on BoardChip_name and specify a .bcd file. Link other definitions as required to the CONNECTION.
5. Select **File → Save and Close** to save your changes and close the Connection Properties window.

6. Start the remote connection broker, that is RealView Network Broker. See the chapter describing getting started in *RealView Debugger v1.7 User Guide* for details on how to do this.

7. Exit RealView Debugger.

8. Ensure that the Multi-ICE server software is not running.

--- **Note**

RealView Debugger does not have to be running on the remote workstation. Only RealView Network Broker is required to make the shared configuration details visible across the network.

### 5.3.2 Configuring the new target

To configure the new target on the local workstation:

1. Start RealView Debugger without connecting to a target.

2. Select **File → Connection → Connection Properties...** to display the Connection Properties window.

3. Right-click on the entry .../rvdebug.brd in the left pane.

4. Select **Make New Group...** to display the Group Type/Name selector dialog, shown in Figure 5-3 on page 5-6.

5. Select the type of group you want to use, that is **CONNECTION**.

6. In the Group Name data field change the name from `new` to something suitable for your target, using only alphanumeric characters, underscore `_`, and dash `-`, for example `Remote_MICE`.

7. Click **OK**. The new entry, `CONNECTION=Remote_MICE`, is added to the List of Entries in the Connection Properties window.

8. Select **File → Save Changes** to save your changes. The new target is added to the Connection Properties window and the right pane is updated with the contents of the new group, shown in Figure 5-8 on page 5-12.
Working with Remote Targets

9. Right-click on Connect with in the right pane, shown in Figure 5-8, and select Explore from the context menu.

10. Right-click on the Manufacturer entry to display the context menu. Select ARM-ARM-PP to specify Multi-ICE direct connect.

11. Right-click on the Configuration entry, shown in Figure 5-8, and select Edit as Filename from the context menu.

   The Enter New Filename dialog box is displayed to enable you to locate the required JTAG file. If, for example, you are connecting to a single ARM® core, set this to \etc\arm.jtg.

12. Click Save to confirm your entries and close the Enter New Filename dialog box.

   The new pathname is displayed in the right pane.

13. Edit the Remote settings values page in the new group, shown in Figure 5-8, to enter the hostname or IP address of the remote workstation, shown in Figure 5-9 on page 5-13.

---

Figure 5-8 Displaying the new Remote_MICE connection properties
14. Right-click on Description in the right pane, shown in Figure 5-8 on page 5-12, to display the context menu. Select Edit Value and enter a short description for the new connection, for example Remote emulator on network test pc 41.

15. Select File → Save and Close to save your changes and close the Connection Properties window.

5.3.3 Connecting to the new target

To connect to the remote target.

1. Display the Connection Control window to see the new remote connection, shown in Figure 5-10.

2. Click on the icon next to the new entry, Remote_MICE, to expand it.
   This shows the available emulators made visible by the remote connection broker. This entry is empty if there are no remote emulators available on the specified workstation.
3. Click on the entry under the new `Remote_MICE` entry to connect to the target. RealView Debugger retrieves information specific to the target.

--- Note ---

Ending a debugging session on the remote workstation, and closing down RealView Debugger, does not terminate the remote connection broker. RealView Network Broker must be shut down explicitly when it is no longer required.
5.4 Using a hosts file

The `rvbroker.brd` entry in the board file enables RealView Debugger to include a special hosts file containing a list of all network nodes that can run or are running RealView Connection Broker. This enables multiple users to access simulators, emulators, and EVMs anywhere on the network by sharing this single file.

If you are using a hosts file, this is the same as adding `RVBROKER=` entries to your board file to manage remote connections across your network. However, the connection details are contained in the `rvbroker.brd` file and not in the `rvdebug.brd` file.

When RealView Debugger reads a board file, for example `rvdebug.brd`, it searches for an `rvbroker.brd` file to use. The debugger searches:

1. in the default settings directory, `install_directory\RVD\Core\...\etc`
2. your home directory
3. the current working directory.

Where you are using a local hosts file, it must be installed in your home directory. Install the hosts file in the default settings directory (`\etc`) for shared access. The search order defines the precedence when using multiple files in different locations.

This section describes how to create and use a hosts file. It includes the following subsections:

- Creating a local hosts file
- Configuring a local hosts file on page 5-16
- Connecting using a local hosts file on page 5-18
- Using multiple connections in a local hosts file on page 5-18
- Using a network hosts file on page 5-19.

5.4.1 Creating a local hosts file

To create a hosts file:

1. Use a text editor to create a file named `rvbroker.brd` containing a single line:
   ```bash
   [RVBROKER=Remote_Hosts]
   ```

   Save this file in your home directory, for example `home\my_user_name\rvbroker.brd`.
5.4.2 Configuring a local hosts file

To configure a hosts file:

1. Start RealView Debugger without connecting to a target.
2. Display the Connection Control window where the new connection entry is visible, shown in Figure 5-11.

![Figure 5-11 Hosts file in the Connection Control window](image)

RealView Debugger locates the rvbroker.brd file in your home directory, or in \etc, and includes the configuration details in the board file, ready for editing.

3. Right-click on the Remote_Hosts entry and select Connection Properties... from the context menu to display the Connection Properties window, shown in Figure 5-12.

![Figure 5-12 Configuring a hosts file](image)
4. Edit the RVBROKER=Remote_Hosts group as follows:
   
a. Edit the Description settings value in the right pane, to identify the remote connection, for example Remote to test board on pc 64.

   b. Ensure that the Disabled settings value, in the right pane, is set to False, the default.

   c. Expand the Remote group, in the right pane, to display the contents. Edit the Remote settings values page to enter the network address of the remote workstation, for example Hostname=armpc64.ournet.arm.com, shown in the example in Figure 5-9 on page 5-13.

5. Select File → Save and Close to save your changes and close the Connection Properties window.

The hosts file, rvbroker.brd, is automatically updated with the new configuration information. You can view the amended file using a text editor, for example:

```
[RVBROKER=Remote_Hosts]
remote.hostname="armpc64.ournet.arm.com"
description="Remote to test board on PC 64"
```

You can also use the network IP address to specify the remote target, for example:

```
[RVBROKER=Remote_Hosts]
remote.hostname="192.168.2.148"
description="Remote to test board on PC 64"
```

**Custom entries in the hosts file**

Default board file entries are not shown in the hosts file. For example, the setting Disabled=False (shown in Figure 5-12 on page 5-16) is the default setting for the new RVBROKER entry and so there is no corresponding entry in the hosts file. However, if you edit this setting so that it becomes *Disabled=False*, an entry is created in the hosts file, for example:

```
[RVBROKER=Remote_Hosts]
remote.hostname="192.168.2.148"
description="Remote to test board on PC 64"
disabled=False
```

You can specify entries in the hosts file even where these are the default settings for the new connection.
5.4.3 Connecting using a local hosts file

To connect to a remote target using the hosts file:

1. Display the Connection Control window to see the new remote connection. An icon identifies that this connection uses a hosts file to store configuration details, shown in Figure 5-13.

![Figure 5-13 Remote target in the Connection Control window](image)

2. Expand the new Remote_Hosts entry to view the available target connections made visible from the remote connection broker.

3. Click on the new entry to make the remote connection.

5.4.4 Using multiple connections in a local hosts file

You can copy the RVBROKER=Remote_Hosts entry to make a list of hosts entries. Configure each one to point to a specified network node.

The hosts file is automatically updated with the new configuration information. You can view the amended file using a text editor, for example:

```plaintext
[RVBROKER=Remote_Hosts]
remote.hostname="armpc64.ournet.arm.com"
description="Remote to test board on pc 64"
disabled=False
[RVBROKER=Remote_Hosts_2]
remote.hostname="armpc106.ournet.arm.com"
description="Remote to test board on pc 106"
disabled=False
```

Each entry in the hosts file appears as an available connection in the Connection Control window, shown in Figure 5-14 on page 5-19.
5.4.5 Using a network hosts file

If required, a hosts file can be installed in a network location, for shared access by other network users. You create the network hosts file in the same way as a local hosts file and save it in a shared location on your network, for example \ournet\devel\rvd\shared\rvbroker.brd. The rvbroker.brd entry in your board file is edited to specify the full network pathname.

To configure a network hosts file:

1. Use a text editor to create a network hosts file named rvbroker.brd containing:

   ```
   [RVBROKER=Network_Hosts]
   remote.hostname=armpc68.ournet.arm.com
   description="Remote to shared network test board on pc 68"
   disabled=False
   [RVBROKER=Network_Hosts_2]
   remote.hostname=armpc108.ournet.arm.com
   description="Remote to shared network test EVM on pc 108"
   disabled=False
   ```

   Save this file in your shared network location, for example \ournet\devel\rvd\shared\rvbroker.brd.

2. Start RealView Debugger without connecting to a target.

3. RealView Debugger locates the rvbroker.brd file in your home directory, or in \etc, and includes the configuration details in the Connection Control window, ready for making a connection, shown in Figure 5-14.

4. Select File → Connection → Connection Properties... to display the Connection Properties window.
5. To include the connection details given in the network hosts file, you must add a hot link from the board file:
   a. Right-click on the top-most entry ...\rvdebug.brd.
   b. Select **Add Hot Link to a File...** from the context menu.
   c. Click **OK** to accept the warning message and continue.
   d. Locate the network hosts file in the Select file to link to dialog box.
   e. Click Open to confirm your choice.

6. Select **File → Save and Close** to save the new settings and close the Connection Properties window.

The board file is automatically updated to include the specified network hosts file and the Connection Control window displays the remote hosts as specified in the local hosts file and the remote network hosts file, shown in Figure 5-15.

![Figure 5-15 Network hosts file in the Connection Control window](image)

Now other users can access the shared hosts file on your network by amending their board files in the same way.
5.5 Disconnecting remote connections

You can use the CLI command `DISCONNECT` to disconnect from a remote connection using either the connection id or the target name, for example:

disconnect 9
disconnect @RSimARM_12

where the name `RSimARM_12` identifies the remote target in the Connection Control window.

See the description of the `DISCONNECT` command in *RealView Debugger v1.7 Command Line Reference Guide* for details on disconnecting this way.
Appendix A
Configuration Settings Reference

This appendix contains reference details about board file entries that define target configurations and custom connections. It contains the following sections:

- About this appendix on page A-2
- Target configuration and connections on page A-3
- Generic groups and settings on page A-5
- Target configuration reference on page A-8
A.1 About this appendix

You configure the way that RealView® Debugger connects to, and interacts with, your debug target using a board file. The default board file, called rvdebug.brd, is set up when you first install RealView Debugger.

The contents of the Connection Control window are defined by elements of the board file. To change target configuration, or to add new connections, you change the entries in the board file and this modifies the elements displayed in the Connection Control window. RealView Debugger provides a GUI interface, the Connection Properties window, to make these changes. See Connection Properties window on page 1-5 for details.

To follow this appendix, you must display the Connection Properties window, either:

- Select File → Connection → Connection Properties....
- Select an entry in the Connection Control window. Right-click and select Connection Properties... from the context menu.
  Use this method to open the Connection Control window at the parent group for the chosen connection, target or processor.

This appendix describes the settings and groups of settings that appear in the board file. It assumes that you are using the RealView Debugger base product that includes built-in configuration files to enable you to make a connection. If you have changed these files, or created new target configuration files of your own, your window looks different.
A.2 Target configuration and connections

RealView Debugger makes a distinction between target configuration, and how a target is accessed, target connection. Both are configured using different types of entry in the board file.

A.2.1 Types of entry

A CONNECTION group is normally used to specify connection details, how the target is accessed. This includes the nature and addresses of the hardware interface, for example the port name of the JTAG interface that is connected to the target. This information is described in the Connect_with block of the board file and in the file associated with the Configuration setting.

--- Note ---

Occasionally, you can use a DEVICE group in place of a CONNECTION group.

Within a specific CONNECTION, one or more BoardChip_name entries are used to associate the connection with a BOARD, CHIP, or COMPONENT description that defines peripherals and memory maps.

It is recommended that the descriptions of the target are only defined in BOARD or CHIP definitions, and that these descriptions are stored in .bcd files. For example, the definition of the registers and peripherals of the ARM® Integrator/AP motherboard is stored in the file AP.bcd. Figure A-1 shows how these groups are linked.

--- Figure A-1 How connections, boards, and chips fit together ---
Therefore, the board file consists of the following types of entry:

- Connection entries using the following groups:
  - CONNECTION or DEVICE
  - RVBROKER.

- Configuration entries using the following groups:
  - BOARD
  - CHIP
  - COMPONENT.

This appendix describes the groups and individual settings that appear in the board file. It assumes that you are familiar with the contents of the Connection Properties window as described in *Connection Properties window* on page 1-5.
A.3 Generic groups and settings

There are several board file entries that are common to many of the settings groups, shown in the example `CONNECTION=RealView ICE` entry in Figure A-2.

These settings are found in `CONNECTION` groups. In some cases, some or all of these settings are also found in other groups, for example `BOARD=AP` in the `AP.bcde` entry. This section describes these generic groups and settings:

- **Connect_with**
  Specifies the vehicle making the connection. It includes the settings values:
  - **Manufacturer**
    The name and the type of the connection. Right-click to see a list of available connection types, but you also require an appropriate license and the hardware to use them.
  - **IOdevice**
    This field contains additional information about the target hardware. Not available in the current release of RealView Debugger.
  - **Speed**
    You can set the emulation speed for some emulators. Not available in the current release of RealView Debugger.
### Remote
This specifies how remote connections are configured. It includes the settings values:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hostname</strong></td>
<td>The hostname, or IP address, of the remote workstation.</td>
</tr>
<tr>
<td><strong>Port</strong></td>
<td>Specifies that a non-default TCP/IP port is used to make the connection to the remote workstation.</td>
</tr>
</tbody>
</table>

### Advanced Information
Provides ETV target configuration information about the debug target.

This feature is described in detail in *The Advanced Information block* on page A-8.

### Configuration
Specifies a named configuration file.

The default is to search for the `.jtg` file with the same name as this group (after the `=` sign). The name in this field does not have to be the same as the name of this group. If the file specified here is a full pathname, then only that location is used. Otherwise, it is searched for.

When working with RealView ICE targets, `.jtg` files are replaced by `.rvc` configuration files. When working with RDI targets, `.jtg` files are replaced by `.cnf` configuration files. When working with custom simulator connections, `.jtg` files are replaced by `.auc` configuration files.

### Auto_connect
If set to `True`, this expands the list of devices on the debug target. The list of devices comes from the `.jtg` file specified by the `BOARD` or `CHIP` group. The list of devices is shown in the Connection Control window. You can then connect to the devices with a single click.

### Pre_connect
Forces an order for device connection. When you connect to a device within the `.jtg` file, this ensures that one or more specific devices are connected first regardless of which device is selected for connection. This enables pre-setup of the specific devices to guarantee correct operation, such as initializations. You can specify the device(s) to connect to first by name, by processor name, or by processor type, as used in `.cnf` files that contain target configuration settings.

### Description
Description of what board, processor, or emulator this is for.
**Project**

Opens one or more projects automatically when connecting to this board. If more than one device is in the scan chain and they are the same processor type, you must set the Specific_device field of the project to bind the project to the correct device(s). If the devices are different processor types, this is not necessary.

See RealView Debugger v1.7 Project Management User Guide for details on working with projects in RealView Debugger.

**Disabled**

Disables this entry so that it is grayed out in the Connection Control window.

**Shared**

Enables the sharing of target configurations for remote connections.

**BoardChip_name**

Refers to the BOARD, CHIP, or COMPONENT group this connection is derived from, using its name. If the group has more than one name separated by a slash, such as ID/name, any of them can be used. If not specified, the name of this group is used to match a board or chip.

**Family_select**

Ensures the correct family member is used (for example, for memory mapping, and registers) when the silicon ID is ambiguous. Some chip families do not use different silicon IDs for different members of the family, and this field enables you to specify which you are using. Specify the family member using one of these formats:

- `name=family_name`
  This enables you to specify the name of the device. Use either the name defined in the .jtg file or the processor name. This is used when multiple chips are housed on the same target but from different families.

- `family_name`
  Choose from the preconfigured list.

- `silicon_id`
  Can be expressed as num.num.num.num, for example 15.255.15.15, or as a value, for example 41029401.

For information about connection configuration, including the CONNECTION and DEVICE entries, see Custom connection reference on page A-26.
A.4 Target configuration reference

This section describes in detail the target configuration entries supported by RealView Debugger. It contains the following sections:

- The Advanced_Information block
- BOARD, CHIP, and COMPONENT groups on page A-25.

A.4.1 The Advanced_Information block

The Advanced_Information block enables you to provide ETV information about the debug target, for example extended memory mapping, mapped registers, and peripherals. Because more than one device might be present, and each might have different details, you can create more than one Advanced_Information block. The base group is called Default and is used if you provide no other information. Entry names can be the same as processor or device names in configuration files, for example in the .rvc file, or .jtg file, for the connection. These apply more specifically to matching devices.

Advanced information settings can be nested so that one might refer to another which might refer to another. These references cause the information to be concatenated. References are made to board and chip definitions.

The Default group in an Advanced_Information block contains:

- Application_Load on page A-9
- Memory_block on page A-9
- Map_rule on page A-13
- Register_enum on page A-14
- Register on page A-14
- Concat_Register on page A-15
- Peripherals on page A-16
- Register_Window on page A-17
- ARM_config on page A-18
- Logic_Analyzer on page A-19
- Cross_trigger on page A-20
- RTOS on page A-21
- Pre_connect on page A-22
- Commands on page A-22
- Connect_mode on page A-22
- Disconnect_mode on page A-23
- Id_chip on page A-24
- Id_match on page A-24
• Chip_name on page A-24
• Endianess on page A-25.

It is recommended that you copy the Default group to create your own, named, groups. Do not delete the Default group so that it is available for future configuration tasks.

Application_Load

Use the Application_Load group to change the way that an executable image is loaded into target memory. The default is to write the memory using the emulator or EVM board. Settings in this group can be used to override the default and so disable all image load, for pure ROM or EPROM systems, or to run an external program to do the load.

The Application_Load group contains the following settings:

Load_using  Specifies how to perform the load.

Load_command

This defines a shell command run to perform the load. The command might contain $ variables which are substituted by RealView Debugger before calling. The possible $ variables are:

$D directory of the application
$P full path of the application
$F filename of the application
$N name of the application without the extension.

If the command starts with an exclamation mark (!), the return value of the shell command is not used to stop the load, otherwise a non-0 return aborts the load. In all cases, the output of the command is shown in the Log tab in the Output pane.

Load_set_pc  This controls how the PC is initialized during an image load. The default is to set the PC to the entry point if an entry point is defined and symbols are loaded and this is not an appended load (it is replace or new). This enables you to disable setting the PC under any situation, or to set it specifically to address 0.

Memory_block

Entries in the Memory_block group are used to build up fixed, enabled, or based memory regions. A fixed block is one that refers to memory that is always enabled, always at the same place, and always the same size. An enabled memory block is enabled or disabled
by a register value (see Map_rule on page A-13). A based memory block has its start or length adjusted or set by a register value. This value might be added to an offset or might itself be the required value.

This contains the following settings:

**Attributes** Additional attributes can be specified for the memory. These are used by simulator models to guide the debugger when accessing this block of memory. The following settings are available:

- **Internal** This memory is internal to the processor core and not treated as external. This affects wait-state timings and other factors.

**Access-rule**

The access rule information is only used by simulators to control timing issues. It is noted if a link command file is generated.

**Access_size**

Use this field to control how RealView Debugger accesses memory internally.

By default, this is set to 0, to indicate the default memory access size for the processor, for example, byte-size for ARM processors.

Specify a value greater than 0 to set access size in bytes, for example 1 or 2. If you do this, you must also specify a size for Memory_block, otherwise RealView Debugger displays an error message.

For external memory with only byte-wide or halfword-wide access enabled, this can be used to ensure proper access to the memory. Depending on the processor, this might have no effect.

**Volatile**

Set to True if access destroys contents or the contents change in response to external events.

**Shared**

This field indicates if the memory is shared with other processors. If it is, it also indicates if directly shared (accessed directly using the bus) or indirectly shared (using the host workstation port of this processor). If this is set, this field indicates what other processors or devices see this memory.

**Shared_id**

This field contains a number that identifies this memory block. You must use the same number for each device when referring to it. RealView Debugger can then correctly update all device views when this memory is modified.
Register_Pos_Len

Used when one or two memory-mapped registers are used to set the base address and length of the memory block, such as for cross-bar switches, and chip-selects. These are not used for enables which are set using map rules (see Map_rule on page A-13). The following settings are available:

Register_base

Enables you to specify a memory block position based on a memory-mapped register. The value of the register is added to the start field to construct the block start address. It can be masked and scaled (multiplied or divided) first.

Base_mask

Mask to apply to register.

Base_scale

Use this to change the value of the register contents, after masking, to define the actual base. If the number is positive, it is multiplied against the register content. If the number is negative, it is divided from the register content. For example, a byte register might select the 64Kbyte region to map to. If the scale is 0x10000 (64K), then register values of 0, 1, 2, 3, ... each select a 64Kbyte region. If the selector occupies part of a register, the mask is applied to select only the selector portion and the scaling value itself might be scaled. Using the example above, if the byte selector portion is the upper byte of a register, the scale value is 0x10000/0x100=0x100 (256). So, mask with 0xFF00 and multiply by 256 to get a 64Kbyte selection.

Register_length

Enables a block of memory to be sized by a register. This is commonly used in multiprocessor shared memory systems. The content of the register is added to the specified length to compute the block length.

Len_mask

Mask to apply to register.

Len_scale  Used like Base_scale.

Len_table

Enables table indexing for the length. The length register is masked and scaled and then used as an index in a table of values. The last value is used if the scaled register value is too large. The table value is added to the length field of the block.
**Update_rule**

Indicates how often to check the register to see if the mapping has changed. For cases where the mapping is set by jumpers, which read as registers, it must be inspected only when first connecting to the device. If the program changes it, it must be tested on each stop. Valid values are:

- **init_time**: Test when connecting to device.
- **update_init**: Test on connect and when register changes.
- **stop_update**: Test on connect, change, and execution stop.

**Start**

The base address of the block. If the block is mapped using a register, this is the offset from that register.

**Length**

The block length of this memory unit. If Length is set by a register, this must be 0 or the amount to add to that register.

**Type**

The type of memory is dependent on the device type. The default is to map to data space. Otherwise, a memory space can be specified. Valid values are:

- **default**
- **program**
- **data**
- **IO**.

**Access**

Indicates how the memory is to be treated. For simulators, this affects the target use of the memory. For hardware targets, this only affects how the debugger uses the memory and any generated linker command files. Right-click to see available access types.

**Wait_states**

Used with simulators to calculate the cycles used when accessing this memory. The default is based on the wait-state model used by the processor for external memory. This value is noted when link command files are generated from this data to enable careful positioning of sections to this memory.

**Flash_type**

Contains the name of a file containing the Flash programming code and information for this processor. Example files for selected ARM targets are provided in the `install_directory\RVD\Core\...\flash\examples` directory. Files are collected in subdirectories based on the target Flash device, for example `...\flash\examples\IntegratorAP`. These files have the file extension `.fme`. 
By using the routines in this file, RealView Debugger can erase, modify and verify the contents of Flash memory.

**Description**  
Description of memory space.

**Volatile**  
Enables you to define a memory block that is volatile on read (and so is marked specially in the Memory pane). The format is an offset from within this block (0 relative). A range can be specified, for example 0x10..0x20 or 0x40..+4. If not a range, it defines a single value.

**Map_rule**  
These entries control the enabling and disabling of memory blocks using target registers. You specify a register to be watched, and when the contents match a given value, a set of memory blocks is enabled. You can define several map rules, one for each of several memory blocks. The following settings are available:

**Register**  
This is the name of a memory-mapped target register that controls the visibility of a memory block. This register is read to determine the current mappings. You must define the register using the `Register` block (see `Register` on page A-14).

You are recommended to name the register itself instead of the bit fields within it when more than one bit field controls the mapping.

**Mask**  
This is ANDed with the contents of the register as described in `Value`.

**Value**  
This is compared with the register contents after the mask is added. The comparison is `(reg-value & mask) == value`. For example, if bit 3 being HIGH means the map is enabled, mask is 0x8 (1<<3) and value is also 0x8.

**On_equal**  
This contains the name of one or more memory blocks to enable when the value matches, or disable when it does not match. To replace one block with another, create one rule that tests for one value and another that tests for a different value.

**Update_rule**  
Indicates how often to check the register to see if the mapping has changed. For cases where the mapping is set by jumpers, which read as registers, it must be inspected only when first connecting to the device. If the program changes it, it must be tested on each stop. Valid values are:

- `init_time`: Test when connecting to device.
- `update_init`: Test on connect and when register changes.
- `stop_update`: Test on connect, change, and execution stop.
Register_enum

Enumerations can be used, instead of values, when a register is displayed in the Register pane. This setting enables you to define the names associated with different values. Names defined in this group are displayed in the Register pane, and can be used to switch the register.

Register bit fields are numbered 0, 1, 2,... regardless of their position in the register.

The following setting is available:

Names You can specify a list of names, either in the form name,name,name,... or in the form name=value,name=value,name=value,...

Register

This group enables you to define memory-mapped registers provided at the board or ASIC level. Each register is named and typed and can be subdivided into bit fields (any number of bits) which act as subregisters.

The Register group contains a Default group which contains a child group called Bit_fields which, in turn, contains the Default settings:

Position Bit position from 0 (LSbit).
Size Size in bits.
Signed Set to True if signed, otherwise set to False.
Enum Enumeration name to show values in the Register pane, derived from the Register_enum group.
Read_only Set to True if read-only (cannot modify).
Volatile Indicates that the register has side effects when it is read or written. A common read side effect is loss of data (when pulled from a UART, for example). A common write side effect is for the device to take some action on write (triggering a DMA, for example). This information is used in the Register pane. Right-click to see available options.
Gui_name Optional name for showing in Register pane.

Other settings in the Register entry, Default group, are:

Start The base address of the block. If the register is mapped using a memory block, this is the offset from that register (see Base).
Length  The block length of this memory unit. If Length is set by a register, this must be 0 or the amount to add to that register.

Base  This specifies how to interpret Start. If Base is Absolute, Start is the memory address of the register. Otherwise, Base can be set to the name of a memory block, and Start is an offset from the base address of that memory block.

Memory_type  The type of memory is dependent on the device type. The default is to map to data space. Otherwise, a memory space can be specified. Valid values are:
- default
- program
- data
- IO.

Type  Specifies how to interpret the value contained in the register. The type names are as in the C language.

Read_only  If set to True, the register is read-only and the debugger does not let you write to it. Otherwise, you can modify the value using the Register pane in the Code window and using CLI expressions.

Write_only  If set to True, the register cannot be read. The debugger does not attempt to query the hardware for the current value when the Register pane is displayed.

Volatile  If set to True, the register value can change without the debugger explicitly modifying it. For example, a hardware timer continues to count even when the processor is halted.

Enum  The name of a Register_enum block that maps a register value to a textual string describing the value.

Gui_name  The name of the register as it appears in the Register pane.

Concat_Register  You can define a concatenated register that is built up using specific bits from other registers. Concatenated registers are usually used only for memory mapping, but you can also use them for control and status. The suggested approach is to name two registers and then shift and mask them into the new register. If you want to concatenate parts from more than two registers, you can build them up in stages.
The following settings are available:

**Low_name**  Name of low register.

**Low_shift**  Amount to shift low register by (<0 for left shift).

**Low_mask**  Amount of low register to mask (after shift).

**High_name**  Name of high register.

**High_shift**  Amount to shift high register by (<0 for left shift).

**High_mask**  Amount of high register to mask (after shift).

**Length**  Length of register, in memory units.

**Type**  An explicit type for the register. If you do not specify the type, the default is the signed scalar C type based on the register size.

**Enum**  Enumeration name to show values to show values in the Register pane, derived from `Register_enum` group.

**Gui_name**  Optional name for showing in Register pane.

### Peripherals

This group enables you to define block peripherals so they can be mapped in memory, for display and control, and accessed for block data, when available. You define the peripheral in terms of the area of memory it occupies (for all its registers), and a breakdown of the registers used for access and control. The following settings are available:

**Access_Method**

This applies only when you can access blocks of data, and contains:

**Type**  Method used to extract data.

**Method_name**  Name of access method function if required.

**Start**  Buffer or DMA start address.

**Length**  Buffer or DMA length.

**Register**  Used to add memory mapped registers provided at the board or ASIC level. Each register is named and typed and can be subdivided into bit fields (any number of bits) which act as subregisters. See `Register` on page A-14 for details.

**Start**  Start address of first peripheral register.
Length  Block length.

Base  Controls how the start field is interpreted. The default is Absolute (from 0), but can be relative to a memory block (if the block is disabled, the peripheral is too).

Type  Basic type of the device. The available values are:
- serial
- parallel
- block
- network
- display
- other.

Description  Description of the device.

Register_Window

This entry contains a set of lines to show in the Register pane. The name of the block is the tab name used for the lines. Each line contains a list of mapped registers displayed in the Register pane, see Register_enum on page A-14, Register on page A-14, and Concat_Register on page A-15 for more details.

The format of a line is name, name, name, ... where each name is the name of a register or bit field. Be aware of the following:

- If the string starts with an equals sign, =, all the registers are shown as name=value in the window, else shown in table form (the name is above the value).
- If a line starts with an underscore character, _, the line shows as a comment label (non-active).
- If the line starts with an exclamation mark, !, it provides a description line for the tab.
- If the line starts with:
  $ the next line starts or ends an expansion block, controlled by + or -.
  $+  indicates a collapsed block
  $-  indicates a expanded block
  $$ this ends a previously opened block.
**ARM_config**

This group enables control of ARM processor settings used for ARM emulators, monitors, or simulators. These control features such as semihosting, vector catching, and memory control (for stack and heap assignment) which must be set or unset depending on the type of runtime you have linked into your application.

You can also set many of these at runtime using pseudo-registers. To do this, name the block `default` if it applies to all devices or give it the name of the scan chain device to which it applies.

The following settings are available:

**Stack Heap** The ARM tools automatically set the stack and heap based on the top of memory using semihosting. The following settings are available:

- **Stack_bottom** Bottom of stack (lowest address).
- **Stack_size** Size of stack in bytes.
- **Heap_base** Bottom of heap (lowest address).
- **Heap_size** Size of heap in bytes.

**Vectors** If `Vector_catch` is set to `True`, the fields within this group enable individual control over each vector.

The following settings are available:

- **Reset** Set this to catch Reset exceptions.
- **Undefined** Set this to catch undefined instruction exceptions.
- **SWI** Set this to catch Software Interrupts.
- **P_Abort** Set this to catch Prefetch abort exceptions.
- **D_Abort** Set this to catch Data abort exceptions.
- **Address** Set this to catch Address exceptions.
- **IRQ** Set this to catch normal interrupt exceptions.
- **FIQ** Set this to catch Fast Interrupt exceptions.
- **Error** Set this to catch errors.

**Semihosting** Enables programs to communicate with the host workstation. Semihosting operations supported include stack and heap assignment and console I/O (`printf` and `scanf` type calls). Semihosting is implemented using the SWI instruction. You can change the semihosting vector during debug using the `@semihost_vector` pseudo-register. You can also define a window number to display semihosting `printf` messages using `@semihost_window`. The window numbers match the `VOPEN` command numbers.
Contains:

- **Enabled**: Set this to enable semihosting.
- **Vector**: Address of SWI vector catch to use.
- **Arm_swi_num**: ARM SWI instruction for semihosting.
- **Thumb_swi_num**: Thumb® SWI instruction for semihosting.

### Armulator

Contains:

- **Clock_speed**: Clock speed in MHz as `num.num`.
- **Fpoint_emu**: True if floating point emulation.
- **Config_file**: The name of the configuration file.

### Top_memory

Enables the semihosting mechanism to return the top of stack and base of heap. If not defined here, the default for each tool is used (different for RealView ICE, Multi-ICE®, or Angel). Any defined value is set into each tool to force this address base. You can use explicit stack and heap sizes and locations below, but this might not be supported by all debug targets. You can also set this during a debugging session using the `@top_mem` pseudo-register.

### Vector_catch

Used to catch possible program errors by setting breakpoints on (or otherwise trapping) the vectors. The default is to catch error-type vectors but leave IRQ, FIQ and SWI alone. SWI is caught separately by semihosting if enabled. To use this, the vectors must be writable. These can also be set during debugging using the `@vector_catch` pseudo-register. In this case, each bit, starting with 1, represents the vectors from reset to FIQ.

### Properties

This enables free-form definition of the properties required by a vehicle (emulator or simulator). The form of the string is `name=value`, where `name` is the name for the property as defined by the vehicle and `value` is a numeric value in hex or decimal.

### Logic_Analyzer

This group is used to define settings for external trace analyzer hardware.

RealView Debugger currently supports:

- ARM-based trace targets
- the Oak and TeakLite DSPs
XScale™ onchip trace.

By default, RealView Debugger is automatically configured with tracing enabled for ARM targets using preset values. These settings are not used for non-ARM trace targets.

If you have set up a new ARM-based trace target, using a new CONNECTION group (as described in Creating new target descriptions on page 3-9) you must configure these settings to enable tracing:

- right-click on Vendor and select ARM
- right-click on Load_when and select connect.

The following settings are available:

**Vendor**
- Supported vendors.

**Machine**
- Name of analyzer or device.

**Config**
- Configuration file required to support your analyzer or device.

**Load_when**
- Defines when RealView Debugger enables tracing:
  - enabled on connection, with Load_when set to connect
  - enabled on image load, with Load_when set to image_load
  - enabled when a specified symbol is matched, with Load_when set to symbol_match
  - enabled when first trigger is specified, with Load_when set to first_use.

**Sym_match**
- Defines the symbol match to enable tracing.

For full details on tracing with RealView Debugger, see RealView Debugger v1.7 Extensions User Guide.

**Cross_trigger**

These settings control the cross-triggering of a stop command between multiple processors that are closely coupled in hardware. They specify whether stopping execution of one processor stops execution of other processors, due to a break or other stop condition:

- input triggering means that the processor is stopped by others
- output triggering means that the processor can stop others.

The available settings are:

**Trig_in_ena**
- List of commands to enable input triggering.
Trig_in_dis  List of commands to disable input triggering.
Trig_out_ena  List of commands to enable output triggering.
Trig_out_dis  List of commands to disable output triggering.

RTOS

This group enables automatic loading of RTOS and kernel awareness. This group enables you to configure an RTOS-enabled connection when you have no vendor-supplied .bcd file. When the RTOS is symbol-hooked, the RTOS plugin is only loaded when the RTOS, or its symbols, are loaded to the target. See the instructions from your vendor for proper setup. If supporting your own RTOS and kernel, use the method that best matches your DLL.

The following settings are available:

Vendor  This three letter value identifies the RTOS plugin, that is the *.dll file supplied by your vendor.

Load_when  Defines when RealView Debugger loads the RTOS plugin:
  •   load the plugin on connection, with Load_when set to connect
  •   wait until an RTOS image is loaded, with Load_when set to image_load.

The RTOS features of the debugger are not enabled until the plugin is loaded and has found the RTOS on your target.

Base_address

Defines a base address, overriding the default address used to locate the RTOS data structures. See your RTOS documentation for details.

Exit_Options

Defines how RTOS awareness is disabled. Use the context menu to specify the action to take when an image is unloaded or when you disconnect. You can also specify a prompt.

RSD  Controls whether RealView Debugger enables or disables RSD. This setting is only relevant if your debug target can support RSD.

System_Stop

Use this setting to specify how RealView Debugger responds to a processor stop request when running in RSD mode.
In some cases, it is important that the processor does not stop. This setting enables you to specify this behavior, use:

- **Never** to disable all actions that might stop the processor.
- **Prompt** to request confirmation before stopping the processor.
- **Don’t_prompt** to stop the processor. This is the default.

For full details on RTOS support with RealView Debugger, see *RealView Debugger v1.7 Extensions User Guide*.

**Pre_connect**

Forces an order for device connection. When you connect to a device within the .jtg file, this ensures that one or more specific devices are connected first no matter which you selected for connection. This enables pre-setup of the specific devices to guarantee correct operation, such as initializations. You can specify the device(s) to connect to first by name, by processor name, or by processor type, as used in .cnf files that contain target configuration settings.

**Commands**

This enables you to specify RealView Debugger commands to run after a connection is established. The most common example is **INCLUDE** to include commands from a file. The commands are run just after the connection is completed. If **Pre_connect** is set, and the pre-connected device is running this command, the command executes before the original device is connected.

**Connect_mode**

When you connect to a target, RealView Debugger attempts to establish the connection using the default connect mode, that is **No Reset and Stop** (see *Setting connect mode* on page 2-13 for details).

Before connecting, RealView Debugger checks to see if a user-defined connect mode has been specified by the Connect_mode setting in your board file, or in any .bcd file linked to the connection. If such a setting is found, it becomes the default connect mode for this connection.

--- **Note** ---

A blank entry in the top-level Connect_mode setting in the board file ensures that any setting in a linked Board/Chip definition file is used instead. This might be important if you are using .bcd files to enable RTOS awareness in RealView Debugger.
Use this setting to specify a connection mode. The options are:

- **no_reset_and_stop**  
  Do not submit a reset and halt any process currently running.

- **no_reset_and_no_stop**  
  Do not submit a reset or halt any process currently running.

- **reset_and_stop**  
  Do a processor reset and halt any process currently running.

- **reset_and_no_stop**  
  Do a processor reset but do not halt any process currently running.

- **prompt**  
  Display a prompt for the connection mode to use.

The options available for the **Connect_mode** setting are generic to all vehicles and supported processors and so might include options that are not supported by your target vehicle.

If you set connect mode from the Connection Control window, using **Connect (Defining Mode)**, this temporarily overrides any user-defined setting(s) in your target configuration file.

**Note**

If a prompt is specified in your board file, or in any .bcd file linked to the connection, it takes priority over any other user-defined connect mode setting. This prompt-first rule holds true regardless of where the setting is in the configuration hierarchy.

**Disconnect_mode**

When you disconnect from a target, RealView Debugger attempts to disconnect using the default disconnect mode, that is **As-is with Debug** (see Setting disconnect mode on page 2-23 for details).

Before disconnecting, RealView Debugger checks to see if a user-defined disconnect mode has been specified by the **Disconnect_mode** setting in your board file, or in any .bcd file linked to the connection. If such a setting is found, it becomes the default disconnect mode for this connection.

**Note**

A blank entry in the top-level **Disconnect_mode** setting in the board file ensures that any setting in a linked Board/Chip definition file is used instead. This might be important if you are using .bcd files to enable RTOS awareness in RealView Debugger.
Use this setting to specify a disconnection mode. The options are:

- **as_is_with_debug**: Leave the target in its current state, whether stopped or running, and maintain any debugging controls.
- **as_is_without_debug**: Leave the target in its current state, whether stopped or running, and remove any debugging controls.
- **prompt**: Display a prompt for the disconnection mode to use.

The options available for the `disconnect_mode` setting are generic to all vehicles and supported processors and so might include options that are not supported by your target vehicle.

If you set disconnect mode from the Connection Control window, using **Disconnect (Defining Mode)**..., this temporarily overrides any user-defined setting(s) in your target configuration file.

**Note**

If a prompt is specified in your board file, or in any `.bcd` file linked to the connection, it takes priority over any other user-defined disconnect mode setting. This prompt-first rule holds true regardless of where the setting is in the configuration hierarchy.

**Id_chip**

By default, the chip-id, or silicon-id, is loaded from the processor. When accessing special custom chips, it might be necessary to force the ID explicitly. The ID can be expressed as a 16-bit number or in `num.num.num` format. It can also be expressed as a name of the family member if known.

**Id_match**

This contains the expected silicon ID from the processor. If it does not match this value, you are prompted to choose whether or not to continue the connect operation. The ID can be expressed as a 16-bit number or in `num.num.num` format.

**Chip_name**

This defines the manufacturer name of the actual device, such as family name or core name. The Chip_name field enables you to specify a name to use in messages and lists displayed by RealView Debugger. It does not enforce the chip family selection. For that, you must use the Id_chip field.
Endianess

This field applies to ARM cores. Use it to set the byte order of the simulated processor.

A.4.2 BOARD, CHIP, and COMPONENT groups

Use these groups to hold connection settings when a standard board or chip, exists. Use a BOARD group to define a target board from a commercial hardware vendor, for example the ARM Evaluator-7T, or a custom design. Similarly, use a CHIP group to define significant devices on a target board or where the device itself is complex. You can use a COMPONENT group to define a core plus ASICs, either commercial or custom.

A CONNECTION group can refer to one or more of these groups by name or ID. A reference to one of these groups enables automatic use of the .jtg file, settings, and ETV information (ASIC, peripherals, and memory).

This type of entry can specify the default connection information which can then be overridden by settings in the CONNECTION group (see CONNECTION groups on page A-26 for details).

When you create a new group, the following entries are available, as described in Generic groups and settings on page A-5:

- Connect_with
- Advanced Information
- Configuration
- Description
- Project
- Family_select
- BoardChip_name.

--- Note ---

If you create a new BOARD, CHIP, or COMPONENT entry, the Connect_with group also contains an Ethernet group of settings. These are not supported in this release.
A.5 Custom connection reference

To configure custom connections, you must set up one or more of the following groups in the board file:

- **CONNECTION groups**
- **DEVICE groups** on page A-27
- **RVBROKER groups** on page A-27
- **Working with JTAG configuration files** on page A-28
- **Working with ARM RDI configuration files** on page A-29.

A.5.1 CONNECTION groups

**CONNECTION** entries are used to get a list of one or more target devices. This setting specifies:

- the type of the device
- the position of the device in the scan chain
- a name used to specify what to connect to.

In some JTAG file forms, additional information such as speed adjust can also be specified. Using a **CONNECTION** group automatically pulls the list of devices from the named file and provides an easy way to keep the two locked together.

The following entries are available, as described in **Generic groups and settings** on page A-5:

- Connect_with
- Remote
- Advanced_Information
- Configuration
- Auto_connect
- Pre_connect
- Description
- Project
- Disabled
- Shared
- BoardChip_name
- Family_select.
A.5.2 DEVICE groups

Use a DEVICE entry when only one device exists on the scan chain or when you have to specify a lot of information for a specific device. The name of this group must be a name within the .jtg file.

The following entries are available, as described in Generic groups and settings on page A-5:
- Connect_with
- Remote
- Advanced Information
- Description
- Project
- Configuration
- Disabled
- Shared
- Family_select
- BoardChip_name.

You can use a CONNECTION entry instead of a DEVICE entry.

A.5.3 RVBROKER groups

Use an RVBROKER entry to configure connections managed by RealView Connection Broker. This group enables you to access local host simulators or remote emulators, or simulators, running on another workstation on your network.

To access remote connections, debug target configuration files are held on the local workstation and are pushed across the network connection to the remote host.

Specific to this entry is the Simulator_info group that defines the simulator being used. The Default group contains:

**Sim_family** Specifies which member of the processor family to use.

**Sim_dll** Specifies the DLLs to load for custom ASICs or boards being simulated.

**Endianness** Used for ARM cores, this specifies the byte order of the processor being simulated.

The following entries are also available, as described in Generic groups and settings on page A-5:
- Remote
- Auto_connect
Using a hosts file

The rvbroker.brd entry allows inclusion of a special hosts file containing a list of all network nodes that can run or are running RealView Connection Broker. This enables multiple users to access simulators, emulators, and EVMs anywhere on the network by sharing this single file.

This is the same as using RealView Connection Broker to manage remote connections across your network. However, the connection details are contained in the rvbroker.brd file and not in the board file. This also enables multiple users to share the configuration details from a central location on the network.

For full details on working with remote connections, see Chapter 5 Working with Remote Targets.

A.5.4 Working with JTAG configuration files

JTAG configuration files, *.jtg, define the devices on the JTAG scan chain and their order. This information might be supplied either by the manufacturer or configured after installation. RealView Debugger uses JTAG files to access emulator targets on the local host for each supported processor.

Groups use the Configuration setting to name a file defining the JTAG scan chain, for example CONNECTION or DEVICE (see Generic groups and settings on page A-5.) These files are expected to use the extension .jtg. There are some shortcuts you can use in defining these files:

- If you provide the name of a .jtg file without specifying a path, RealView Debugger searches for it, first in the current working directory, then in your home directory, and then in the default settings directory \etc.
- If you use the same name for the .jtg file as the name of the CONNECTION, and the Configuration entry is blank, RealView Debugger searches for a file called connectionname.jtg, first in the current working directory, then in your home directory, and then in the default settings directory \etc.
- If the Configuration entry is blank and a .jtg file cannot be found, RealView Debugger prompts you to complete the configuration details.

Based on the information contained in the .jtg file, RealView Debugger determines the appropriate scan length and access sequence for the processor you are communicating with.
You use the JTAG file editor, accessed using the Connection Properties window, to edit a .jtg file, and you use the Connection Properties window itself to supplement this information if required, for example to define how the connection is made to the board.

The RealView Debugger base product includes the following JTAG files:

- **arm.jtg**: Specifies a single ARM processor on the scan chain.
- **arm_mp.jtg**: Specifies two ARM processors on the scan chain.
- **arm_oak.jtg**: Specifies a DSP Group Oak processor and then an ARM processor on the scan chain.

You must have the RealView Debugger DSP license to access the DSP Group processors.

**Note**

Where you have chosen a Custom installation, other .jtg files might be available to configure custom connections, for example for multiprocessor access such as ARM and DSP.

For details on working with JTAG files, see *Working with JTAG files* on page 4-36.

### A.5.5 Working with ARM RDI configuration files

RDI configuration entries in the board file are generated by the RDI configuration utilities, for example (*.rbe) ARM RDI Configuration Entries.

You can expand this entry to see the second-level entries that list autodetected targets. Expand one of these, for example multiice.rbe, to see the CONNECTION entry, CONNECTION=Multi-ICE, where the configuration file, multiice.cnf, is specified.

For details on working with RDI files, see *Working with RDI targets* on page 4-14.
Glossary

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

Access-provider connection
A debug target connection item that can connect to one or more target processors. The term is normally used when describing the RealView Debugger Connection Control window.

Address breakpoint
A type of breakpoint.

See also Breakpoint.

ADS
See ARM Developer Suite.

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

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

See also RealView Developer Suite.
ARM state
A processor that is executing ARM (32-bit) instructions is operating in ARM state.

See also Thumb state

Asynchronous execution
Asynchronous execution of a command means that the debugger accepts new commands as soon as this command has been started, enabling you to continue do other work with the debugger.

Backtracing
See Call Stack.

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

See also Little-endian.

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

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

Breakpoint
A user defined point at which execution stops in order that a debugger can examine the state of memory and registers.

See also Hardware breakpoint and Software breakpoint.

Call Stack
This is a list of procedure or function call instances on the current program stack. It might also include information about call parameters and local variables for each instance.

Conditional breakpoint
A breakpoint that halts execution when a particular condition becomes true. The condition normally references the values of program variables that are in scope at the breakpoint location.

Context menu
See Pop-up menu.

Core module
In the context of Integrator, an add-on development board that contains an ARM processor and local memory. Core modules can run stand-alone, or can be stacked onto Integrator motherboards.

See also Integrator

Current Program Status Register (CPSR)
See Program Status Register.

DCC
See Debug Communications Channel.
**Debug Communications Channel (DCC)**

A debug communications channel enables data to be passed between RealView Debugger and the EmbeddedICE logic on the target using the JTAG interface, without stopping the program flow or entering debug state.

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

ARM code generation tools generate debug information in DWARF2 format.

**Deprecated**

A deprecated option or feature is one that you are strongly discouraged from using. Deprecated options and features will not be supported in future versions of the product.

**Digital Signal Processor (DSP)**

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

**Doubleword**

A 64-bit unit of information.

**DSP**

*See Digital Signal Processor.*

**DWARF**

*See Debug With Arbitrary Record Format.*

**ELF**

Executable and Linking Format. ARM code generation tools produce objects and executable images in ELF format.

**Embedded Trace Macrocell (ETM)**

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

**EmbeddedICE logic**

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

*See also* IEEE1149.1.

**Emulator**

In the context of target connection hardware, an emulator provides an interface to the pins of a real core (emulating the pins to the external world) and enables you to control or manipulate signals on those pins.

**Endpoint connection**

A debug target processor, normally accessed through an *access-provider connection.*

**ETM**

*See Embedded Trace Macrocell.*

**ETV**

*See Extended Target Visibility.*
Extended Target Visibility (ETV)

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

Floating Point Emulator (FPE)

Software that emulates the action of a hardware unit dedicated to performing arithmetic operations on floating-point values.

FPE

See Floating Point Emulator.

Halfword

A 16-bit unit of information.

Hardware breakpoint

A breakpoint that is implemented using non-intrusive additional hardware. Hardware breakpoints are the only method of halting execution when the location is in Read Only Memory (ROM). Using a hardware breakpoint often results in the processor halting completely. This is usually undesirable for a real-time system.

See also Breakpoint and Software breakpoint.

IEEE Std. 1149.1

The IEEE Standard that defines TAP. Commonly (but incorrectly) referred to as JTAG.

See also Test Access Port

Integrator

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

Joint Test Action Group (JTAG)

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

JTAG

See Joint Test Action Group.

JTAG interface unit

A protocol converter that converts low-level commands from RealView Debugger into JTAG signals to the processor, for example to the EmbeddedICE logic and to the ETM.

Little-endian

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

See also Big-endian.

Multi-ICE

The ARM JTAG emulator debug tool for embedded systems. ARM registered trademark.
Pop-up menu  Also known as Context menu. A menu that is displayed temporarily, offering items relevant to your current situation. Obtainable in most RealView Debugger windows or panes by right-clicking with the mouse pointer inside the window. In some windows the pop-up menu can vary according to the line the mouse pointer is on and the tabbed page that is currently selected.

Processor core  The part of a microprocessor that reads instructions from memory and executes them, including the instruction fetch unit, arithmetic and logic unit and the register bank. It excludes optional coprocessors, caches, and the memory management unit.

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

Program Status Register (PSR)  Contains information about the current execution context. It is also referred to as the Current PSR (CPSR), to emphasize the distinction between it and the Saved PSR (SPSR), which records information about an alternate processor mode.

PSR  See Program Status Register.

RDI  See Remote Debug Interface.

RealView ARMulator ISS (RVISS)  The most recent version of the ARM simulator, RealView ARMulator ISS is supplied with RealView Developer Suite. It communicates with a debug target using RV-msg, through the RealView Connection Broker interface, and RDI.

See also RDI and RealView Connection Broker.

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

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

See also RealView ARMulator ISS.

RealView Debugger Trace  Part of the RealView Debugger product that extends the debugging capability with the addition of real-time program and data tracing. It is available from the Code window.

RealView ICE (RVI)  A JTAG-based debug solution to debug software running on ARM processors.
Remote Debug Interface (RDI)
The Remote Debug Interface (RDI) is an ARM standard procedural interface between a debugger and the debug agent. RDI gives the debugger a uniform way to communicate with:

- a simulator running on the host (for example, RVISS)
- a debug monitor running on hardware that is based on an ARM core accessed through a communication link (for example, Angel)
- a debug agent controlling an ARM processor through hardware debug support (for example, RealView ICE or Multi-ICE).

Remote_A
Remote_A is a software protocol converter and configuration interface. It converts between the RDI 1.5 software interface of a debugger and the Angel Debug Protocol used by Angel targets. It can communicate over a serial or Ethernet interface.

RTOS
Real Time Operating System.

RVCT
See RealView Compilation Tools.

RVISS
See RealView ARMulator ISS.

Scan chain
A scan chain is made up of serially-connected devices that implement boundary-scan technology using a standard JTAG TAP interface. Each device contains at least one TAP controller containing shift registers that form the chain. Processors might contain several shift registers to enable you to access selected parts of the device.

Scope
The range within which it is valid to access such items as a variable or a function.

Script
A file specifying a sequence of debugger commands that you can submit to the command-line interface using the `include` command.

Semihosting
A mechanism whereby I/O requests made in the application code are communicated to the host system, rather than being executed on the target.

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

Software breakpoint
A breakpoint that is implemented by replacing an instruction in memory with one that causes the processor to take exceptional action. Because instruction memory must be altered software breakpoints cannot be used where instructions are stored in read-only memory. Using software breakpoints can enable interrupt processing to continue during the breakpoint, making them more suitable for use in real-time systems.

See also Breakpoint and Hardware breakpoint.

Software Interrupt (SWI)
An instruction that causes the processor to call a programmer-specified subroutine. Used by the ARM standard C library to handle semihosting.
Glossary

SPSR  
Saved Program Status Register.  
*See also* Program Status Register.

SWI  
*See* Software Interrupt.

**Synchronous execution**  
*Synchronous execution* of a command means that the debugger stops accepting new commands until this command is complete.

**Synchronous starting**  
Setting several processors to a particular program location and state, and starting them together.

**Synchronous stopping**  
Stopping several processors in such a way that they stop executing at the same instant.

**TAP**  
*See* Test Access Port.

**TAP Controller**  
Logic on a device which enables access to some or all of that device for test purposes. The circuit functionality is defined in Std. IEEE1149.1.  
*See also* Test Access Port and IEEE Std. 1149.1.

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

**Target vehicle**  
Target vehicles provide RealView Debugger with a standard interface to disparate targets so that the debugger can connect easily to new target types without having to make changes to the debugger core software.

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

**Test Access Port (TAP)**  
The port used to access the TAP Controller for a given device. Comprises TCK, TMS, TDI, TDO, and nTRST (optional).

**Thumb state**  
A processor that is executing Thumb (16-bit) instructions is operating in Thumb state.  
*See also* ARM state

**Tracepoint**  
A tracepoint can be a line of source code, a line of assembly code, or a memory address. In RealView Debugger, you can set a variety of tracepoints to determine exactly what program information is traced.
Glossary

**Trigger**
In the context of breakpoints, a trigger is the action of noticing that the breakpoint has been reached by the target and that any associated conditions are met.

In the context of tracing, a trigger is an event that instructs the debugger to stop collecting trace and display the trace information around the trigger position, without halting the processor. The exact information that is displayed depends on the position of the trigger within the buffer.

**TVS**
*See* Target Vehicle Server.

**Vector Floating Point (VFP)**
A standard for floating-point coprocessors where several data values can be processed by a single instruction.

**VFP**
*See* Vector Floating Point.

**Watch**
A watch is a variable or expression that you require the debugger to display at every step or breakpoint so that you can see how its value changes. The Watch pane is part of the RealView Debugger Code window that displays the watches you have defined.

**Watchpoint**
In RealView Debugger, this is a hardware breakpoint.

**Word**
A 32-bit unit of information.
# Index

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

## A

<table>
<thead>
<tr>
<th>Item</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>About this book</td>
<td>vi</td>
</tr>
<tr>
<td>Access configuration settings</td>
<td>A-12</td>
</tr>
<tr>
<td>Access-provider connections</td>
<td></td>
</tr>
<tr>
<td>in Connection Control window</td>
<td>2-3</td>
</tr>
<tr>
<td>menu</td>
<td>2-8</td>
</tr>
<tr>
<td>Access-rule configuration settings</td>
<td>A-10</td>
</tr>
<tr>
<td>Access_Method configuration settings</td>
<td>A-16</td>
</tr>
<tr>
<td>Access_size configuration settings</td>
<td>A-10</td>
</tr>
<tr>
<td>Address configuration settings</td>
<td>A-18</td>
</tr>
<tr>
<td>ADI</td>
<td>4-25</td>
</tr>
<tr>
<td>Agilent JTAG probe configuration files</td>
<td>1-11</td>
</tr>
<tr>
<td>configuring devices</td>
<td>4-25</td>
</tr>
<tr>
<td>configuring JTAG Frequency</td>
<td>4-25</td>
</tr>
<tr>
<td>configuring targets</td>
<td>4-25</td>
</tr>
<tr>
<td>ADI configuration dialog</td>
<td>4-25</td>
</tr>
<tr>
<td>Advanced Information block</td>
<td>1-3, 1-9, A-8</td>
</tr>
<tr>
<td>Connect mode</td>
<td>2-14</td>
</tr>
<tr>
<td>Disconnect mode</td>
<td>2-24</td>
</tr>
<tr>
<td>Advanced Information configuration settings</td>
<td>A-6</td>
</tr>
<tr>
<td>Agilent JTAG probe</td>
<td>ADI 4-25</td>
</tr>
<tr>
<td>Angel debug monitor</td>
<td>2-3, 4-16, 4-21</td>
</tr>
<tr>
<td>see also RDI, Remote_A</td>
<td></td>
</tr>
<tr>
<td>Application_Load configuration settings</td>
<td>A-9</td>
</tr>
<tr>
<td>ARM Agilent Debug Interface</td>
<td>4-25</td>
</tr>
<tr>
<td>ARMulator Configuration dialog</td>
<td>4-19, 4-29</td>
</tr>
<tr>
<td>ARM_config configuration settings</td>
<td>A-18</td>
</tr>
<tr>
<td>Arm_swi_num configuration settings</td>
<td>A-19</td>
</tr>
<tr>
<td>Attributes configuration settings</td>
<td>A-10</td>
</tr>
<tr>
<td>Audience, intended</td>
<td>vi</td>
</tr>
<tr>
<td>Auto-disconnect</td>
<td></td>
</tr>
<tr>
<td>in single processor mode</td>
<td>2-22</td>
</tr>
<tr>
<td>Auto_connect configuration settings</td>
<td>A-6</td>
</tr>
<tr>
<td>AXYS Design Automation Inc</td>
<td>4-27, 4-35</td>
</tr>
</tbody>
</table>

## B

<table>
<thead>
<tr>
<th>Item</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backup files</td>
<td></td>
</tr>
<tr>
<td>automatic</td>
<td>1-15</td>
</tr>
<tr>
<td>manual</td>
<td>1-15</td>
</tr>
<tr>
<td>restoring</td>
<td>1-16</td>
</tr>
<tr>
<td>Backups configuration files</td>
<td>3-10</td>
</tr>
<tr>
<td>files</td>
<td>1-15</td>
</tr>
<tr>
<td>restoring</td>
<td>1-16</td>
</tr>
<tr>
<td>Base configuration settings</td>
<td>A-15, A-17</td>
</tr>
<tr>
<td>Base_address configuration settings</td>
<td></td>
</tr>
<tr>
<td>Base_mask configuration settings</td>
<td>A-11</td>
</tr>
<tr>
<td>Base_scale configuration settings</td>
<td>A-11</td>
</tr>
<tr>
<td>BOARD and CHIP entries</td>
<td>3-19</td>
</tr>
<tr>
<td>entries in board file</td>
<td>1-8, 3-12, 3-19</td>
</tr>
</tbody>
</table>
Index

linking entries 3-15, 3-17, 3-18
settings reference A-25
Board file 1-2
about 1-2, 4-2
Advanced Information block
configuration settings A-8
asterisk 4-5
BOARD groups A-25
CHIP groups A-25
COMPONENT groups A-25
collection entries A-4, A-8
connection entries A-4, A-26
disabled entries 4-3
disabling entries 4-5
duplicate entries 4-4
enabled entries 4-3
enabling entries 4-4
entries 1-7, A-3
in the workspace 2-6
location 1-2, 3-3
merging hardware settings 3-17
ordering hardware 3-17, 3-18, 3-22
ordering hardware descriptions 3-17, 3-18, 3-22
restoring defaults 4-5
restoring entries 4-5
selecting 2-6
settings reference A-3
target configuration 3-3
BoardChip_name configuration settings A-7
Board/Chip Definitions 1-3, 1-8
creating new 3-10
ETV 3-5
files 3-5
in search path 1-14
linking 3-8
linking to connection 3-14
location 1-14
referring to 3-8
 supplied in installation 3-7
Book, about this vi

C

Channel viewers
in Remote_A 4-22
CHIP

and BOARD entries 3-19
entries in board file 1-8, 1-3, 3-19
settings reference A-25
Chip_name configuration settings A-24
Clock_speed configuration settings A-19
Command line arguments
-Home 1-13
-install 1-13
-user 1-14
Commands
CONNECT 2-18
DISCONNECT 5-21
Commands configuration settings A-22
Comments
on documentation x
on RealView Debugger x
COMPONENT
entries in board file 1-8, 3-13
settings reference A-25
Concat_Register configuration settings A-15
Config configuration settings A-20
Configuration
backups 1-15
entries 1-8
files 1-11
entries in board file A-4, A-8
Config_file A-19
Connect_mode A-23
Connect_with A-5
Cross_trigger A-20
Description A-6, A-13
Disabled A-7
Disconnection_mode A-24
D_Abort A-18
Enabled A-19
Endianness A-25, A-27
Enum A-14, A-15, A-16
Error A-18
Exit_Options A-21
Family_select A-7
FIQ A-18
Flash_type A-12
Fpoint_emu A-19
generic settings A-5
Gui_name A-14, A-15, A-16
Heap_base A-18
Heap_size A-18
High_mask A-16
High_name A-16

setting registers 3-38
stack heap 3-45
top of memory 3-45

Configuration settings
Access A-12
Access-rule A-10
Access_Method A-16
Address A-18
Advanced Information A-6
Application_Load A-9
ARM_config A-18
Arm_swim num A-19
Attributes A-10
Auto_connect A-6
Base A-15, A-17
Base_address A-21
Base_mask A-11
Base_scale A-11
BoardChip_name A-7
Chip_name A-24
Clock_speed A-19
Commands A-22
Concat_Register A-15
Config A-20
Configuration A-6
Config_file A-19
Connect_mode A-23
Connect_with A-5
Cross_trigger A-20
Description A-6, A-13
Disabled A-7
Disconnection_mode A-24
D_Abort A-18
Enabled A-19
Endianness A-25, A-27
Enum A-14, A-15, A-16
Error A-18
Exit_Options A-21
Family_select A-7
FIQ A-18
Flash_type A-12
Fpoint_emu A-19
generic settings A-5
Gui_name A-14, A-15, A-16
Heap_base A-18
Heap_size A-18
High_mask A-16
High_name A-16

Index-2 Copyright © 2002-2004 ARM Limited. All rights reserved. ARM DUI 0182E
Index

High_shift A-16
Hostname A-6
Id_chip A-24
Id_match A-24
Internal A-10
IODEVICE A-5
IRQ A-18
Length A-12, A-15, A-16, A-17
Len_mask A-11
Len_scale A-11
Len_table A-11
Load_command A-9
Load_set_pc A-9
Load_using A-9
Load_when A-20, A-21
Logic_Analyzer A-19
Low_mask A-16
Low_name A-16
Low_shift A-16
Machine A-20
Manufacturer A-5
Map_rule A-13
Mask A-13
Memory_block A-10
Memory_type A-15
Method_name A-16
Names A-14
On_equal A-13
ordering 3-22
Peripherals A-16
Port A-6
Position A-14
precedence 3-22
Pre_connect A-6, A-22
Project A-7
Properties A-19
P_Abort A-18
Read_only A-14, A-15
RealView ARMulator ISS A-19
reference A-3
Register A-13, A-14, A-16
Register_base A-11
Register_enum A-14
Register_length A-11
Register_Pos_Len A-11
Register_Window A-17
Remote A-6
Reset A-18
RSD A-21
RTOS A-21
Semihosting A-18, A-19
Shared A-7, A-10
Shared_id A-10
Signed A-14
Sim_dll A-27
Sim_family A-27
Size A-14
Speed A-5
Stack_bottom A-18
Stack_Heap A-18
Stack_size A-18
Start A-12, A-14, A-16
SWI A-18
Sym_match A-20
System_Stop A-21
Thumb_sw_num A-19
Top_memory A-19
Trig_in_dis A-21
Trig_in_ena A-20
Trig_out_dis A-21
Trig_out_ena A-21
Type A-12, A-15, A-16, A-17
Undefined A-18
Update_rule A-12, A-13
Value A-13
Vector A-19
Vectors A-18
Vector_catch A-19
Vendor A-20, A-21
Volatile A-10, A-13, A-14, A-15
Wait_states A-12
Write_only A-15
Config_file configuration settings
A-19
Connect
No Reset and No Stop 2-15
No Reset and Stop 2-15
Reset and No Stop 2-16
Reset and Stop 2-15
CONNECT command 2-18
Connect mode 2-13
configuration example 3-33
configuration rules 3-32
default 2-13
default state 2-16
defining 2-15
defining for a connection 2-15
in Advanced_Information block 2-14
not honored 2-14, 3-32
prompting for a connection 2-16
prompting for a disconnection 2-28
selection box 2-15
state options 2-15
substitution 2-14, 3-32
user-defined 2-13, 2-14
Connecting 1-2, 1-3, 1-7, 2-4, 2-5
default files 2-5
to Multi-ICE 4-24
to multiple targets 2-17
to RealView ARMulator ISS 2-12
to remote targets 5-8, 5-13
to targets 2-11
unconfigured targets 2-12
using names 2-18
using numbers 2-17
CONNECTION
entries in board file A-26
Connection 1-2
collapsed in Connection Control
window 2-5
creation 1-2
creating custom 1-8
DLLs 1-7
entries 1-7
entries in board file A-4, A-26
menu 2-12
viewing details 4-34
viewing remote connections 5-9
Connection Broker 4-27
configuring RealView ARMulator
ISS 2-9
local host connection 2-5, 4-31, 5-2
local mode 4-27
remote host connection 4-31, 5-2
Simulator Broker 4-27
using a hosts file 5-15
using DSP simulators 1-3
using simulators 4-35
Connection Control button 2-2
Connection Control window 1-3, 1-7,
2-2
Access-provider connections 2-3
adding RDI targets 4-15
collapsing connections 2-5
collapsing entries 2-4
Index

Index-4

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

ARM DUI 0182E
Disconnecting  5-21
  auto-disconnect  2-22
  by exiting  2-28
Code window behavior  2-21
  confirming  2-28
  from Connection Control window  2-22
  in multiprocessor mode  2-21
  project behavior  2-21
  reconnecting  2-29
  using the CLI  2-30
  using the menu  2-22
  Disconnect_mode configuration settings  A-24
Documentation feedback  x
Domain name
  configuring in remote targets  5-7
  DSP
  configuring new connection  4-39
  connecting to hardware  4-40
  connecting with RealView ICE  4-40
  simulator  4-27, 4-35
  D_About configuration settings  A-18
  Family_select configuration settings  A-7
  Feedback
    on documentation  x
    on RealView Debugger  x
  File search path  1-14
  Files
    board file  1-2
    configuration  1-11
    JTAG  4-36
    rvbroker.brd  5-3, 5-15
    rvdebug.brd  1-11
    using a hosts file  5-15
    *.auc  4-31
    *.bcd  1-3, 1-8, 1-11
    *.bcd supplied  3-7
    *.brd  1-11
    *.cnf  1-11
    *.jtg  4-36
    *.prc  1-12
    *.rbe  1-12
    *.rvc  1-12
    .jtg  3-4
    .rvc  3-3
  FIQ configuration settings  A-18
  Flash
    programming example  3-53
  Flash_type configuration settings  A-12
  Format, of registers  3-7
  Fpoint_emu configuration settings  A-19
  Glossary
    Glossary-1
  Group Name/Type selector dialog  4-9,  4-32
  Gui_name configuration settings  A-14,
  A-15, A-16
  Error configuration settings  A-18
  ETV  1-3, 1-9, 1-2
  Evaluator-7T  3-7
  Exit_Options configuration settings  A-21
  Extended Target Visibility  3-2
  ETV  3-2
  High_mask configuration settings  A-16
  High_name configuration settings  A-16
  High_shift configuration settings  A-16
  Home
    see Directories
  Hostname
    configuring in remote targets  5-7
    Hostname configuration settings  A-6
  Hosts
    accessing remote targets  5-15
    Hosts file  5-15
    configuring local  5-16
    configuring network  5-19
    connecting local  5-18
    contents  5-17, 5-18
    creating  5-17
    creating local  5-15
    editing  5-17
    in board file  5-16
    in Connection Control window  5-16, 5-18
    multiple targets  5-18
    network  5-19
  Id_chip configuration settings  A-24
  Id_match configuration settings  A-24
  Integrator
    configuration example  3-24
    Integrator/AP  3-7, 3-8, 3-12
    Integrator/CM  3-8
    Integrator/CP  3-8
  Intended audience vi
  Internal configuration settings  A-10
  IOdevice configuration settings  A-5
  IP address
    configuring in remote targets  5-7
  IRQ configuration settings  A-18
  Heap_base configuration settings  A-18
  Heap_size configuration settings  A-18
  JTAG
    configuration files  3-4
    JTAG files
Index

adding new devices 4-38
configuring Bypass 4-38
configuring connect options 4-38
configuring device name 4-37
configuring devices 4-38
configuring extra entries 4-38
configuring processor name 4-38
copying devices 4-39
Edit Configuration-File Contents 4-36, 4-37
editing 4-36
in Oak DSP connection 4-40
in the Connection Control window 4-36
location 3-4
target configuration 4-36
TDI 4-38
TDO 4-38
Test Data In 4-38
Test Data Out 4-38
testing 2-10
JTAG targets
JTAG target menu 2-10

L

Length configuration settings A-12, A-15, A-16, A-17
Len_mask configuration settings A-11
Len_scale configuration settings A-11
Len_table configuration settings A-11
Licenses, valid 2-19
Linking
a board to a connection 3-15
boards to a connection 3-17
boards to boards 3-18
Load_command configuration settings A-9
Load_set_pc configuration settings A-9
Load_using configuration settings A-9
Load_when configuration settings A-20, A-21
Logic_Analyzer configuration settings A-19
Low_mask configuration settings A-16
Low_name configuration settings A-16
Low_shift configuration settings A-16
Multi-ICE direct connect 4-22, 4-36

M

Machine configuration settings A-20
Manufacturer configuration settings A-5
Map_rule configuration settings A-13
Mask configuration settings A-13
MaxSim simulator 4-27, 4-35
Memory
setting configuration example 3-42
Memory block
configuration example 3-42
Memory map
configuration example 3-36
Memory_block configuration settings A-10
Memory_type configuration settings A-15
Method_name configuration settings A-16
Multi-ICE
accessing remote targets 5-2
and RMHost 3-49
compatibility with RealView Debugger 4-25
configuration 4-23
configuration files 1-11
configuring cache clean 4-24
configuring connection 4-23
configuring DSP targets 4-22
configuring endianess 4-23
configuring in multiprocessor mode 4-24
configuring processor settings 4-23
configuring RDI targets 4-22
configuring targets 4-22
configuring the target 2-5, 2-13, 2-20
configuring trace 4-24
connections 4-24
DCC semihosting 4-25
direct connect 4-22
dll version numbers 4-24
setting per-processor options 4-24
troubleshooting connections 2-20
Multi-ICE configuration dialog 4-23

N

Names configuration settings A-14
Network hosts file 5-19
configuring 5-19
contents 5-19
creating 5-19
editing 5-19
in board file 5-19
in Connection Control window 5-20

O

Oak simulator 4-27, 4-35
On_equal configuration settings A-13

P

Peripherals configuration settings A-16
Plugins
for RealMonitor 3-5
for RTOS 3-5
Port configuration settings A-6
Position configuration settings A-14
Pre_connect configuration settings A-6, A-22
Problem solving x
Product feedback x
Project configuration settings A-7
Projects
on disconnecting 2-21
Properties configuration settings A-19
P_Abort configuration settings A-18

Q

Queries x
R

RDI

adding devices 2-7, 2-9, 4-14, 4-15
adding DLLs 4-15
adding targets 2-7, 2-9, 4-14, 4-15
adding third-party 2-7, 2-9, 4-14, 4-15
configuration files 3-4
configuring targets 4-14, 4-17
creating new target 4-15
*rbe files 3-4
*.cnf files 3-4
RDI Target List dialog 2-7, 2-9, 4-14, 4-15
adding DLLs 4-15
enabling RDI connections 4-15
RDI 1.5.1 4-14
Read_only configuration settings A-14, A-15
RealMonitor

Board/Chip definition files 3-5
configuration 4-26
configuration dialog 3-51
configuration files 1-11
configuring JTAG Controller 4-26
configuring RDI Module Server 4-26
configuring the target 2-5, 2-13, 2-20
debugging demonstration 3-52
example 3-48
loading symbols 3-52
Multi-ICE and RMHost 3-49
plugins 3-5
RMHost 3-48
RealMonitor configuration dialog 4-26
RealView ARMulator ISS 4-18, 4-27, 4-28, 4-30
configuration settings A-19
configuring 2-9, 4-18
configuring clock 4-19, 4-29
configuring connections 4-28
configuring devices 4-28
configuring endianess 4-19, 4-20, 4-29, 4-30
configuring FPA 4-19
configuring map files 4-20
configuring MMU 4-20, 4-30
configuring new connection 4-31
configuring paletables 4-20, 4-30
configuring processor 4-19, 4-29
configuring targets 4-19
configuring VFP 4-20, 4-30
Simulator Broker connections 4-27
RealView Debugger
compatibility with Multi-ICE 4-25
connecting to a target 1-2
target configuration 1-2
RealView ICE
Disconnect mode 2-14, 3-32
RealView ICE
accessing remote targets 5-2
adding targets 4-8
configuration files 1-11, 1-16
configuring devices 4-11
configuring interface unit 4-11
configuring new connection 4-9
configuring the target 2-5, 2-13, 2-20
Connect mode 2-14, 3-32
in the Connection Control window 4-8
Integrator configuration example 3-24
limitations 3-25
RVConfig 2-9, 4-11
troubleshooting connections 2-20
viewing targets 4-8
RealView Network Broker 5-2
closing down 5-9, 5-11, 5-14
see also Connection Broker
Register configuration settings A-13, A-14, A-16
Register format 3-7
Registers
setting configuration example 3-38
Register_base configuration settings A-11
Register_enum configuration settings A-14
Register_length configuration settings A-11
Register_Pos_Len configuration settings A-11
Register_Window configuration settings A-17
Remote configuration settings A-6
Remote targets
accessing 5-2
accessing Multi-ICE 5-2
accessing RealView ICE 5-2
accessing simulators 5-3
configuring domain name 5-7
configuring host name 5-7
configuring IP address 5-7
configuring new emulator 5-10
configuring new simulator 5-5
connecting 5-8, 5-13
emulators 5-10
RVBROKER 5-3
rvbroker.brd 5-3
simulators 5-5
viewing connections 5-9
Remote_A
channel viewers 4-22
connection driver 4-21
domain 4-22
domain name 4-22
target configuration 4-21
Realview_A connection dialog 4-21
Reset configuration settings A-18
RMHost 3-48
RSD configuration settings A-21
RTOS
Board/Chip definition files 3-5
plugins 3-5
RTOS configuration settings A-21
RVBroker
entries in board file 5-3, A-27
rvbroker.brd 5-3
entries in board file 5-3
in search path 5-15
RVCfg
Advanced settings 4-13
and .rvc files 1-12
AutoConfigure Scan Chain 4-12
with RealView ICE 4-11
RVCfg dialog 2-9, 4-12
RVDEBUG_HOME 1-13
RVDEBUG_INSTALL 1-13, 1-14
RVDEBUG_SHARE 1-14
RVISS 4-18
RV-msg
configuration files 1-11, 3-3
Index

S
Search path, file 1-14
Semihosting configuration settings A-18, A-19
Shared configuration settings A-7, A-10
Shared_id configuration settings A-10
Signed configuration settings A-10
Simulator Broker
  Connection Broker 4-27
  connections to RealView
    ARMulator ISS 4-27
    connections to simulators 4-27
Sim_dll configuration settings A-27
Sim_family configuration settings A-27
Size configuration settings A-14
Speed configuration settings A-5
Stack heap
  configuration example 3-45
Stack_bottom configuration settings A-18
Stack_Heap configuration settings A-18
Stack_size configuration settings A-18
Start configuration settings A-12, A-14, A-16
Structure of this book vi
SWI configuration settings A-18
Sym_match configuration settings A-20
Synch tab
  Connection Control window 2-2
System_Stop configuration settings A-21

T
Target
  see Debug target
Target vehicle
  in Connection Control window 2-3
Terminology Glossary-1
Thumb_sw1_num configuration settings A-19
Top of memory
  configuration example 3-45
Top_memory configuration settings A-19
Trig_in_dis configuration settings A-21
Trig_in_ena configuration settings A-20
Trig_out_dis configuration settings A-21
Trig_out_ena configuration settings A-21
Troubleshooting
  configurations 3-57
  connections 2-20
  Multi-ICE connections 2-20
  RealView ICE connections 2-20
Type configuration settings A-12, A-15, A-16, A-17

U
Undefined configuration settings A-18
Update_rule configuration settings A-12, A-13

V
Valid licenses 2-19
Value configuration settings A-13
Vector configuration settings A-19
Vectors configuration settings A-18
Vector_catch configuration settings A-19
Vehicle 1-7
  connecting to RealView ARMulator ISS 1-7
  connecting to RealView ICE 1-7
  menu 2-7
Vendor configuration settings A-20, A-21
Volatile configuration settings A-10, A-13, A-14, A-15

W
Wait_states configuration settings A-12