RealView™ Debugger

Version 1.6.1

Target Configuration Guide

ARM DUI 0182D

RealView Debugger
Target Configuration Guide

Release Information

The following changes have been made to this document.

Change History

Date	Issue	Change
April 2002	A	RealView Debugger 1.5 release.
September 2002	B	RealView Debugger v1.6 release.
February 2003	C	RealView Debugger v1.6.1 release.
September 2003	D	RealView Debugger v1.6.1 release for RVDS 2.0.

Product Status

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

Web Address

`http://www.arm.com`

Contents

RealView Debugger Target Configuration Guide

Preface

This preface introduces the *RealView™ Debugger v1.6 Target Configuration Guide*. It contains the following sections:

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

About this book

RealView Debugger version 1.6 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.

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.

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 v1.6 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.

You must refer to the manual for the product, for example the *Multi-ICE® Version 2.2 User Guide*, for details of each RDI debug agent.

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

Refer to this for explanations of terms used in this book.

Typographical conventions

The following typographical conventions are used in this book:

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

Further reading

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

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

ARM publications

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

- *RealView Debugger v1.6 Essentials Guide* (ARM DUI 0181)
- *RealView Debugger v1.6 User Guide* (ARM DUI 0153)
- *RealView Debugger v1.6 Command Line Reference Guide* (ARM DUI 0175)
- *RealView Debugger v1.6 Extensions User Guide* (ARM DUI 0174).

Refer to the following books in the RVCT document suite for more information on the compilation tools component of RVDS 2.0:

- *RealView Compilation Tools Essentials Guide* (ARM DUI 0202)
- *RealView Compilation Tools Compiler and Libraries Guide* (ARM DUI 0205)
- *RealView Compilation Tools Linker and Utilities Guide* (ARM DUI 0206)
- *RealView Compilation Tools Assembler Guide* (ARM DUI 0204)
- *RealView Compilation Tools Developer Guide* (ARM DUI 0203).

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

- *ARM Architecture Reference Manual* (ARM DDI 0100). David Seal, *ARM Architecture Reference Manual, Second Edition*, 2001, Addison Wesley. ISBN 0-201-73719-1.
- *ARM Reference Peripheral Specification* (ARM DDI 0062).
- *ARM-Thumb® Procedure Call Standard (ATPCS) Specification* (SWS ESPC 0002).

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

- *Multi-ICE Version 2.2 User Guide* (ARM DUI 0048)
- *ARM Agilent Debug Interface User Guide* (ARM DUI 0158)
- *ARM Firmware Suite Version 1.4 User Guide* (ARM DUI 0136)
- *ARM RMHost User Guide* (ARM DUI 0137)
- *ARM RMTarget Integration Guide* (ARM DUI 0142).

Refer to the following documentation for information relating to specific ARM Limited processors:

- *ARM7DI™ Datasheet* (ARM DDI 0027)
- *ARM710T™ Datasheet* (ARM DDI 0086)
- *ARM720T™ Datasheet* (ARM DDI 0087)
- *ARM740T™ Datasheet* (ARM DDI 0008)
- *ARM7TDMI™ Technical Reference Manual* (ARM DDI 0210)
- *ARM7EJ-S™ Technical Reference Manual* (ARM DDI 0214)
- *ARM9TDMI™ Technical Reference Manual* (ARM DDI 0180)
- *ARM920T™ Technical Reference Manual* (ARM DDI 0151)
- *ARM922T™ Technical Reference Manual* (ARM DDI 0184)
- *ARM9EJ-S™ Technical Reference Manual* (ARM DDI 0222)
- *ARM926EJ-S™ Technical Reference Manual* (ARM DDI 0198)
- *ARM940T™ Technical Reference Manual* (ARM DDI 0144)
- *ARM946E-S™ Technical Reference Manual* (ARM DDI 0201)
- *ARM966E-S™ Technical Reference Manual* (ARM DDI 0213)
- *ARM1020E™ Technical Reference Manual* (ARM DDI 0177)
- *ARM1022E™ Technical Reference Manual* (ARM DDI 0237).

Refer to the following documentation for details on the FLEX*lm* license management system, supplied by GLOBEtrotter Inc., that controls the use of ARM applications:

- *ARM FLEXlm License Management Guide v3.0* (ARM DUI 0209).

Make sure that you use version 3.0 of this documentation for details on license management in RealView Debugger v1.6.1 for RVDS 2.0.

Other publications

For a comprehensive introduction to ARM architecture see:

Steve Furber, *ARM System-on-Chip Architecture, Second Edition*, 2000, Addison Wesley, ISBN 0-201-67519-6.

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

Jeffrey E. F. Friedl, *Mastering Regular Expressions, Powerful Techniques for Perl and Other Tools*, 1997, O'Reilly & Associates, Inc. ISBN 1-56592-257-3.

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

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

For more information about the JTAG standard, see:

IEEE Standard Test Access Port and Boundary Scan Architecture (IEEE Std. 1149.1), available from the IEEE (`www.ieee.org`).

For specific information relating to Oak and TeakLite processors from the DSP Group see `http://www.dspg.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 RealView configuration* on page 1-2
- *Comparing target configuration to connection configuration* on page 1-3
- *Configuration files* on page 1-6.

1.1 About RealView configuration

You configure the way that RealView Debugger connects to and interacts with your target using a *board file*. Specifically, the board file enables you to configure:

- debugger to target connection details, such as interface type and instance, TAP controller positions, and connection interface address
- target processor characteristics, for example processor type and endianess
- target peripheral register and memory configuration.

The board file is initially called `rvdebug.brd` and, along with files that it references, it is stored in your default home directory, created for you by RealView Debugger, for example in `\home\`*`user_name`*. For more information, see *About target configuration* on page 3-2.

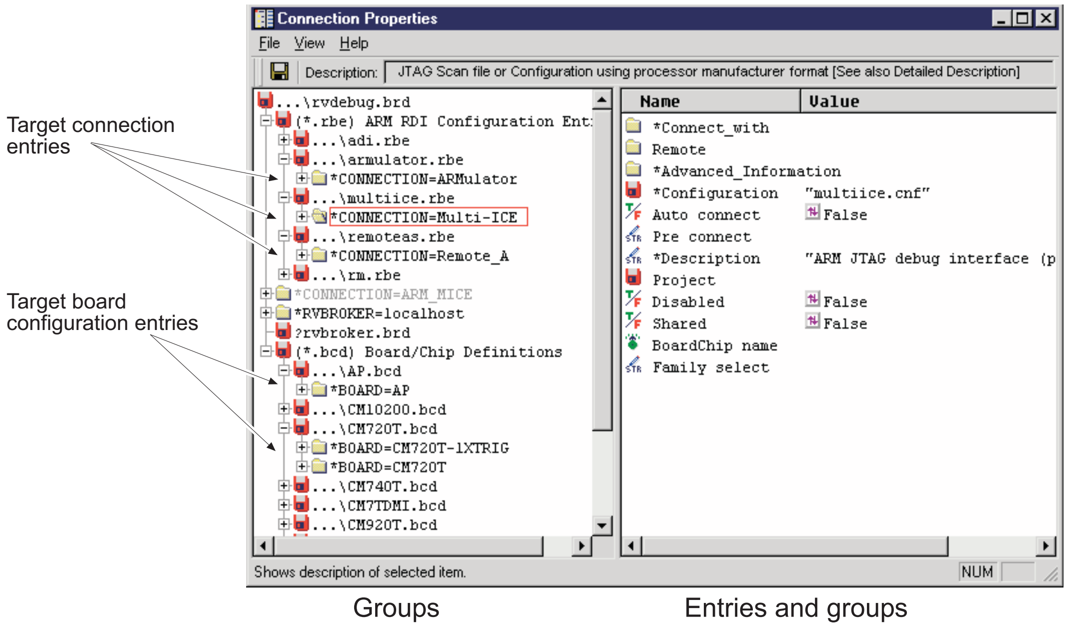
The *RealView Debugger v1.6 Essentials Guide* explains how to connect to a simple target using the Connection Control window. The contents of this window are defined by elements of the board file. You can change the board file to add to, and modify, the elements displayed in the Connection Control window.

1.2 Comparing target configuration to connection configuration

There is a distinction in RealView Debugger between configuring the target, for example memory layout, and configuring how the target is accessed. Within the board file you can specify two types of entry, shown in Figure 1-1:

- connection information, using `CONNECTION` and `DEVICE`
- target configuration information, using `BOARD`, `CHIP`, and `COMPONENT`.

Entries such as `CONNECTION` and `CHIP` contain many individual settings and in this book are called *groups*. Groups can contain subgroups. Individual settings, such as the `Disabled` setting shown in Figure 1-1, are always stored in groups, such as the `CONNECTION` group.



Description: JTAG Scan file or Configuration using processor manufacturer format [See also Detailed Description]

```
...\rvdebug.brd
(*.rbe) ARM RDI Configuration Ent:
  ...\adi.rbe
  ...\armulator.rbe
    *CONNECTION=ARMulator
  ...\multiice.rbe
    *CONNECTION=Multi-ICE
  ...\remoteas.rbe
    *CONNECTION=Remote_A
  ...\rm.rbe
*CONNECTION=ARM_MICE
*RVBROKER=localhost
?rvbroker.brd
(*.bcd) Board/Chip Definitions
  ...\AP.bcd
    *BOARD=AP
  ...\CM10200.bcd
  ...\CM720T.bcd
    *BOARD=CM720T-1XTRIG
    *BOARD=CM720T
  ...\CM740T.bcd
  ...\CM7TDMI.bcd
  ...\CM920T.bcd
```

Name	Value
*Connect_with	
Remote	
*Advanced_Information	
*Configuration	"multiice.cnf"
Auto connect	False
Pre connect	
*Description	"ARM JTAG debug interface (p
Project	
Disabled	False
Shared	False
BoardChip name	
Family select	

Shows description of selected item. NUM

Figure 1-1 Connection Properties window

The left pane of the Connection Properties window lists the groups for the board file. The red floppy disk icons indicate a group that is stored in its own file. The yellow folder icons indicate that the group is stored within an enclosing group.

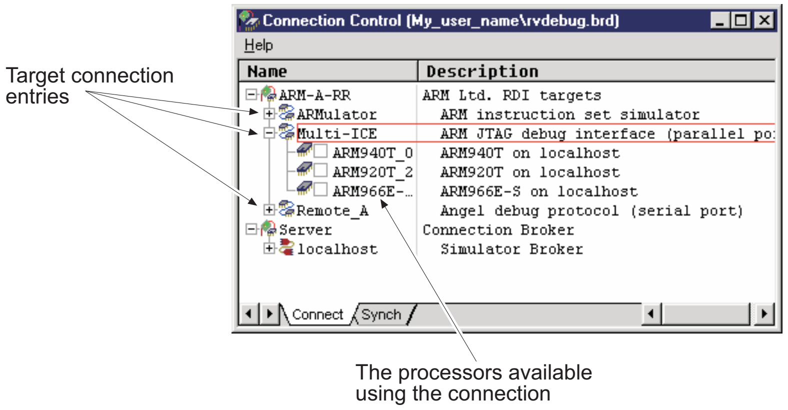
The right pane of the Connection Properties window lists the groups and other entries that a selected group contains. Yellow icons in this pane mean the same as yellow folder icons in the left pane, but a red floppy disk icon in the right pane is also used for any setting that references a file, for example the `Configuration` value in Figure 1-1.

1.2.1 Connection entries

The target connection groups shown in Figure 1-1 on page 1-3 are set up for you when you install *ARM Developer Suite* (ADS) and ARM Multi-ICE, and are provided by a RealView Debugger connection interface component, or *vehicle*, called ARM-A-RR. The ARM-A-RR vehicle provides the interface to ARM RDI debug targets. RealView Debugger can support other vehicles, for example to communicate with the DSP Group Oak processors.

The connection entries in the board file have a corresponding entry in the Connection Control window. So, for the board file shown in Figure 1-1 on page 1-3, the Connection Control window in Figure 1-2 shows connections for ARMulator, Multi-ICE, and Remote_A.

To see these connection entries, you must have installed ADS, for ARMulator and Remote_A, and ARM Multi-ICE (Version 2.0 or later), for Multi-ICE. RealView Debugger does not include any connection software of its own.


Connection Control (My_user_name\rvdebug.brd)
Help
Target connection entries
Connect
Synch
The processors available using the connection

```
Name                 Description
⊟ARM-A-RR            ARM Ltd. RDI targets
  ⊞ARMulator           ARM instruction set simulator
  ⊟Multi-ICE           ARM JTAG debug interface (parallel po:
      ARM940T_0          ARM940T on localhost
      ARM920T_2          ARM920T on localhost
      ARM966E-…          ARM966E-S on localhost
  ⊞Remote_A            Angel debug protocol (serial port)
⊟Server              Connection Broker
  ⊞localhost           Simulator Broker
```

Figure 1-2 Connection entries in the Connection Control window

The Connection Control window uses the connection configurations listed in the board file to create a tree of possible connection vehicles and the processors that can be reached using them.

1.2.2 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 treated as flash memory.

Target configuration is possible using the Advanced_Information group that is found in almost all the main groups. For example, there is an Advanced_Information group in the CONNECTION=Multi-ICE entry of the ARM-A-RR group, and there also is one in the BOARD=AP entry of the (*.bcd) Board/Chip Definitions group.

However, to minimize the potential for mistakes, it is suggested that you only modify the Advanced_Information settings of the following group types:

- BOARD
- CHIP
- COMPONENT.

These groups names can be 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.

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. Configuration entries in files loaded into this group can be referenced from any other connection, which makes the target description independent of the connection used to access it, and makes it easier to use target descriptions in multiple instances of the debugger.

Note

In the board file, both target connection groups, for example CONNECTION=Multi-ICE, and target configuration groups, for example BOARD=AP, have an Advanced_Information group. Although you can use the Advanced_Information group of the target connection, it is suggested that you use target configuration groups and then reference these from the connection entry you are using.

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 installation that define standard ARM targets such as Integrator/AP. If you have ADS, Multi-ICE, or ARM ADI installed, configuration files are also created for the standard ARM RDI DLLs included with these products.

This section describes how the debugger install and home directories are located and how they are used to find the configuration files. It contains the following sections:

- *The install directory*
- *The home directory*
- *The RealView Debugger search path* on page 1-7
- *What the configuration files contain* on page 1-8
- *Saving and restoring connection properties* on page 1-9.

1.3.1 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:

1. The `-install` command line option:

   ```
   -install="D:\WinApp\RVDebug\”
   ```

2. The `$RVDEBUG_INSTALL` environment variable:

   ```
   set RVDEBUG_INSTALL=D:\WinApp\RVDebug\
   ```

3. The default location, for example: `C:\Program Files\ARM\RVD\`

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

The shortcuts that are installed on the Windows **Start** → **Programs** menu 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.

1.3.2 The home directory

RealView Debugger requires a debugger-specific home directory to store user-specific settings such as your board file. Information about the other files that are stored in the debugger home directory is in an appendix to the *RealView Debugger v1.6 Essentials Guide*.

The location of this 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`*`\RVDebug"`

   The home directory is `C:\Documents and Settings\`*`user_name`*`\RVDebug`

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\RVDebug`

3. You can use the `-install` command line option on its own or together with `-user` or `$USER`:

   `-install="D:\WinApp\RVDebug\" -user="MyTeam"`

   The home directory is `D:\WinApp\RVDebug\home\MyTeam\`.

   You can use `-user` or `$USER` to specify an alternative for `MyTeam`.

4. You can use the `$RVDEBUG_INSTALL` environment variable on its own or together with `-user` or `$USER`:

   `set RVDEBUG_INSTALL=D:\WinApp\RVDebug\`

   `set USER=MyTeam`

   The home directory is `D:\WinApp\RVDebug\home\MyTeam`.

5. The default location, for example *`install_directory`*`\RVD\Core\1.6.1\`*`release`*`\win_32-pentium\home\`*`user_name`*`\`

6. If the product is not found in any of these places, the debugger cannot find the files it requires and it terminates.

——— **Note** ———

If the debugger home directory location is defined, but the directory itself does not exist, the debugger creates it and copies the standard set of configuration files there from the system defaults in the default settings directory (`\etc`).

1.3.3 The RealView Debugger search path

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

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

2. The `RVDEBUG_SHARE` environment variable, if it is set.
3. The RealView Debugger home directory, described in *The home directory* on page 1-6, 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 Board/Chip definition files are found. Where two or more files with the same filename in more than one of the searched directories are found, the first file found is loaded and others are ignored.

1.3.4 What the configuration files contain

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

`*.brd`	Top-level board files. Normally this is `rvdebug.brd`. This file includes the filenames of the other configuration files. You change this file when you save settings in the Connection Properties window.
`*.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.
`*.rbe`	The board file that 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.
`*.bcd`	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.

There are other files that are stored in your debugger home directory. For more information about these files, see the appendix in *RealView Debugger v1.6 Essentials Guide*.

1.3.5 Saving and restoring connection properties

When you are configuring RealView Debugger, and especially when you are testing out worked examples, you are recommended to keep backups of known-good configuration information before changing settings. These are the backup systems you can use:

- you can rely on the per-file backups that RealView Debugger makes whenever it saves the Connection Properties window
- you can copy specific files or the whole home directory to a backup area.

Using the automatic backup files

RealView Debugger automatically renames any existing file named in *What the configuration files contain* on page 1-8 by adding a `.bak` file extension whenever the file is edited. 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 1-6 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 you have performed. These hints might help:

- If you have only changed RDI connection settings, for example by changing items in the Multi-ICE configuration dialog, 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.
- 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 everything, or you are not sure what to select, selecting all the files listed in *What the configuration files contain* on page 1-8 will restore the Connection Properties window to its original state.

Chapter 2
Connecting to Targets

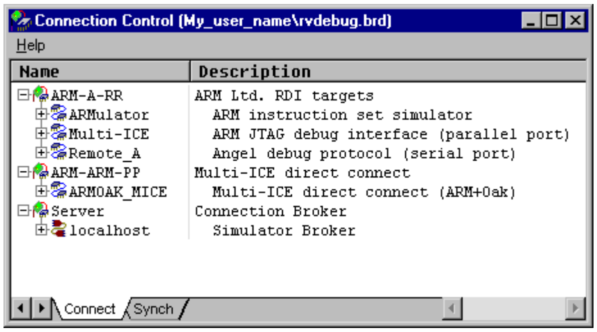
This chapter describes how the Connection Control window is used to view connection details and to configure new ones. It contains the following sections:

2.1 The Connection Control window

The Connection Control window enables you to make connections, change existing connections, and configure new ones if required.

You can display the Connection Control window, shown in Figure 2-1, 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.



Name	Description
ARM-A-RR	ARM Ltd. RDI targets
ARMulator	ARM instruction set simulator
Multi-ICE	ARM JTAG debug interface (parallel port)
Remote_A	Angel debug protocol (serial port)
ARM-ARM-PP	Multi-ICE direct connect
ARMOAK_MICE	Multi-ICE direct connect (ARM+Oak)
Server	Connection Broker
localhost	Simulator Broker

Figure 2-1 Connection Control window

If you are licensed to work with multiprocessor debug target systems using different processor families, the Connection Control window contains tabs, for example the **Synch** tab, shown in Figure 2-1. In single processor debugging mode, these tabs are not available.

If you are using the RTOS extension to work with multithreaded applications, the window might contain additional tabs not shown here. See the RTOS chapter in *RealView Debugger v1.6 Extensions User Guide*, for more details.

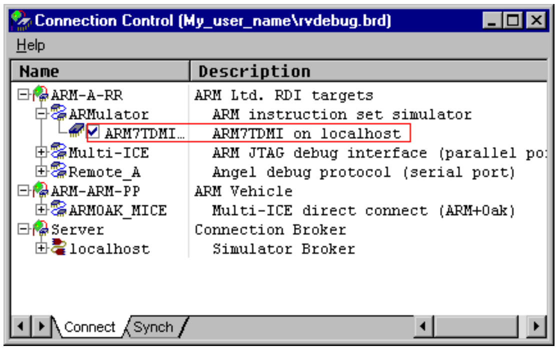
2.1.1 Using the Connection Control window

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 the example in Figure 2-1, this is the default file, stored in `\home\My_user_name\rvdebug.brd`, in the root installation.

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 -.

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 2-2 shows a Connection Control window after groups have been expanded. If a group is selected, a box is drawn around it.

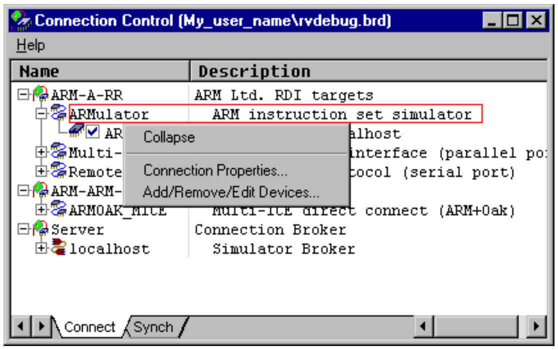

Connection Control (My_user_name\rvdebug.brd)

Help

Name	Description
ARM-A-RR	ARM Ltd. RDI targets
ARMulator	ARM instruction set simulator
ARM7TDMI…	ARM7TDMI on localhost
Multi-ICE	ARM JTAG debug interface (parallel po:
Remote_A	Angel debug protocol (serial port)
ARM-ARM-PP	ARM Vehicle
ARMOAK_MICE	Multi-ICE direct connect (ARM+Oak)
Server	Connection Broker
localhost	Simulator Broker

Figure 2-2 Expanding groups in the Connection Control window

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-A-RR`, and select **Expand Vehicles** from the context menu. To expand the second-level entries, right-click on an entry, for example `ARMulator`, and select **Expand**. To collapse them again, select **Collapse** from the context menu, shown in Figure 2-3.

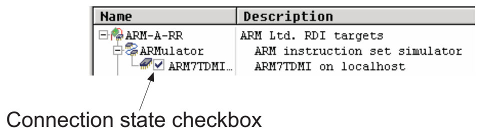

Connection Control (My_user_name\rvdebug.brd)

Help

Name	Description
ARM-A-RR	ARM Ltd. RDI targets
ARMulator	ARM instruction set simulator
AR	alhost
Multi-	interface (parallel po:
Remote	ocol (serial port)
ARM-ARM-	
ARMOAK_MICE	Multi-ICE direct connect (ARM+Oak)
Server	Connection Broker
localhost	Simulator Broker

Figure 2-3 Displaying the context menu for the ARMulator vehicle

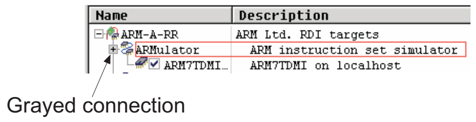
You can connect or disconnect by checking or unchecking the connection state check box shown in Figure 2-4 on page 2-4.



Name	Description
ARM-A-RR	ARM Ltd. RDI targets
ARMulator	ARM instruction set simulator
ARM7TDMI…	ARM7TDMI on localhost

Figure 2-4 The Connection Control check box

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, as shown for the `ARMulator` entry in Figure 2-5.



Name	Description
ARM-A-RR	ARM Ltd. RDI targets
ARMulator	ARM instruction set simulator
ARM7TDMI…	ARM7TDMI on localhost

Figure 2-5 Collapsed, active connection display

The Connection Control window shows available connections as defined by the target configuration settings. This is based on information about the available target processors as defined in the default configuration files. For example, the ARMulator configuration files, installed as part of the root installation, specify the `ARM7TDMI` core as the default processor.

You can connect to any of the default connections, shown in the Connection Control window, without making any further changes to configuration files. See *Connecting to many targets* on page 2-13 for details.

2.1.2 Groups in the Connection Control window

Connection and target details are displayed as a hierarchy, in the `Name` column of the Connection Control window:

Target Vehicles

Top-level groups are supported target vehicles as specified by the target configuration settings, for example `ARM-A-RR` for ARM RDI targets.

Access-provider connections

Second-level groups show the type of vehicle, or the debug target interface used to support the connection, for example `Multi-ICE`, the ARM JTAG debug tool for embedded systems, or `Remote_A`, the Angel debug monitor. The ARM RDI second-level groups are enabled in your board file when you install the ADS, ARM ADI, or ARM Multi-ICE software.

Endpoint connections

Third-level entries show the target processors that are made available by the access provider, for example the `ARM7TDMI` core being simulated by ARMulator, or the `ARM920T` core connected using Multi-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 2-4 on page 2-4. Where no connections have been made, the check boxes are blank.

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

2.2 Managing connections

Use context menus in the Connection Control window to:

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

There are several context menus that can be displayed, depending on the tree item you click on:

- *The vehicle menu*
- *The ARM RDI target menu* on page 2-7
- *The JTAG target menu* on page 2-8.

2.2.1 The vehicle menu

Right-click anywhere on a top-level tree name, for example the `ARM-A-RR` vehicle, to see the vehicle context menu shown in Figure 2-6.

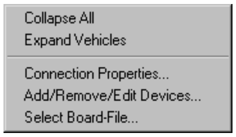


Figure 2-6 Vehicle menu

The options available from this menu are:

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.

Connection Properties...

Displays the Connection Properties window to amend current configuration details or to add target configurations.

Add/Remove/Edit Devices...

Only displayed for the `ARM-A-RR` vehicle, this item 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.

Select Board-File...

Displays the Select Board-File to Read dialog, where you can specify a new board file for target configuration in this session.

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

Disconnect All

Disconnects all connected targets and collapses the hierarchical tree to display the top-level and second-level entries.

2.2.2 The ARM RDI target menu

Right-click on a second-level entry within `ARM-A-RR`, for example `ARMulator` or `Multi-ICE`, to see the RDI target context menu, shown in Figure 2-7.

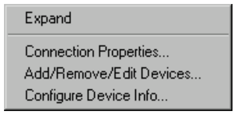


Figure 2-7 RDI target menu

The options available from this menu are:

Expand Expands the hierarchical tree to display all the second-level entries. This then changes to **Collapse**. Expanding the connection causes the RDI interface to be initialized and queried. This might:

- cause a short delay
- result in error messages being displayed if previously configured connection no longer operates because, for example, the JTAG cable is not connected
- result in the connection configuration dialog for the RDI software being displayed.

Connection Properties...

Displays the Connection Properties window, as for the vehicle menu.

Add/Remove/Edit Devices...

Enables you to add or remove RDI targets, as for `ARM-A-RR` above.

Configure Device Info...

Enables you to configure the debug target, for example by displaying the ARMulator Configuration dialog box where you can configure simulated debug targets.

For details on using these menu options to configure RDI targets, see Chapter 4 *Configuring Custom Connections*.

2.2.3 The JTAG target menu

This menu is displayed for connections that use *On-Chip Debugging* (OCD) JTAG-based connections.

Right-click on `ARMOAK_MICE`, to see the JTAG target context menu, shown in Figure 2-8.

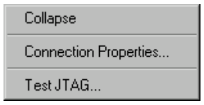


Figure 2-8 JTAG Target menu

The options available from this menu are:

Collapse Collapses the hierarchical tree to display only the top-level nodes. This then changes to **Expand**.

Connection Properties...

Displays the Connection Properties window to amend current configuration details or to add target configurations.

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.

2.3 Connecting to a target

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

- *Using the Connection Control window*
- *Including the connection in the workspace* on page 2-10
- *Using CLI commands* on page 2-10
- *Setting connect mode* on page 2-11.

If you are already connected to a target processor, in single processor debugging mode, making a new connection automatically disconnects the existing connection. The auto-disconnect does not occur until the new connection is successfully established so it is not necessary to disconnect yourself before making a new connection. Making multiprocessor connections is described in the *RealView Debugger v1.6 Extensions User Guide*.

2.3.1 Using the Connection Control window

Select **File** → **Connection** → **Connect to Target...** from the Code window main menu 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 entry.
- Select the check box for a required entry so that it is checked.
- Right-click on a connection entry and select **Connect** from the **Connection** context menu, shown in Figure 2-9 on page 2-10.
- Right-click on a connection entry and select **Connect (Defining Mode)...** from the **Connection** context menu, shown in Figure 2-9 on page 2-10.

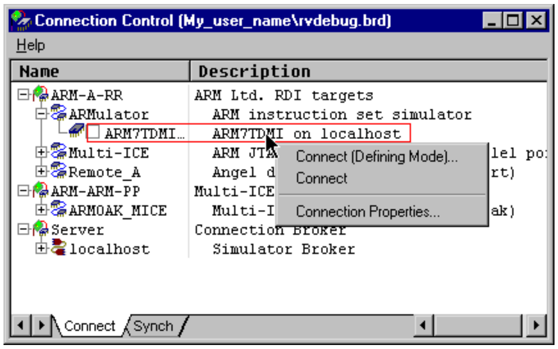
  ——— **Note** ———

  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 2-11.

  ————————

For example, to connect to an ARMulator model:

1. Display the Connection Control window.
2. Expand the top-level `ARM_A_RR` vehicle.
3. Expand the second-level `ARMulator` entry.
4. Select the processor connection, for example the default `ARM7TDMI_0`.


Connection Control (My_user_name\rvdebug.brd)
Help
Connect (Defining Mode)...
Connect
Connection Properties...
Connect
Synch

```
Name              Description
⊟ARM-A-RR         ARM Ltd. RDI targets
  ⊟ARMulator        ARM instruction set simulator
     ARM7TDMI...    ARM7TDMI on localhost
  ⊞Multi-ICE        ARM JTA                        lel po:
  ⊞Remote_A         Angel d                        rt)
⊟ARM-ARM-PP       Multi-ICE
  ⊞ARMOAK_MICE      Multi-I                        ak)
⊟Server           Connection Broker
  ⊞localhost        Simulator Broker
```

Figure 2-9 Connection menu

——— **Note** ———

When you connect to a Multi-ICE target with two or more processors, the order the processors appear in is neither alphabetical nor based on the TAP number.

———————————

If the chosen vehicle provides an RDI target that has not been configured, a dialog box is displayed to indicate that the connection has failed (see *Failing to make a connection* on page 2-14 for details). This enables you to configure the target and connect.

RealView Debugger connects to the specified target using the default connection mode for that target, unless you choose the **Connect (Defining Mode)...** option. You can, however, specify the connection mode to use, see *Setting connect mode* on page 2-11.

2.3.2 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.

2.3.3 Using CLI commands

The `CONNECT` and `RUN` commands can be used to make a connection to your debug target. For details on using these commands, see *RealView Debugger v1.6 Command Line Reference Guide*.

2.3.4 Setting connect mode

You can control the way a processor starts when you connect. 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 set the connect mode use either:

- the **Connect (Defining Mode)...** submenu of the **File** menu
- the `Advanced_Information` setting `Connect_mode`.

The connection mode that is used is defined by the `Connect_mode` setting in the `Advanced_Information` group in your board file unless you use the option **Connect (Defining Mode)...**. For more information about setting connect mode in the board file, see the description of the `Advanced_Information` block in Appendix A *Configuration Settings Reference*.

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

1. Right-click on a connection entry, for example `ARM7TDMI`, using `ARMulator`, to display the context menu, shown in Figure 2-9 on page 2-10.
2. Select **Connect (Defining Mode)...** to display the Connect Mode selection box shown in Figure 2-10.

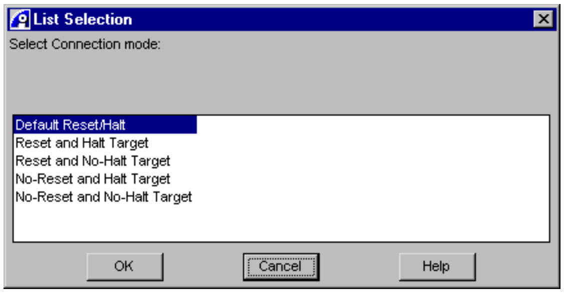


Figure 2-10 Connect Mode selection box

The state options shown depend on the target vehicle making the connection. 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.

Choose from this list to establish a connection and control the startup state of the target processor:

Default Reset/Halt

Submits a processor reset and halts any process currently running by issuing a `Stop` command. This is the default.

Reset and Halt Target

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

Reset and No-Halt Target

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

No-Reset and Halt Target

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

No-Reset and No-Halt Target

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

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

2.4 Connecting to many 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 automatically disconnects the previous connection.

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.

——— **Note** ———

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 is displayed.
3. Click the **Advanced** tab.
4. Ensure that the **Start-up with cache enabled** check box is not selected, and click **OK**.

———————————

Managing your connection options is described in *Using the Connection Control window* on page 2-9.

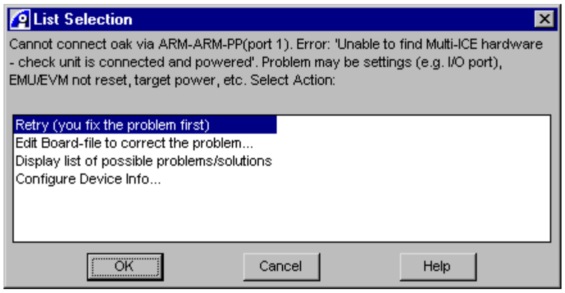
For details on making multiprocessor connections see the multiprocessing chapter in *RealView Debugger v1.6 Extensions User Guide*.

2.5 Failing to make a connection

Clicking inside a check box, in the Connection Control window, 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 RDI target has not been configured.
- 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 selection box to offer possible actions, shown in Figure 2-11. Some endpoint connections, such as Multi-ICE, might also display their own dialogs or messages.


Cannot connect oak via ARM-ARM-PP(port 1). Error: 'Unable to find Multi-ICE hardware - check unit is connected and powered'. Problem may be settings (e.g. I/O port), EMU/EVM not reset, target power, etc. Select Action:

Retry (you fix the problem first)
Edit Board-file to correct the problem...
Display list of possible problems/solutions
Configure Device Info...

Figure 2-11 Failing to make a connection

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.

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

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.

Configure Device Info...

Select this option to configure the target. If you are accessing an RDI target for the first time, for example Multi-ICE, it must be configured before it can be used.

The error message displayed at the top of the window indicates the type of error encountered by RealView Debugger. To close the error message and abandon the connection, click **Cancel**.

2.5.1 Troubleshooting

This section describes how to identify and fix some problems you might encounter:

- 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.

  You can use either the Connection Control window check boxes or the CLI `DISCONNECT` and `CONNECT` commands.

- 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 3-46 for information describing how to recover from this situation.

2.6 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 desired state of the processor, or process, following disconnection.

If you are already connected to a debug target processor, in single processor debugging mode, making a new connection automatically disconnects the existing connection. The auto-disconnect does not occur until the new connection is successfully established so it is not necessary to disconnect yourself before making a new connection.

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.

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 disconnecting during multiprocessor debugging sessions see the multiprocessing chapter in the *RealView Debugger v1.6 Extensions User Guide*.

The disconnection options available are:

- *Using the File menu*
- *Using the Connection Control window* on page 2-17
- *Using the CLI* on page 2-18
- *Disconnecting by exiting* on page 2-18
- *Setting disconnect mode* on page 2-19.

2.6.1 Using the File menu

If you are connected to a single debug target processor, you can disconnect from the current connection. Select **File** → **Connection** → **Disconnect** from the Code window main menu. 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.

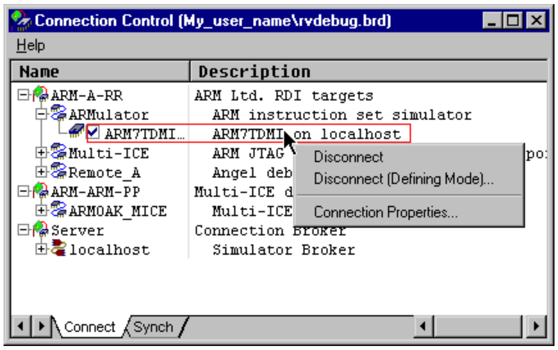
2.6.2 Using the Connection Control window

At any point in your debugging session, you can disconnect from a target using the Connection Control window. This can be done in different 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 2-12
- right-click on a connection entry and select **Disconnect (Defining Mode)...** from the **Disconnection** context menu, shown in Figure 2-12.

  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* on page 2-19.


Connection Control (My_user_name\rvdebug.brd)
Help
Disconnect
Disconnect (Defining Mode)...
Connection Properties...
Connect
Synch

Name	Description
ARM-A-RR	ARM Ltd. RDI targets
ARMulator	ARM instruction set simulator
ARM7TDMI...	ARM7TDMI on localhost
Multi-ICE	ARM JTAG
Remote_A	Angel deb
ARM-ARM-PP	Multi-ICE d
ARMOAK_MICE	Multi-ICE
Server	Connection Broker
localhost	Simulator Broker

Figure 2-12 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 target using the default disconnect mode for that target, unless you use the **Disconnect (Defining Mode)...** option. You can, however, specify the connection mode to use, see *Setting disconnect mode* on page 2-19.

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

2.6.3 Using the CLI

You can disconnect a connection using the CLI command `DISCONNECT`. This also enables you to specify the disconnection mode. See the *RealView Debugger v1.6 Command Line Reference Guide* for more information.

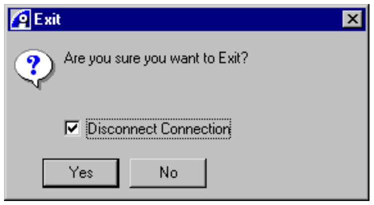
2.6.4 Disconnecting by exiting

Exiting the debugger with a connection active causes details of the connection to be stored in the current workspace. However, although saving the workspace on exit is the default behavior, you might have changed this. See the chapter describing configuring workspaces in *RealView Debugger v1.6 User Guide* for more details. When you exit, the debugger prompts you to make sure you want to disconnect, as described in *Disconnection confirmation*.

When RealView Debugger starts up with a workspace that includes stored connection information, it will try to reconnect. If this fails, you are prompted for the next action, as described in *Reconnecting stored connections* on page 2-19.

Disconnection confirmation

If you exit the debugger with active connections, the debugger asks whether these connections can be disconnected, shown in Figure 2-13.


Exit

Are you sure you want to Exit?

☑ Disconnect Connection

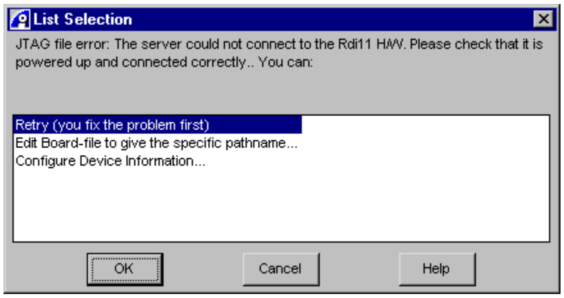
Figure 2-13 Disconnect confirmation

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 redisplays the connection.

If you do disconnect when you exit 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 2-14. Select **Configure Device Information...** from the list and click **OK** to reconfigure the connection. Click **Cancel** to abort the connection attempt.


List Selection

JTAG file error: The server could not connect to the Rdi11 H/W. Please check that it is powered up and connected correctly.. You can:

Retry (you fix the problem first)
Edit Board-file to give the specific pathname...
Configure Device Information...

Figure 2-14 Disconnect reconfiguration or retry

2.6.5 Setting disconnect mode

You can control the way a processor is left when you disconnect. This is useful when debugging multiprocessor debug target systems or multithreaded applications, and 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.

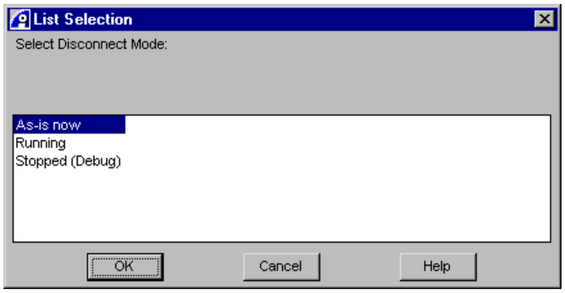
To set the disconnect mode use either:

- the **Disconnect (Defining Mode)...** submenu of the **File** menu
- the `Advanced_Information` setting `Disconnect_mode`.

The disconnection mode that is used is defined by the `Disconnect_mode` setting in the `Advanced_Information` group in your board file unless you use the option **Disconnect (Defining Mode)...**. For more information about setting disconnect mode in the board file, see the description of the `Advanced_Information` block in Appendix A *Configuration Settings Reference*.

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

1. Right-click on a connection entry to display the **Disconnection** context menu, shown in Figure 2-12 on page 2-17.
2. Select **Disconnect (Defining Mode)...** to display the Disconnect Mode selection box, shown in Figure 2-15.


List Selection

Select Disconnect Mode:

As-is now
Running
Stopped (Debug)

Figure 2-15 Disconnect Mode selection box

The state options shown depend on the target vehicle handling the connection. If you select an option that is not supported by your target processor, a warning is displayed to show that the chosen mode is invalid and is being ignored.

Choose from the list to close a connection and leave the target processor in one of these states:

Running (Debug)

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

Free Running

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

Running This leaves the processor running, with the state of breakpoints defined by the vehicle in use. This means the program might enter debug state after the debugger has disconnected, depending on the code paths the program takes.

As-is now This leaves the processor in the current state, whether stopped or running, and with the state of any breakpoints unchanged.

Stopped (Debug)

The processor is left stopped.

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.

Chapter 3
Configuring Custom Targets

This chapter describes 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:

3.1 About target configuration

RealView Debugger works in conjunction with either a hardware or a software debug target. An ARM development board, communicating through Multi-ICE, is an example of a hardware debug target system. ARMulator is an example of a software debug target system. This section provides an introduction to target configuration. It contains the following sections:

- *Target configuration*
- *Configuration files*
- *Default configuration files* on page 3-3
- *Using other board files* on page 3-5.

3.1.1 Target configuration

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:

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

Default configuration files are supplied as part of the RealView Debugger installation. If you have ADS installed, these are set up for you to make a connection to the supported target connection vehicles.

RealView Debugger creates a personal home directory for you, as described in *Configuration files* on page 1-6, containing default configuration files.

How the configuration files are linked together

The board file references several other configuration files, for example the RDI definitions, `*.rbe`, and the Board/Chip definition files, `*.bcd`., to form the complete configuration. This is shown in Figure 3-1 on page 3-3.

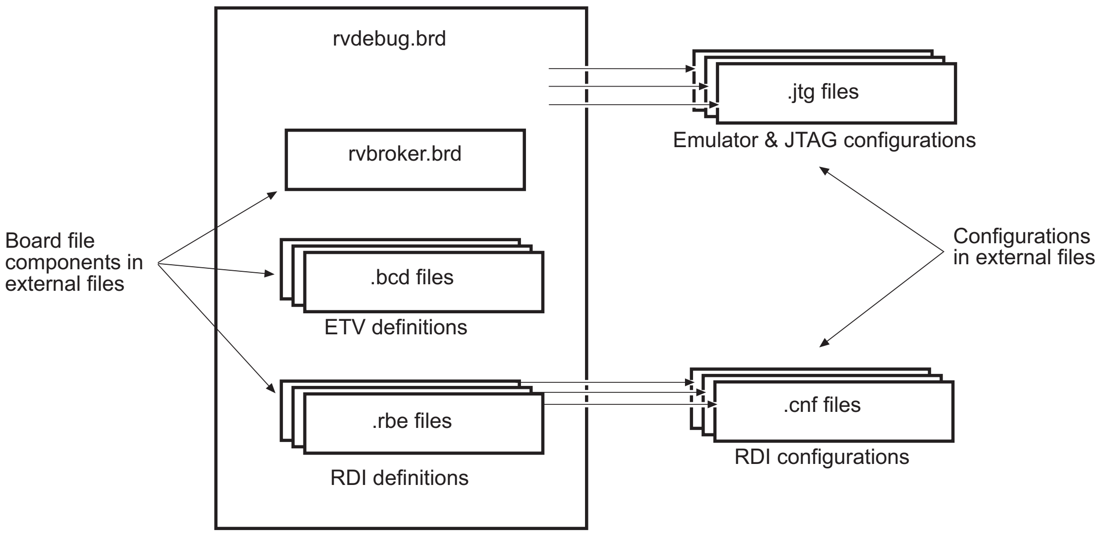


Figure 3-1 Board file configuration file structure

The 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.

3.1.3 Default configuration files

The debug target configuration settings are maintained through the use of a hierarchy of configuration files:

- *Board file*
- *RDI configuration files* on page 3-4
- *JTAG files* on page 3-4
- *Board/Chip definition files* on page 3-5.

Board file

RealView Debugger uses a board file to access information about the debugging environment and the debug targets available to you. 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.

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 RealView 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.

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 files	There is one `.rbe` file for each RDI target available for connection.
.cnf files	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*.

JTAG 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.

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, or the ARMulator simulator, JTAG files are replaced by the `.cnf` configuration files.

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. 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 `CM966ES.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 need to edit these files but you can specify their location so that they are included in the configuration settings. However, where changes are required, use the Connection Properties window to make the necessary changes.

3.1.4 Using other board files

The default board file, `rvdebug.brd`, can be used as a template to create additional board files. RealView Debugger can only access one board file at a time but you can specify which `.brd` file to use for a particular debugging session. You can use this facility to share a configuration between developers.

3.2 The supplied target descriptions

This section describes the target board descriptions that are supplied with RealView Debugger. These descriptions can be used with the associated target without further modification. Reference them from the target connection, see *Linking a board, chip, or component to a connection* on page 3-12 for 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 3-8 for details of creating your own configurations.

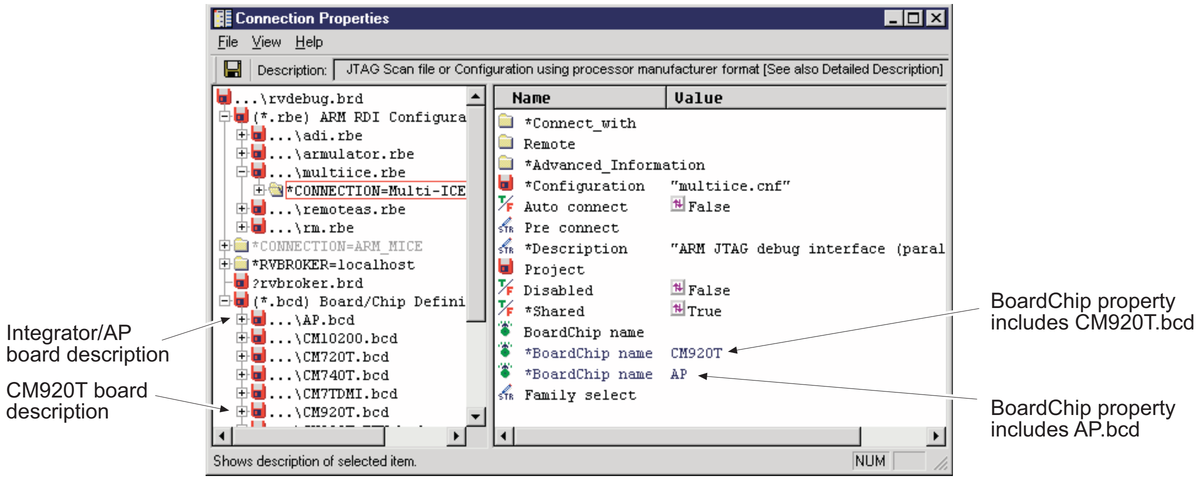
The supplied descriptions include:

`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 the Integrator/CM platform.
`Eval7T.bcd`	A description of the ARM Evaluator-7T registers and memory map, including a description of the KS32C50100 processor internal registers.
`CM7TDMI.bcd`	A description of the ARM CM7TDMI processor core module registers and memory map.
`CM720T.bcd`	A description of the ARM CM720T 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.
`CM920_ETM.bcd`	A description of the ARM CM920T-ETM processor core module registers and memory map.
`CM940T.bcd`	A description of the ARM CM920T processor core module registers and memory map.

`CM966ES.bcd`	A description of the ARM CM966E-S processor core module registers and memory map.
`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.

Note

If you are using an Integrator/AP, Integrator/CM, or Integrator/CP motherboard with a core module, you can combine the platform and core module descriptions by using multiple `BoardChip_name` assignments in the `CONNECTION` section of the board file, shown in Figure 3-2.


Connection Properties
File View Help
Integrator/AP board description
CM920T board description
BoardChip property includes CM920T.bcd
BoardChip property includes AP.bcd
NUM

Description: JTAG Scan file or Configuration using processor manufacturer format [See also Detailed Description]

```
...\rvdebug.brd
(*.rbe) ARM RDI Configura
  ...\adi.rbe
  ...\armulator.rbe
  ...\multiice.rbe
    *CONNECTION=Multi-ICE
  ...\remoteas.rbe
  ...\rm.rbe
*CONNECTION=ARM_MICE
*RVBROKER=localhost
?rvbroker.brd
(*.bcd) Board/Chip Defini
  ...\AP.bcd
  ...\CM10200.bcd
  ...\CM720T.bcd
  ...\CM740T.bcd
  ...\CM7TDMI.bcd
  ...\CM920T.bcd
```

Name	Value
*Connect_with	
Remote	
*Advanced_Information	
*Configuration	"multiice.cnf"
Auto connect	False
Pre connect	
*Description	"ARM JTAG debug interface (paral
Project	
Disabled	False
*Shared	True
BoardChip name	
*BoardChip name	CM920T
*BoardChip name	AP
Family select	

Shows description of selected item.

Figure 3-2 Connection Properties window, showing use of BoardChip_name setting

You can 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 Integrator CM920T processor core module with Multi-ICE, you modify the Multi-ICE `CONNECTION` setting `BoardChip_name` to reference the `CM920T` description. For further instructions on doing this, see *Linking a board, chip, or component to a connection* on page 3-12.

3.3 Creating new target descriptions

This section describes how to create variations of existing configurations and new target descriptions. Creating new target descriptions provides these advantages:

- with 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
- with definitions of 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.

Creating new target descriptions involves these steps:

1. Create a `BOARD`, `CHIP`, or `COMPONENT` group for the configuration. This requires the following steps:
   a. *Creating a *.bcd file* on page 3-9, to store the group.
   b. *Creating and naming a board, chip, or component* on page 3-11.
2. Define the target using the configuration items in the group. This is described in *Example descriptions* on page 3-20.
3. Link the new target definition to the `CONNECTION` that the target uses. This is described in *Linking a board, chip, or component to a connection* on page 3-12.

Note

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

3.3.1 Saving and restoring your .brd file

In these examples, you are changing your board file. This is stored in your RealView Debugger home directory, for example `\home\`*`user_name`*`\`, where *`user_name`* is your Windows login or user name. Target configuration files are also stored in this directory, for example `.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 1-6 for instructions on making backups of your configuration
- *Restoring your .brd file* on page 3-45 for instructions on restoring a default configuration

- *Troubleshooting* on page 3-46 for instructions on recovering from an incorrectly configured debugger home directory, whether or not you have a backup.

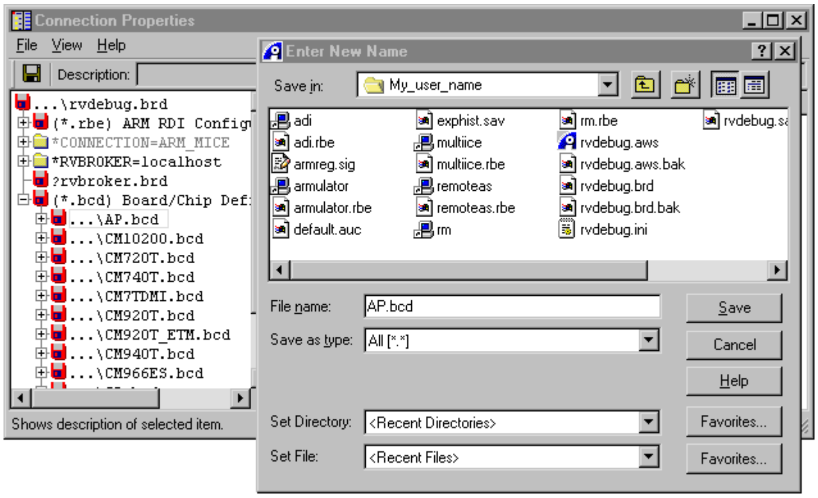
3.3.2 Creating a *.bcd file

To create a new `*.bcd` file, you must copy one of the existing files. You can do this in Windows Explorer or from within RealView Debugger.

To copy the file in Windows Explorer, use the Windows copy commands in the normal way. Start RealView Debugger and display the Connection Properties window. Resume at step 8 of the following procedure.

To copy a `*.bcd` file from within the debugger:

1. Select **File** → **Connection** → **Connection Properties...** to display the Connection Properties window.
2. Expand the group `(*.bcd) Board/Chip Definitions` to show the current list of target descriptions.
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 shown in Figure 3-3.



```
...\rvdebug.brd
(*.rbe) ARM RDI Config
*CONNECTION=ARM_MICE
*RVBROKER=localhost
?rvbroker.brd
(*.bcd) Board/Chip Def
...\AP.bcd
...\CM10200.bcd
...\CM720T.bcd
...\CM740T.bcd
...\CM7TDMI.bcd
...\CM920T.bcd
...\CM920T_ETM.bcd
...\CM940T.bcd
...\CM966ES.bcd
```

Save in: My_user_name

adi	exphist.sav	rm.rbe	rvdebug.sa
adi.rbe	multiice	rvdebug.aws	
armreg.sig	multiice.rbe	rvdebug.aws.bak	
armulator	remoteas	rvdebug.brd	
armulator.rbe	remoteas.rbe	rvdebug.brd.bak	
default.auc	rm	rvdebug.ini	

File name: AP.bcd

Save as type: All [*.*]

Set Directory: <Recent Directories>

Set File: <Recent Files>

Figure 3-3 Saving an existing *.bcd file with a new name

By default, `*.bcd` files are saved in your default settings directory `\etc`. Locate your RealView Debugger home directory to save the new file.

5. Enter the new filename, for example `AM.bcd`. You must use the `.bcd` file extension.

6. Click **Save**. The dialog 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 new `.bcd` file group by clicking on the ⊞ icon. Right-click on the contents (for example, a `BOARD` entry), to display the context menu shown in Figure 3-4.

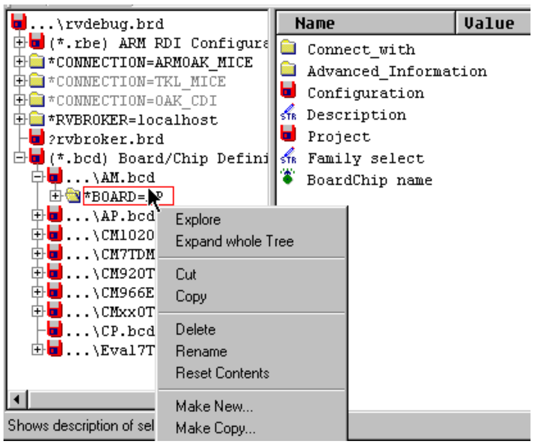


Figure 3-4 Deleting the original contents of the copied file

9. Select **Delete** from the context menu to delete the `BOARD` (or other groups) in the file. By deleting and then recreating the group, you avoid problems caused by old and inappropriate settings.

10. Select **File** → **Save and Close**.

Note

The general layout and controls of the RealView Debugger settings windows are described in the online help topic *Changing Settings*.

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 *.bcd file* on page 3-9, 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 use. It is however recommended that you use them as follows so that it is clear what the group describes:

`BOARD`	Target boards as a whole, for example, the Evaluator-7T, or the Integrator/AP motherboard, including the effect of glue logic implementing memory maps and small peripheral components.
`CHIP`	Significant devices on a target board, especially where you might use the device on more than one board, or where the device is in itself complex. For example, in the supplied target descriptions, the Evaluator-7T `BOARD` references the KS32C50100 `CHIP` to define the processor and ASIC components of that device.
`COMPONENT`	Other components not covered by the above.

To create a group:

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

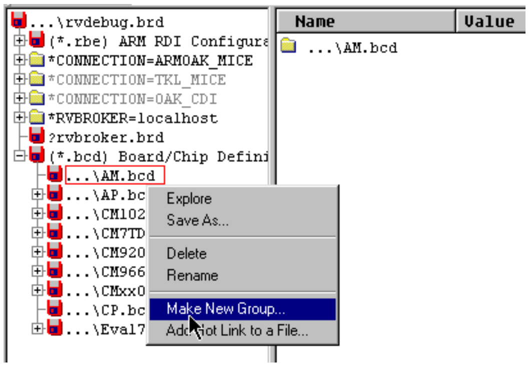


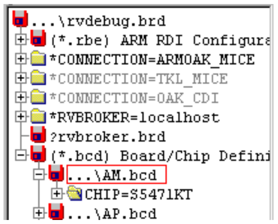
Figure 3-5 Adding a new group to a *.bcd file

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

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. It is initially displayed collapsed, so you must click on the ⊞ icon to display the group, shown in Figure 3-6.



```
...\rvdebug.brd
  (*.rbe) ARM RDI Configura
  *CONNECTION=ARMOAK_MICE
  *CONNECTION=TKL_MICE
  *CONNECTION=OAK_CDI
  *RVBROKER=localhost
  ?rvbroker.brd
  (*.bcd) Board/Chip Defini
    ...\AM.bcd
      CHIP=S5471KT
    ...\AP.bcd
```

Figure 3-6 Viewing the new group in the *.bcd file

6. Select **File** → **Save and Close**.

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 you reference it from a connection. There are several cases to consider, presented here in order of increasing complexity:

- *Linking one board group to one processor connection*
- *Linking several board groups to one processor connection* on page 3-14
- *Linking one or more board groups to another board group* on page 3-16
- *Linking one or more board groups to multiple processor connections* on page 3-18.

Linking one board group to one processor connection

This configuration is shown as in the form of a tree in Figure 3-7.

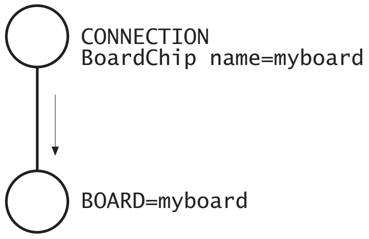
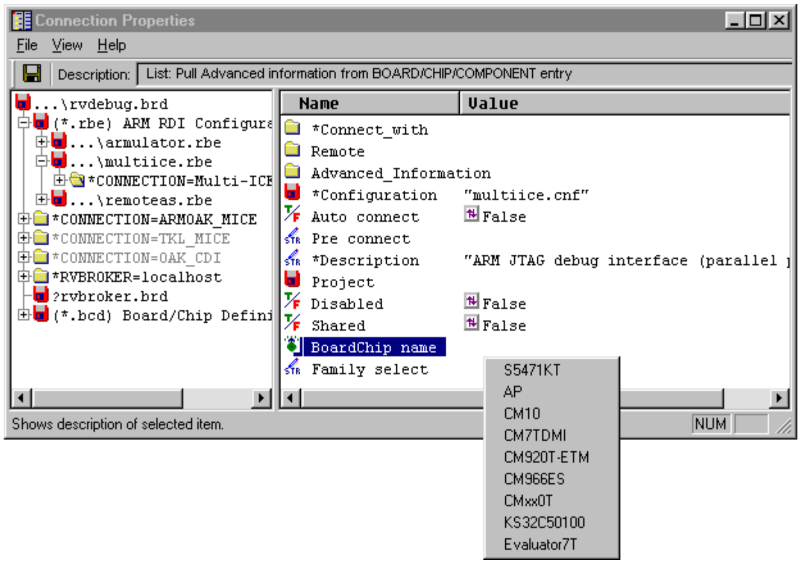


Figure 3-7 Linking one connection to one board

To link to this you must first ensure that the `*.bcd` file contains the required `BOARD`, `CHIP`, or `COMPONENT` groups. Then:

1. In the Connection Properties window, expand the connection that you are using. For example, if you are using a Multi-ICE interface, expand these entries:
   a. `(*.rbe) ARM RDI Configuration Entries`
   b. `...\multiice.rbe`
2. Click on `CONNECTION` within the entry you selected in step 1b. The right pane displays a set of properties including `BoardChip_name`.
3. Left-click on the `BoardChip_name`. It becomes highlighted.
4. Left-click on the `BoardChip_name` again. This displays a context menu including a list of available `.bcd` files, shown in Figure 3-8.



Description: List: Pull Advanced information from BOARD/CHIP/COMPONENT entry

```
...\rvdebug.brd
(*.rbe) ARM RDI Configura
  ...\armulator.rbe
  ...\multiice.rbe
    *CONNECTION=Multi-ICE
  ...\remoteas.rbe
*CONNECTION=ARMOAK_MICE
*CONNECTION=TKL_MICE
*CONNECTION=OAK_CDI
*RVBROKER=localhost
?rvbroker.brd
(*.bcd) Board/Chip Defini
```

Name	Value
*Connect_with	
Remote	
Advanced_Information	
*Configuration	"multiice.cnf"
Auto connect	False
Pre connect	
*Description	"ARM JTAG debug interface (parallel
Project	
Disabled	False
Shared	False
BoardChip name	
Family select	

Figure 3-8 Linking a board

Note

The two clicks in steps 3 and 4 must be distinct. A double-click does not work.

5. Select the name of the board group, for example the `CHIP` called `S5471KT` that was created in *Creating and naming a board, chip, or component* on page 3-11.
6. Select **File** → **Save and Close**.

7. Restart the debugger. When you connect using Multi-ICE, the configuration defined in your board group is applied to the connection.

Linking several board groups to one processor connection

You might want to link several groups to a single processor 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.

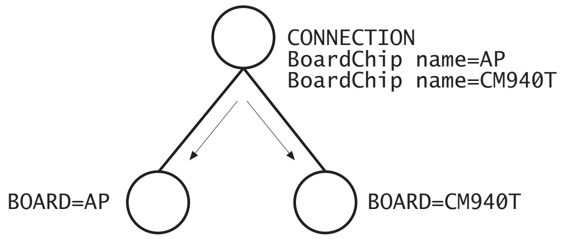


Figure 3-9 Linking one connection to two boards

When you reference multiple boards, RealView Debugger merges the settings from each group in a breadth-first search of the group tree. Therefore the complete configuration is the combined configurations of all of the 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 do this you must first ensure that the `*.bcd` files exist and then reference them from your board file using the required `BOARD`, `CHIP`, or `COMPONENT` groups:

1. In the Connection Properties window, expand the connection that you are using. For example, if you are using a Multi-ICE interface, expand these entries:
   a. `(*.rbe) ARM RDI Configuration Entries`
   b. `...\multiice.rbe`
2. Click on `CONNECTION` within the entry you selected in step 1b, so that it is highlighted. The right pane displays a set of properties including `BoardChip_name`.
3. Left-click on the `BoardChip_name` to highlight it.
4. Left-click on the `BoardChip_name` once more. A context menu is displayed, shown in Figure 3-8 on page 3-13.

   ——— **Note** ———

   The two clicks in steps 3 and 4 must be distinct. A double-click does not work.

   ————————————

5. Select the name of the board group. A new entry is displayed in the right pane with an asterisk * beside it. For example, `*BoardChip_name CM940T`.

6. Right-click on the `BoardChip_name` that does not have an asterisk to see the context menu, shown in Figure 3-10.

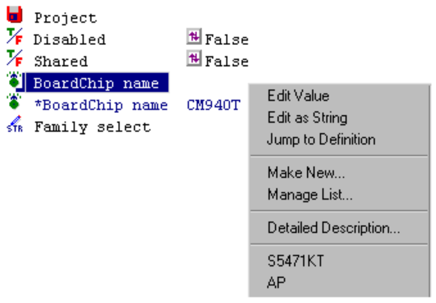


Figure 3-10 Linking a second board

7. Select the name of the board group. A new entry is added to the right pane with an asterisk * that indicates this value is not a default, for example, `*BoardChip_name AP`.

8. Select **File** → **Save and Close**.

9. Restart the debugger. When you connect using Multi-ICE, the configuration defined in your board group is applied to the connection.

You can repeat steps 6, 7, and 8 until all of the groups you require are included.

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 3-14 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, select the context menu as described above and click on **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.

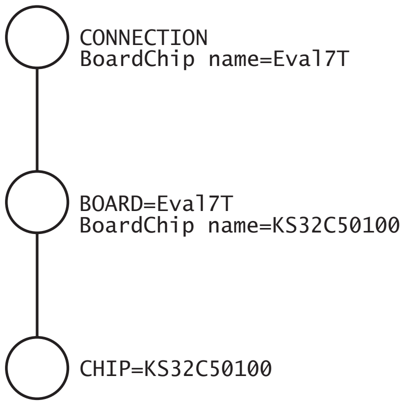


Figure 3-11 Linking one board into another board

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.

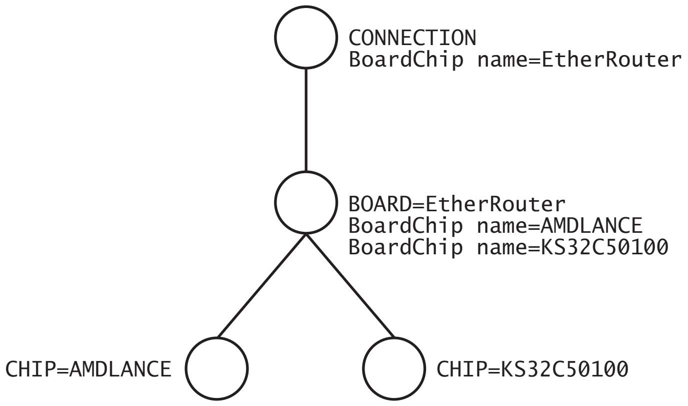


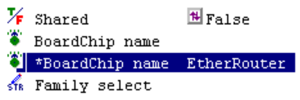
Figure 3-12 Linking one board into other boards

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 can help you later on because it is then easier to share descriptions or reuse a description for another project.

To create the structure shown in Figure 3-12 on page 3-16 you must first ensure that the *.bcd files exist and then reference them from your board file using the required BOARD, CHIP, or COMPONENT groups:

1. In the Connection Properties window, expand the connection that you are using. For example, if you are using a Multi-ICE interface, expand these entries:
   a. `(*.rbe) ARM RDI Configuration Entries`
   b. `...\multiice.rbe`
2. Click on CONNECTION within the entry you selected in step 1b, so that it is highlighted. The right pane displays a set of properties including BoardChip_name.
3. Right-click on the BoardChip_name. A context menu is displayed, shown in Figure 3-10 on page 3-15.
4. Select the name of the board group. A new entry is added to the right pane with an asterisk * beside it, shown in Figure 3-13.



```
Shared             False
BoardChip name
*BoardChip name    EtherRouter
Family select
```

Figure 3-13 Setting up EtherRouter

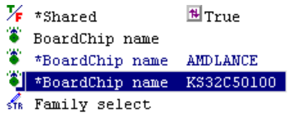
5. In the left pane of the Connection Properties window, expand the group for the board that is inheriting information from the CHIPs. For example, expand:
   a. `(*.bcd) Board/Chip Definitions`
   b. `...\ether.bcd`

   where you have created the file ether.bcd to contain the EtherRouter BOARD definition, as described in *Creating a *.bcd file* on page 3-9.
6. Click on BOARD within the *.bcd file you selected in step 5b, so that it is highlighted. The right pane displays a set of properties including BoardChip_name.
7. Right-click on the BoardChip_name that does not have an asterisk to see the context menu, shown in Figure 3-10 on page 3-15.

8. 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.

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

10. 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.



```
*Shared          True
BoardChip name
*BoardChip name  AMDLANCE
*BoardChip name  KS32C50100
Family select
```

Figure 3-14 Setting up EtherRouter

11. Select **File** → **Save and Close**.

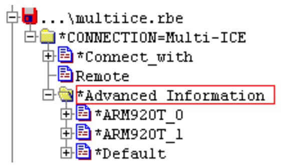
12. Restart the debugger. When you connect using Multi-ICE, the configuration defined in your board group is applied to the connection.

Linking one or more board groups to multiple processor connections

If you want to use the 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, Multi-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, Multi-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 3-19, 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.



```
...\multiice.rbe
  *CONNECTION=Multi-ICE
    *Connect_with
    Remote
    *Advanced_Information
      *ARM920T_0
      *ARM920T_1
      *Default
```

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 v1.6 Extensions User Guide*.

3.4 Example descriptions

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

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 with the debugger connected to a target.
- See *Configuration files* on page 1-6 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*
- *Configuring a memory map* on page 3-26
- *Setting up a custom register* on page 3-28
- *Setting up memory blocks* on page 3-32
- *Setting top of memory and stack heap values* on page 3-35
- *Using RealMonitor* on page 3-38
- *Flash programming* on page 3-43.

This section also includes:

- *Restoring your .brd file* on page 3-45 for instructions on restoring your factory settings
- *Troubleshooting* on page 3-46 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 Multi-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 Multi-ICE server*
2. *Configuring the new target*
3. *Connecting to the new target* on page 3-24
4. *Viewing the new target definition* on page 3-25.

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. 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.

   Note

   - Multi-ICE Release 1.4 works with RealView Debugger with some limitations, for example it does not support multiple simultaneous connections. See *Configuring RDI targets* on page 4-8 for more information. For best results use Multi-ICE Version 2.0 or later.
   - See the *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 `ARM-A-RR` vehicle entry and select **Add/Remove/Edit Devices...** from the context menu (Figure 3-16 on page 3-22).

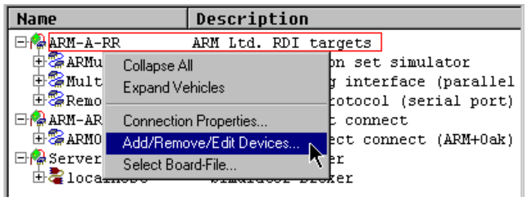
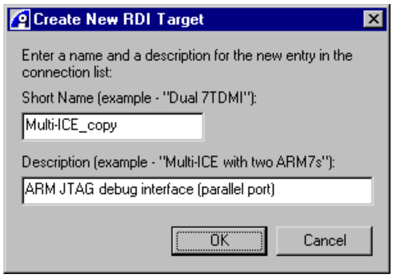


Figure 3-16 The ARM-A-RR vehicle context menu

4. Click in the `Description` column for Multi-ICE in the RDI target List dialog, to select it. The **Duplicate** button is enabled.

5. Click **Duplicate** to display the Create New RDI Target dialog (Figure 3-17).



Create New RDI Target

Enter a name and a description for the new entry in the connection list:

Short Name (example - "Dual 7TDMI"):

Multi-ICE_copy

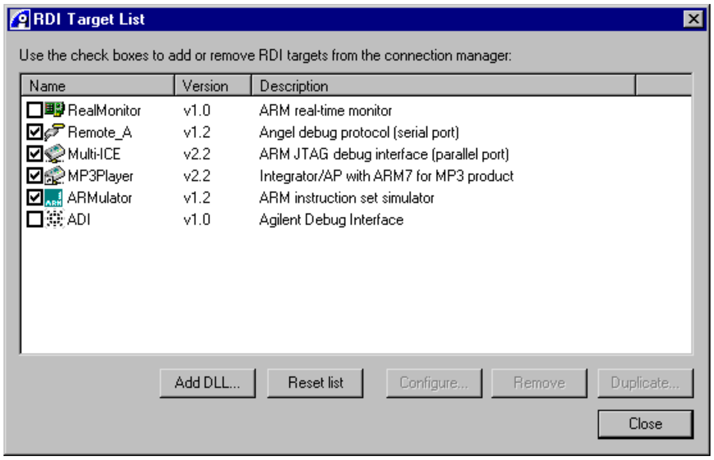
Description (example - "Multi-ICE with two ARM7s"):

ARM JTAG debug interface (parallel port)

Figure 3-17 Creating a new RDI Target connection

6. Edit the Short Name... and Description... fields as required. For example, enter `MP3Player` as the name and `Integrator/AP with ARM7 for MP3 product` as the description.

7. Click **OK**. The RDI Target List dialog now looks like Figure 3-18 on page 3-23. The contents of this window depend on the software you have installed.


RDI Target List

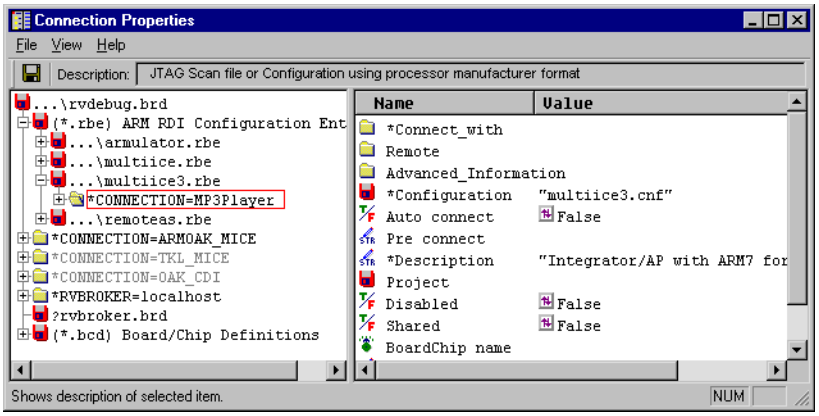
Use the check boxes to add or remove RDI targets from the connection manager:

Name	Version	Description
☐ RealMonitor	v1.0	ARM real-time monitor
☑ Remote_A	v1.2	Angel debug protocol (serial port)
☑ Multi-ICE	v2.2	ARM JTAG debug interface (parallel port)
☑ MP3Player	v2.2	Integrator/AP with ARM7 for MP3 product
☑ ARMulator	v1.2	ARM instruction set simulator
☐ ADI	v1.0	Agilent Debug Interface

Figure 3-18 The RDI Target List with a new connection

8. Click **Close**. 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 with the new connection expanded (Figure 3-19).


Connection Properties

Description: JTAG Scan file or Configuration using processor manufacturer format

```
...\rvdebug.brd
(*.rbe) ARM RDI Configuration Ent
  ...\armulator.rbe
  ...\multiice.rbe
  ...\multiice3.rbe
    *CONNECTION=MP3Player
  ...\remoteas.rbe
*CONNECTION=ARMOAK_MICE
*CONNECTION=TKL_MICE
*CONNECTION=OAK_CDI
*RVBROKER=localhost
?rvbroker.brd
(*.bcd) Board/Chip Definitions
```

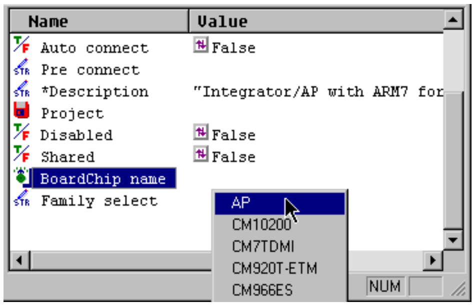
Name	Value
*Connect_with	
Remote	
Advanced_Information	
*Configuration	"multiice3.cnf"
Auto connect	False
Pre connect	
*Description	"Integrator/AP with ARM7 for
Project	
Disabled	False
Shared	False
BoardChip name	

Shows description of selected item.

Figure 3-19 The MP3Player connection properties

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

11. Click **AP** to select the Integrator/AP description (Figure 3-20 on page 3-24). A new entry `*BoardChip_name AP` is added to the pane.

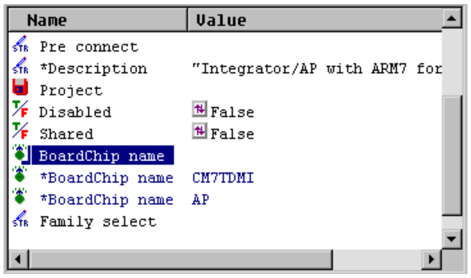


Name	Value
Auto connect	False
Pre connect	
*Description	"Integrator/AP with ARM7 for
Project	
Disabled	False
Shared	False
BoardChip name	
Family select	

Figure 3-20 The new RDI Target connection

12. Click on `BoardChip_name` (not on `*BoardChip_name`). The context menu is displayed again.

13. Click on **CM7TDMI** to select the ARM7TDMI core module description. The Connection Properties window now shows two `BoardChip_name` settings, shown in Figure 3-21.



Name	Value
Pre connect	
*Description	"Integrator/AP with ARM7 for
Project	
Disabled	False
Shared	False
BoardChip name	
*BoardChip name	CM7TDMI
*BoardChip name	AP
Family select	

Figure 3-21 The new RDI Target connection

14. Select **File** → **Save and Close** to save the new settings 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.

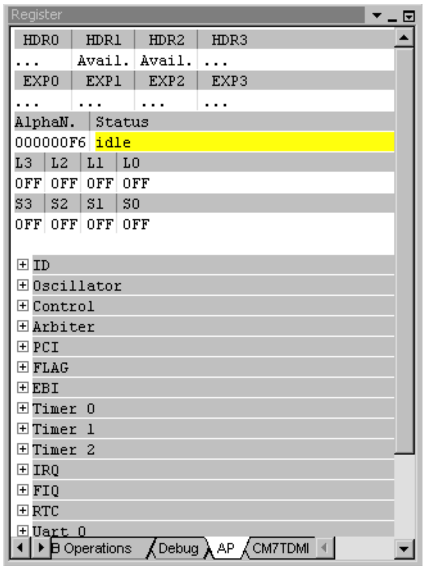
If a List Selection dialog appears, you need to configure Multi-ICE before continuing. Select **Configure Device Information...**, and click **OK**. The interface for configuring Multi-ICE is displayed. Ensure the details are correct, click **OK**. The Connection Control window is displayed, and you can now expand the new Multi-ICE entry.

3. Click on the processor entry under the new Multi-ICE 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 **CM7TDMI**.
2. Click on the **AP** tab. RealView Debugger shows the abstraction of the hardware information specific to the Integrator/AP board, shown in Figure 3-22.



```
 HDR0   HDR1   HDR2   HDR3
...    Avail. Avail. ...
 EXP0   EXP1   EXP2   EXP3
...    ...    ...    ...
AlphaN.  Status
000000F6 idle
L3  L2  L1  L0
OFF OFF OFF OFF
S3  S2  S1  S0
OFF OFF OFF OFF

⊞ID
⊞Oscillator
⊞Control
⊞Arbiter
⊞PCI
⊞FLAG
⊞EBI
⊞Timer 0
⊞Timer 1
⊞Timer 2
⊞IRQ
⊞FIQ
⊞RTC
⊞Uart 0
```

Figure 3-22 AP tab in the Register pane

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 **CM7TDMI** 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 monitoring execution chapter in *RealView Debugger v1.6 User Guide*.

5. In the Output pane at the bottom of the Code window, click on the **Log** tab. The display includes the line `Using BoardChips: AP, 7TDMI`, 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 3-43).

3.4.2 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. To do this:

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

2. Expand the following entries in turn:

   a. `(*.rbe) ARM RDI Configuration Entries`

   b. `...\armulator.rbe` *(change as required)*

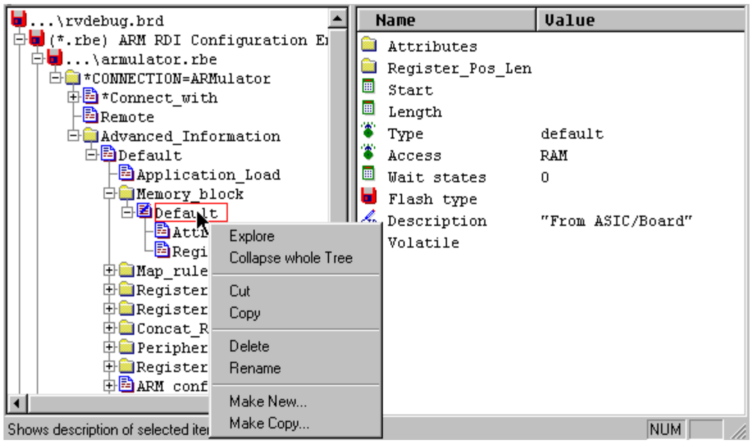
   c. `CONNECTION=ARMulator` *(change as required)*

   d. `Advanced_Information`

   e. `Default`

   f. `Memory_block`

3. Right-click on the `Default` entry, under `Memory_block`, to display the context menu, shown in Figure 3-23 on page 3-27.



```
...\rvdebug.brd
(*.rbe) ARM RDI Configuration E
...\armulator.rbe
*CONNECTION=ARMulator
*Connect_with
Remote
Advanced_Information
Default
Application_Load
Memory_block
Default
Attr
Regi
Map_rule
Register
Register
Concat_R
Peripher
Register
ARM conf
```

Name	Value
Attributes	
Register_Pos_Len	
Start	
Length	
Type	default
Access	RAM
Wait states	0
Flash type	
Description	"From ASIC/Board"
Volatile	

Figure 3-23 The default Memory_block settings for ARMulator

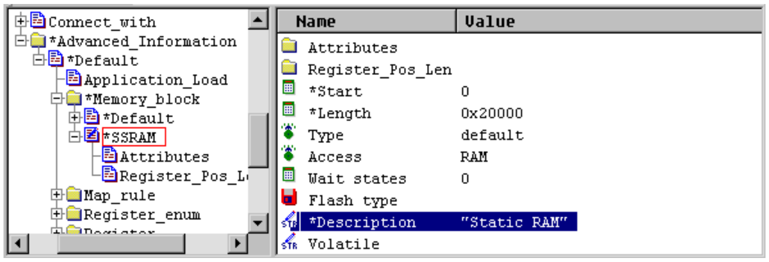
4. Select **Make Copy...** to describe the memory map for the chosen target. Give this entry a suitable name, for example `SSRAM`, and click **Create**.

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

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

7. Set the value of `Length` to `0x20000`.

8. Set the value of `Description` to `Static RAM` as shown in Figure 3-24.



```
Connect_with
*Advanced_Information
*Default
Application_Load
*Memory_block
*Default
*SSRAM
Attributes
Register_Pos_L
Map_rule
Register_enum
```

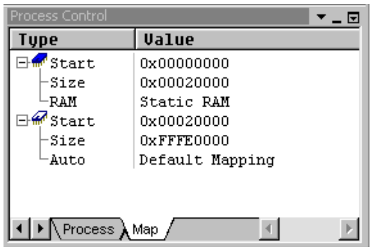
Name	Value
Attributes	
Register_Pos_Len	
*Start	0
*Length	0x20000
Type	default
Access	RAM
Wait states	0
Flash type	
*Description	"Static RAM"
Volatile	

Figure 3-24 Viewing the contents of the new group

9. Select **File** → **Save and Close** to save the new settings and close the window.

10. Connect to your target.

11. Select **View** → **Pane Views** → **Memory Map** to view the new memory map before loading an image, shown in Figure 3-25 on page 3-28.



Type	Value
Start	0x00000000
Size	0x00020000
RAM	Static RAM
Start	0x00020000
Size	0xFFFE0000
Auto	Default Mapping

Figure 3-25 New memory map in the Process Control pane

3.4.3 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.

In this example:

- The custom register, named `MYREG`, has an offset of `0x20` from the base of the I/O Register base.
- 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 memory region used for registers is called `REGS` and is addressed from `0x10000000-0x107FFFFF`.

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

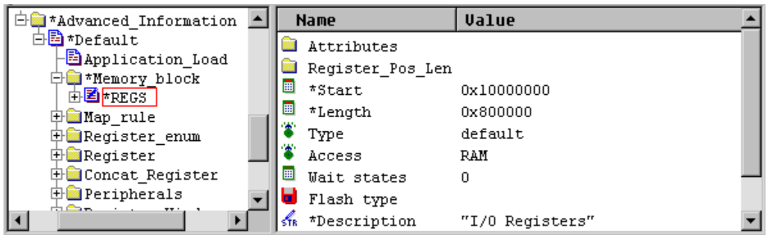
1. *Setting up the configuration*
2. *Creating enumerations for the register values* on page 3-29
3. *Creating the register descriptions* on page 3-30
4. *Creating the register tab* on page 3-31
5. *Displaying the register* on page 3-32.

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. Expand the following entries of the selected board group:
   a. `(*.rbe) ARM RDI Configuration Entries`
   b. `...\multiice.rbe` *(change as required)*
   c. `CONNECTION=Multi-ICE` *(change as required)*
   d. `Advanced_Information`
   e. `Default`
3. Expand the `Memory_block` group.
4. Rename the `Default` entry under `Memory_block` to `REGS`.
5. Click on the `REGS` entry, in the left pane, to display the group contents.
6. Set the value of `Start`, in the right pane, to `0x10000000`.
7. Set the value of `Length` to `0x800000` (Figure 3-26).



Name	Value
Attributes	
Register_Pos_Len	
*Start	0x10000000
*Length	0x800000
Type	default
Access	RAM
Wait states	0
Flash type	
*Description	"I/O Registers"

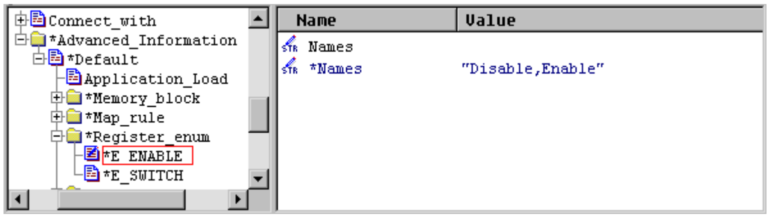
Figure 3-26 Configuring REGS

8. Set the value of `Description` to `I/O Registers`.

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. Use **Make New...** to create a new `Register_enum`. 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` (Figure 3-27).



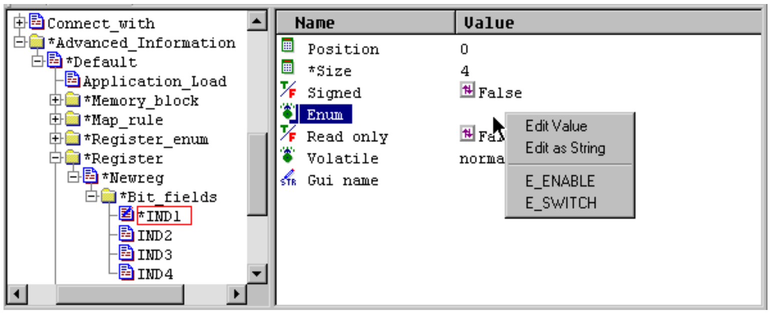
Name	Value
Names	
*Names	"Disable,Enable"

Figure 3-27 Creating enumerations

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`.
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 (Figure 3-28 on page 3-31):
    - `Position=0` (this is the default)
    - `Size=4`
    - `Enum=E_ENABLE`.



```
⊞ Connect_with
⊟ *Advanced_Information
  ⊟ *Default
      Application_Load
    ⊞ *Memory_block
    ⊞ *Map_rule
    ⊞ *Register_enum
    ⊟ *Register
      ⊟ *Newreg
        ⊟ *Bit_fields
            *IND1
            IND2
            IND3
            IND4
```

Name	Value
Position	0
*Size	4
Signed	False
Enum	
Read only	Fa
Volatile	norma
Gui name	

Figure 3-28 Creating bit field descriptions

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`.

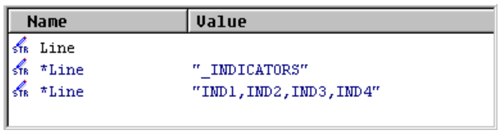
14. Click on the `IND4` entry and set these values:
    - `Position=12`
    - `Size=4`.

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 3-29 on page 3-32.



Name	Value
Line	
*Line	"_INDICATORS"
*Line	"IND1,IND2,IND3,IND4"

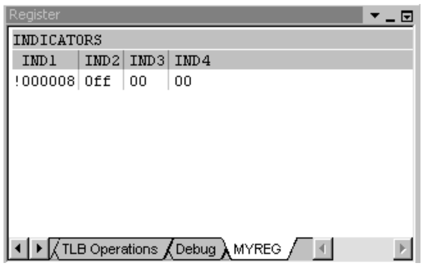
Figure 3-29 The MYREG group

All board file entries are now complete.

Displaying the register

In the last stage save the changes and display the new register in the Register pane:

1. Select **File** → **Save and Close** to save the new settings and close the Connection Properties window.
2. Connect to your target.
3. Select **View** → **Pane Views** → **Registers** to view the new tab, **MYREG**, shown in Figure 3-30.



```
INDICATORS
 IND1    IND2 IND3 IND4
!000008 0ff  00   00
```

Figure 3-30 MYREG in the Register pane

3.4.4 Setting up memory blocks

This example assumes that you have worked through *Setting up a custom register* on page 3-28, 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 3-28. This is displayed in the **MYREG** tab in the Register pane.

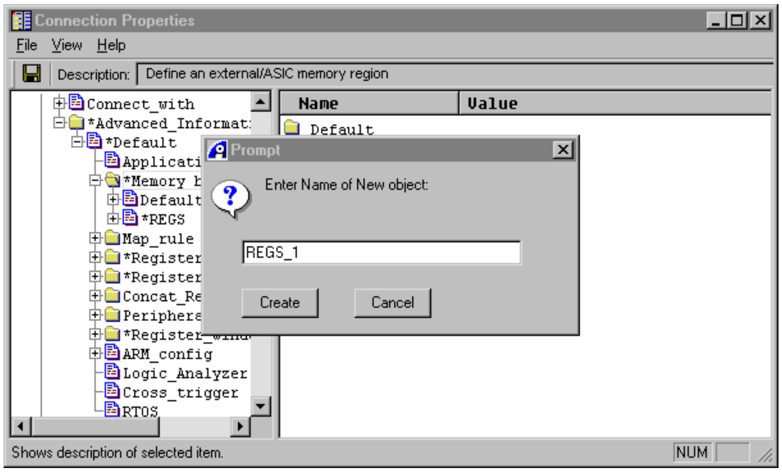
This example also describes setting a memory rule to specify how the memory is used. When `Newreg` is zero, `MEM2` is activated. Otherwise, `MEM1` is used. The example is split into these sections, which must be executed in this sequence:

1. *Defining the memory blocks*
2. *Defining the memory rules* on page 3-34.

Defining the memory blocks

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

1. Ensure that RealView Debugger is not connected to a target.
2. Expand the following entries of the selected board group:
   a. `(*.rbe) ARM RDI Configuration Entries`
   b. `...\multiice.rbe` *(change as required)*
   c. `CONNECTION=Multi-ICE` *(change as required)*
   d. `Advanced_Information`
   e. `Default`
   f. `Memory_Block`
3. Right-click on the `REGS` entry, in the left pane, and select **Make New...** from the context menu (Figure 3-31).



Description: Define an external/ASIC memory region

```
Connect_with
*Advanced_Informat:
  *Default
    Applicati
    *Memory_b
      Default
      *REGS
    Map_rule
    *Register
    *Register
    Concat_Re
    Periphera
    *Register_wind
    ARM_config
    Logic_Analyzer
    Cross_trigger
    RTOS
```

Default

Enter Name of New object:

REGS_1

Shows description of selected item.

Figure 3-31 Creating a new memory block

4. Enter a new name for this entry, for example `MEM1`, and click **Create**.
5. Click on the `MEM1` entry, in the left pane.

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

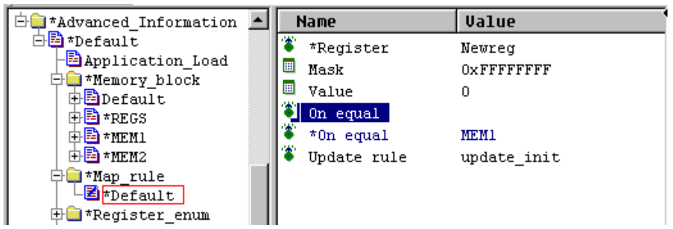
Defining the memory rules

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`. (Figure 3-32)

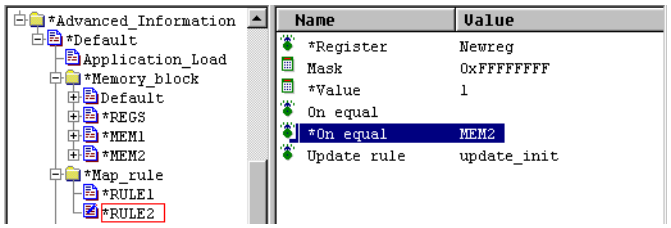


```
*Advanced_Information
  *Default
    Application_Load
    *Memory_block
      Default
      *REGS
      *MEM1
      *MEM2
    *Map_rule
      *Default
    *Register_enum
```

Name	Value
*Register	Newreg
Mask	0xFFFFFFFF
Value	0
On equal	
*On equal	MEM1
Update rule	update_init

Figure 3-32 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`. All board file entries are now complete (Figure 3-33 on page 3-35).



```
*Advanced_Information
  *Default
    Application_Load
    *Memory_block
      Default
      *REGS
      *MEM1
      *MEM2
    *Map_rule
      *RULE1
      *RULE2
```

Name	Value
*Register	Newreg
Mask	0xFFFFFFFF
*Value	1
On equal	
*On equal	MEM2
Update rule	update_init

Figure 3-33 Settings for the second map rule

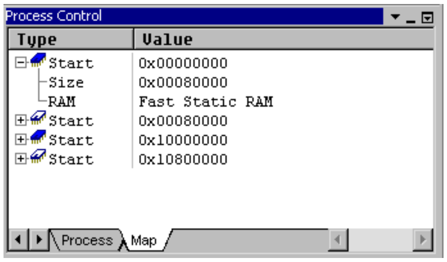
9. Select **File** → **Save and Close** to save the new settings and close the Connection Properties window.

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 3-34.



Type	Value
Start	0x00000000
Size	0x00080000
RAM	Fast Static RAM
Start	0x00080000
Start	0x10000000
Start	0x10800000

Figure 3-34 New memory block in the Map tab

3.4.5 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.

This example uses an Integrator/AP board with a core module, but the procedure for amending the settings is the same for any target.

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.

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 section in the *Multi-ICE User Guide*. You can set the value of `top_of_memory` in a `BOARD` description of the target or in the `CONNECTION` you use to connect to the target.

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. Expand the following entries of the selected board group:
   a. `(*.rbe) ARM RDI Configuration Entries`
   b. `...\multiice.rbe` *(change as required)*
   c. `CONNECTION=Multi-ICE` *(change as required)*
   d. `Advanced_Information`
   e. `Default`
   f. `ARM_config`
3. 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`.

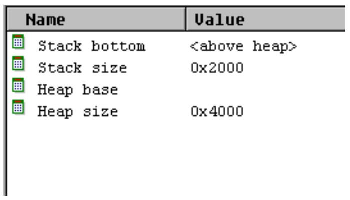
   Note

   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.

   ---

4. Double-click on the `Stack_Heap` group, in the right pane, to display the contents (Figure 3-35).



Name	Value
Stack bottom	<above heap>
Stack size	0x2000
Heap base	
Heap size	0x4000

Figure 3-35 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 (Figure 3-36).

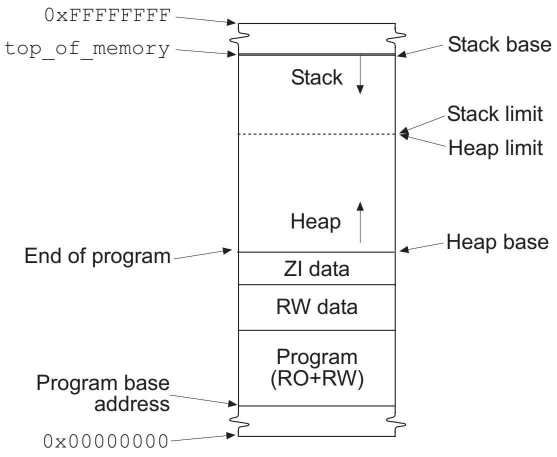


Figure 3-36 Relating top_of_memory to single section program layout

5. 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.

6. Select **File** → **Save and Close** to 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. For details on how to connect, see *Connecting to the new target* on page 3-24.

3.4.6 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*
3. *Connecting and running the image* on page 3-40
4. *Configuring RealMonitor* on page 3-41
5. *Connecting and running the RealMonitor image* on page 3-41.

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

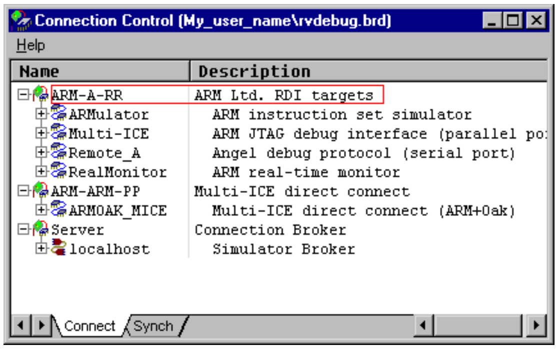
   If you are using the Multi-ICE server with RMHost, you must ensure that it is not autoconfigured because this causes the target to be reset and disrupts any running program. See the *ARM RMHost User Guide* for more details on configuring Multi-ICE server with RMHost.

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 3-37.

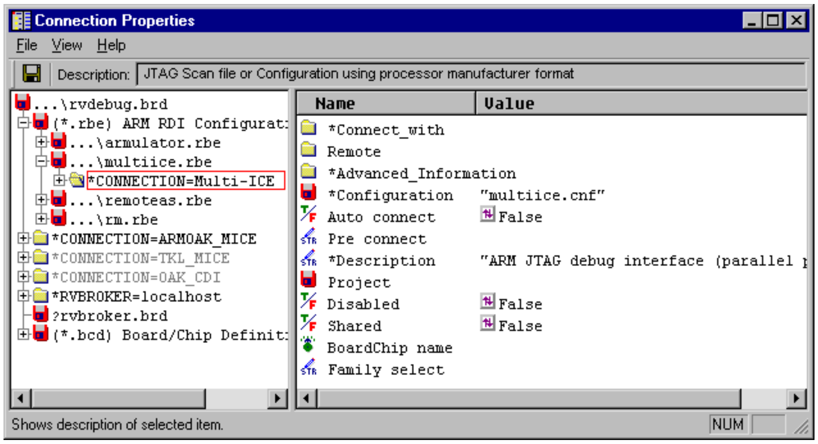


Name	Description
ARM-A-RR	ARM Ltd. RDI targets
ARMulator	ARM instruction set simulator
Multi-ICE	ARM JTAG debug interface (parallel po:
Remote_A	Angel debug protocol (serial port)
RealMonitor	ARM real-time monitor
ARM-ARM-PP	Multi-ICE direct connect
ARMOAK_MICE	Multi-ICE direct connect (ARM+Oak)
Server	Connection Broker
localhost	Simulator Broker

Figure 3-37 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 RDI targets* on page 4-6 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 (Figure 3-38).


Connection Properties
File View Help
Description: JTAG Scan file or Configuration using processor manufacturer format
Shows description of selected item.
NUM

```
...\rvdebug.brd
  (*.rbe) ARM RDI Configurat:
    ...\armulator.rbe
    ...\multiice.rbe
      *CONNECTION=Multi-ICE
    ...\remoteas.rbe
    ...\rm.rbe
  *CONNECTION=ARMOAK_MICE
  *CONNECTION=TKL_MICE
  *CONNECTION=OAK_CDI
  *RVBROKER=localhost
  ?rvbroker.brd
  (*.bcd) Board/Chip Definit:
```

Name	Value
*Connect_with	
Remote	
*Advanced_Information	
*Configuration	"multiice.cnf"
Auto connect	False
Pre connect	
*Description	"ARM JTAG debug interface (parallel ı
Project	
Disabled	False
Shared	False
BoardChip name	
Family select	

Figure 3-38 Multi-ICE connection properties

4. Expand the following entries of the selected board group:
   a. `(*.rbe) ARM RDI Configuration Entries`
   b. `...\multiice.rbe` (change as required)

   c. `CONNECTION=Multi-ICE` (change as required)

5. Set the `BoardChip_name` in the right pane, as described in *Setting up an Integrator board and core module* on page 3-20, for example:
   - `BoardChip_name=AP`
   - `BoardChip_name=CM7TDMI`.
6. Expand the following entries of the selected board group:
   a. `(*.rbe) ARM RDI Configuration Entries`
   b. `...\rm.rbe`
   c. `CONNECTION=RealMonitor`
7. Set the `BoardChip_name` in the right pane, for example:
   - `BoardChip_name=AP`
   - `BoardChip_name=CM7TDMI`.
8. Select **File** → **Save and Close** to save the new settings 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 where you can locate the LEDs demonstration for your target, for example `*install_directory*\AFSv1_4\Demos\Integrator7TDMI\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 the *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 2-19 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 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 4-14 for details.
4. Click **OK** to close the RealMonitor configuration dialog.

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 2-11 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 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.

   ————————————

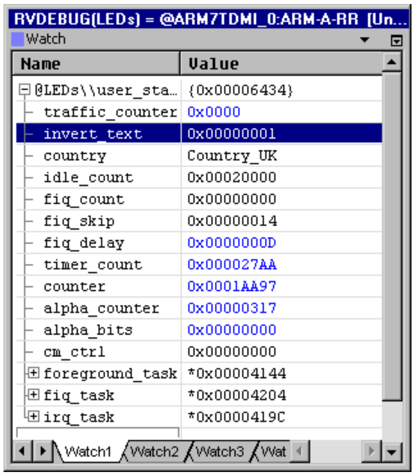
5. To get the context, you have to stop and restart the image:

   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 3-39.



RVDEBUG(LEDs) = @ARM7TDMI_0:ARM-A-RR [Un...

Watch

Name	Value
@LEDs\\user_sta...	{0x00006434}
traffic_counter	0x0000
invert_text	0x00000001
country	Country_UK
idle_count	0x00020000
fiq_count	0x00000000
fiq_skip	0x00000014
fiq_delay	0x0000000D
timer_count	0x000027AA
counter	0x0001AA97
alpha_counter	0x00000317
alpha_bits	0x00000000
cm_ctrl	0x00000000
foreground_task	*0x00004144
fiq_task	*0x00004204
irq_task	*0x0000419C

Figure 3-39 Watching the LEDs user_state variable

3.4.7 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 3-26
- 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 in a directory called `\flash`, as part of the base product. The board-specific files have names starting with `b_`, for example `b_aeb1.s`. The Flash memory files have names starting with `f_`, for example `f_atmel.s`.

RealView Debugger projects to create FME files from these sources are also provided, for example in `\flash\examples\...`

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 system 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* on page 3-44

2. *Programming the image into Flash.*

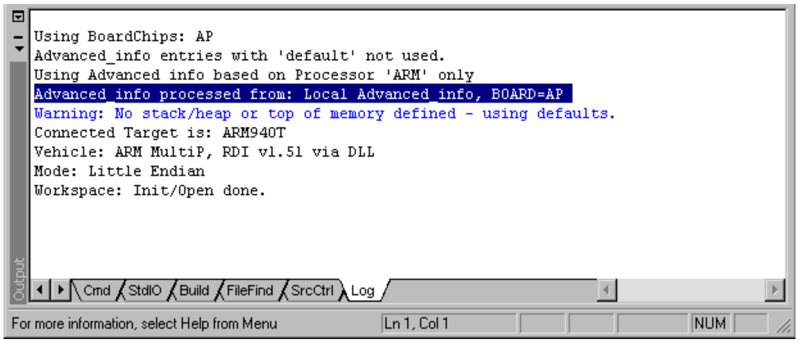
Defining the new target

To configure the Flash target:

1. Ensure that RealView Debugger is not connected to a target.
2. Click on the `CONNECTION=` entry, in the right pane, to display the settings values in the left pane.
3. Set the `BoardChip_name` of the `CONNECTION` to **AP**, so that the predefined Integrator/AP board file is used for this connection.
4. Select **File** → **Save and Close** to close the Connection Properties window.
5. Connect to the target using the Connection Control window.
6. Click on the **Log** tab in the Output pane, shown in Figure 3-40. This includes the line:

   ```
   Using BoardChips: AP
   ```

   This tells you that RealView Debugger is using the Integrator/AP Board/Chip Definition file (`AP.bcd`). As a result, the memory map now contains the definitions required to use the Flash memory on the Integrator board.



```
Using BoardChips: AP
Advanced_info entries with 'default' not used.
Using Advanced info based on Processor 'ARM' only
Advanced_info processed from: Local Advanced_info, BOARD=AP
Warning: No stack/heap or top of memory defined - using defaults.
Connected Target is: ARM940T
Vehicle: ARM MultiP, RDI v1.51 via DLL
Mode: Little Endian
Workspace: Init/Open done.
```

Figure 3-40 Output pane after configuration

Programming the image into Flash

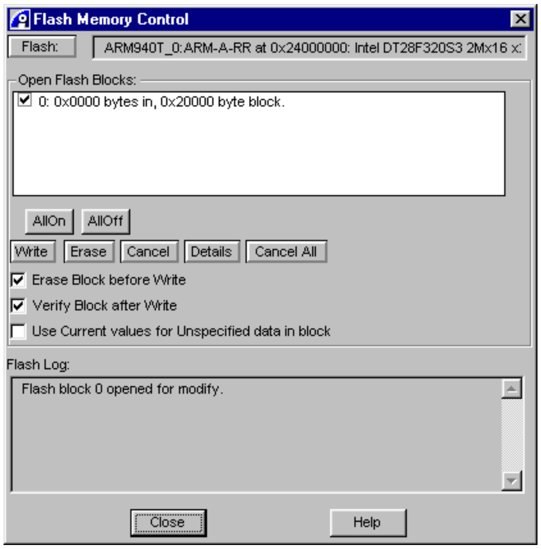
To program the image, you request RealView Debugger to write to the Flash memory region that you have defined by using the Integrator/AP board file. The Integrator Flash starts at memory address `0x24000000`, so to write an image to flash:

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. 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. The Flash Memory Control dialog appears, shown in Figure 3-41.



ARM940T_0:ARM-A-RR at 0x24000000: Intel DT28F320S3 2Mx16 x:

Open Flash Blocks:

☑ 0: 0x0000 bytes in, 0x20000 byte block.

☑ Erase Block before Write

☑ Verify Block after Write

☐ Use Current values for Unspecified data in block

Flash Log:

```
Flash block 0 opened for modify.
```

Figure 3-41 The Flash Memory Control dialog

4. Click **Write** to program the image into Flash.

5. Click **Close** to close the Flash Memory Control dialog.

3.4.8 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 `\home\`*`user_name`*.

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

3.4.9 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 1-9.
- 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 1-6 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 1-6 for more details.
  3. Using a second Windows Explorer window, locate the RealView Debugger installation directory.

     See *The install directory* on page 1-6 for more details.
  4. Use the hints given in *Using manual file or directory backups* on page 1-9 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 RDI targets such as Multi-ICE are configured. It contains the following sections:

- *Working with connection properties* on page 4-2
- *Working with RDI targets* on page 4-6
- *Working with JTAG files* on page 4-15.

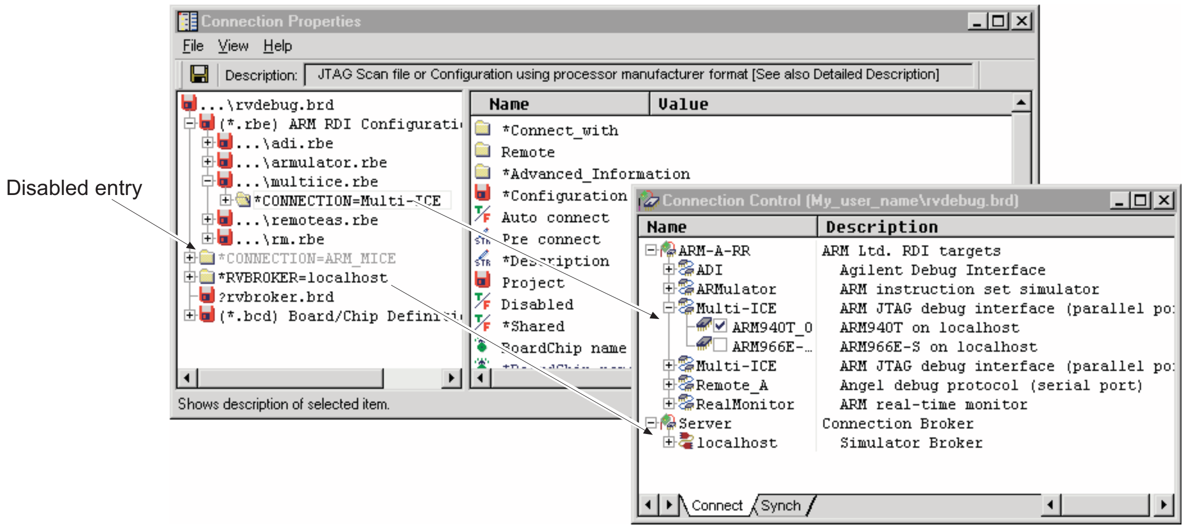
4.1 Working with connection properties

Connection properties, contained in board file entries, define possible connections to debug target systems. A debug target might be a simulator, an emulator, or an evaluation board installed on your host workstation.

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.

You can enable and disable each entry in the board file. 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 4-3 for details on how to do this.

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



```
Description: JTAG Scan file or Configuration using processor manufacturer format [See also Detailed Description]

...\rvdebug.brd
(*.rbe) ARM RDI Configurati
  ...\adi.rbe
  ...\armulator.rbe
  ...\multiice.rbe
    *CONNECTION=Multi-ICE
  ...\remoteas.rbe
  ...\rm.rbe
*CONNECTION=ARM_MICE
*RVBROKER=localhost
?rvbroker.brd
(*.bcd) Board/Chip Defini

Shows description of selected item.
```

Name	Value
*Connect_with	
Remote	
*Advanced_Information	
*Configuration	
Auto connect	
Pre connect	
*Description	
Project	
Disabled	
*Shared	
BoardChip name	

Name	Description
ARM-A-RR	ARM Ltd. RDI targets
ADI	Agilent Debug Interface
ARMulator	ARM instruction set simulator
Multi-ICE	ARM JTAG debug interface (parallel po:
☑ ARM940T_0	ARM940T on localhost
☐ ARM966E-...	ARM966E-S on localhost
Multi-ICE	ARM JTAG debug interface (parallel po:
Remote_A	Angel debug protocol (serial port)
RealMonitor	ARM real-time monitor
Server	Connection Broker
localhost	Simulator Broker

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

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 4-4 for details on how to do this.

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 emulator or board
- the host workstation I/O device address, 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.

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` or `CONNECTION`. You can create your own custom entries in this hierarchy using other types of entry, for example `BOARD`, `CHIP`, `COMPONENT`, or `DEVICE`.

Note

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*.

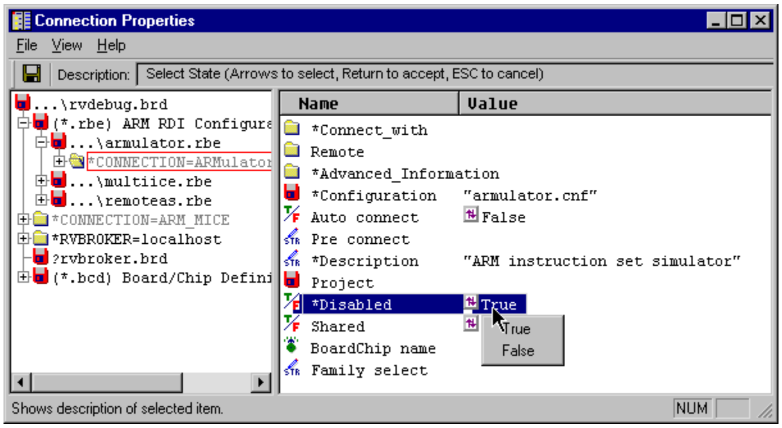
4.1.1 Enabling or disabling a board file entry

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, if you are using ARMulator, expand these entries:
   a. `(*.rbe) ARM RDI Configuration Entries`
   b. `...\armulator.rbe`

   It becomes the selected entry and its contents are displayed in the right pane, shown in Figure 4-2 on page 4-4.



Select State (Arrows to select, Return to accept, ESC to cancel)

```
...\rvdebug.brd
(*.rbe) ARM RDI Configure
  ...\armulator.rbe
    *CONNECTION=ARMulator
  ...\multiice.rbe
  ...\remoteas.rbe
*CONNECTION=ARM_MICE
*RVBROKER=localhost
?rvbroker.brd
(*.bcd) Board/Chip Defini
```

Name	Value
*Connect_with	
Remote	
*Advanced_Information	
*Configuration	"armulator.cnf"
Auto connect	False
Pre connect	
*Description	"ARM instruction set simulator"
Project	
*Disabled	True
Shared	
BoardChip name	
Family select	

True
False

Shows description of selected item.

Figure 4-2 Enabling or disabling a board file entry

4. Click on the `Disabled` entry, in the right pane, and select **True** from the menu.

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

6. Display the Connection Control window where this entry is no longer available.

You enable disabled board file entries in the same way.

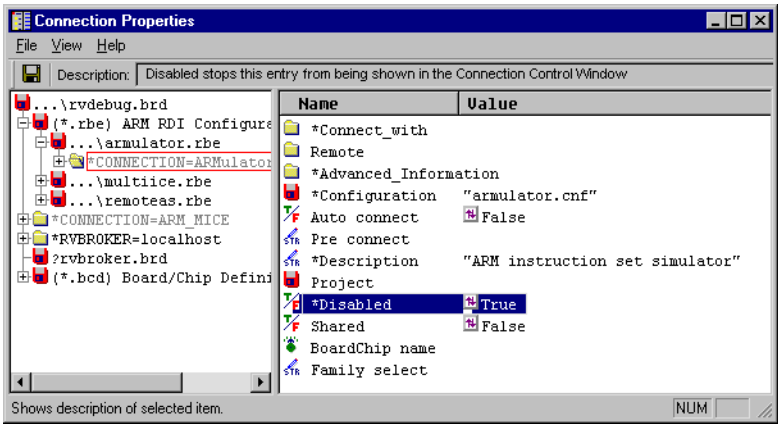
4.1.2 Restoring board file entry defaults

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 4-3, the `CONNECTION=ARMulator` entry was disabled, which means it contains a custom setting. To restore the default values for this entry:

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

2. Click on the `CONNECTION=ARMulator` 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 4-5.



Description: Disabled stops this entry from being shown in the Connection Control Window

```
...\rvdebug.brd
(*.rbe) ARM RDI Configure
  ...\armulator.rbe
    *CONNECTION=ARMulator
  ...\multiice.rbe
  ...\remoteas.rbe
*CONNECTION=ARM_MICE
*RVBROKER=localhost
?rvbroker.brd
(*.bcd) Board/Chip Defini
```

Name	Value
*Connect_with	
Remote	
*Advanced_Information	
*Configuration	"armulator.cnf"
Auto connect	False
Pre connect	
*Description	"ARM instruction set simulator"
Project	
*Disabled	True
Shared	False
BoardChip name	
Family select	

Shows description of selected item. NUM

Figure 4-3 Restoring board file entry defaults

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 the changes to your board file 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 `CONNECTION=ARMulator` 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. Enter a new description, for example `ARM RDI ARMulator` 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.

4.2 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.

The RealView Debugger base product includes RDI configuration files to simulate the `ARM7TDMI` core using ARMulator. You can change this by reconfiguring ARMulator, described in *Configuring ARMulator* on page 4-9. 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 4-11.

To add new RDI targets to the configuration settings, you must first add the required DLL to specify the device and then configure the chosen target.

See the following sections for details on these operations:

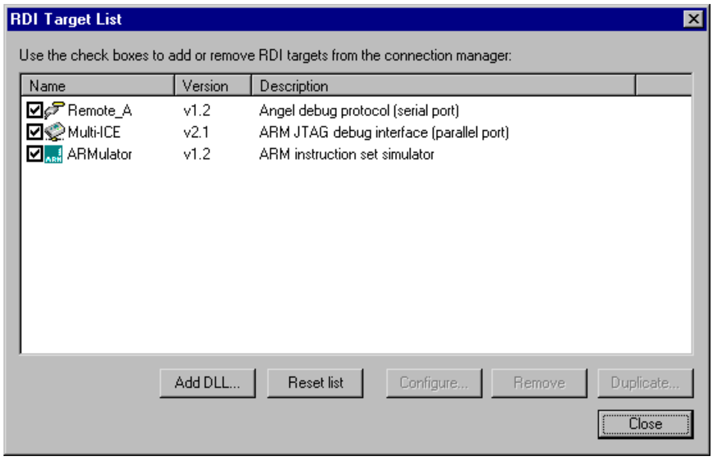
- *Adding RDI targets*
- *Configuring RDI targets* on page 4-8.

4.2.1 Adding RDI targets

RDI targets are automatically installed for ARM products such as ARM ADS 1.2 and ARM Multi-ICE 2.1. 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:

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 an RDI target, for example `ARMulator`, to display the **RDI Target** context menu.
4. Select the option **Add/Remove/Edit Devices...** to display the RDI Target List configuration dialog box shown in Figure 4-4 on page 4-7.


RDI Target List

Use the check boxes to add or remove RDI targets from the connection manager:

Name	Version	Description
☑ Remote_A	v1.2	Angel debug protocol (serial port)
☑ Multi-ICE	v2.1	ARM JTAG debug interface (parallel port)
☑ ARMulator	v1.2	ARM instruction set simulator

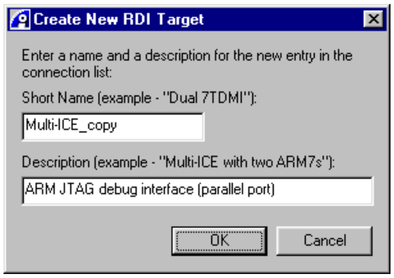
Figure 4-4 RDI Target List dialog box

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 4-5.


Create New RDI Target

Enter a name and a description for the new entry in the connection list:

Short Name (example - "Dual 7TDMI"):

Multi-ICE_copy

Description (example - "Multi-ICE with two ARM7s"):

ARM JTAG debug interface (parallel port)

Figure 4-5 Create New RDI Target dialog box

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 for the chosen entry. This is described in detail in *Configuring RDI targets*.

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:

- Autodetected targets cannot be removed from the display list.
- If you choose to reset list entries any targets added to the list since installation are also removed.

———

Click **Close** to close the RDI Target List dialog box and return to the Connection Control window.

4.2.2 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 4-4 on page 4-7. Highlight the entry in the display list and click **Configure...**.

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 ARMulator*
- *Configuring Remote_A* on page 4-11
- *Configuring ARM Multi-ICE* on page 4-11
- *Configuring ARM Agilent Debug Interface* on page 4-13.

These instructions do not replace the original manuals.

Configuring ARMulator

ARMulator is the ARM processor simulator and is supplied with ADS. For the `ARMulator` RDI target entry, the configuration dialog enables you to examine and change the following settings:

Processor Use the drop-down list to specify which ARM processor you want ARMulator 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 an emulation 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 ARMulator 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 4-1 ARMulator Endian settings

Usage	Debug Endian	Start target Endian
A target that is always little-endian. This is the default.	Little	Debug Endian
A target that is always big-endian	Big	Debug Endian
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	Big	Hardware Endian

Memory Map File

Specify a memory map file for use with ARMulator.

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 3-26 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 the *RVISS User Guide* for more information on how these settings apply to ARMulator.

Configuring Remote_A

To enable RealView Debugger to communicate with an Angel debug target, you use Remote_A, supplied with ADS. The Remote_A configuration dialog enables you to 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.

Configuring ARM Multi-ICE

You can use the Multi-ICE interface unit in two ways:

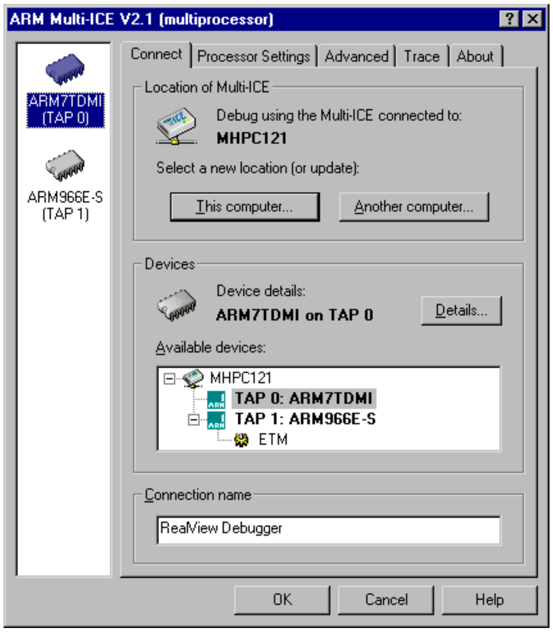
- If you are only connecting to ARM processors, use the RDI Multi-ICE DLL and the Multi-ICE server.
- If you want to connect to a DSP, use the Multi-ICE interface unit with no Multi-ICE server and the `ARM-ARM-PP Multi-ICE direct connect` vehicle.

  See *Working with JTAG files* on page 4-15 for general information about configuring Multi-ICE direct connect.

  See the *RealView Debugger v1.6 Extensions User Guide* for more information about connecting to DSP processors

The Multi-ICE DLL configuration dialog, shown in Figure 4-6, contains these 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.


ARM Multi-ICE V2.1 (multiprocessor)

Connect | Processor Settings | Advanced | Trace | About

ARM7TDMI (TAP 0)

ARM966E-S (TAP 1)

Location of Multi-ICE

Debug using the Multi-ICE connected to:
MHPC121

Select a new location (or update):

This computer... | Another computer...

Devices

Device details:
ARM7TDMI on TAP 0

Details...

Available devices:

```
MHPC121
  TAP 0: ARM7TDMI
  TAP 1: ARM966E-S
    ETM
```

Connection name

RealView Debugger

OK | Cancel | Help

Figure 4-6 Multi-ICE DLL multiprocessor configuration mode

If you connect to a Multi-ICE server that is configured with more than one ARM processor, the configuration dialog includes a side-panel showing icons for each of these processors, shown in Figure 4-6. This is not shown if you connect to a single processor.

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.

Processor Settings

This tab contains any processor-specific settings. See the *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 4-2 Multi-ICE Endian settings

Usage	Selection
A target that is always little-endian. This is the default.	Little-endian
A target that is always big-endian.	Big-endian

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.

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.

Configuring ARM Agilent Debug Interface

The ARM *Agilent Debug Interface* (ADI) configuration dialog 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, 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 the *ARM Agilent Debug Interface User Guide* for more information.

Configuring ARM RealMonitor

The ARM RealMonitor configuration dialog 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 the *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 4-6 on page 4-12.

See the *Multi-ICE User Guide* or the documentation that accompanies your JTAG unit for more details.

See the *ARM RMHost User Guide* for more information.

4.3 Working with JTAG files

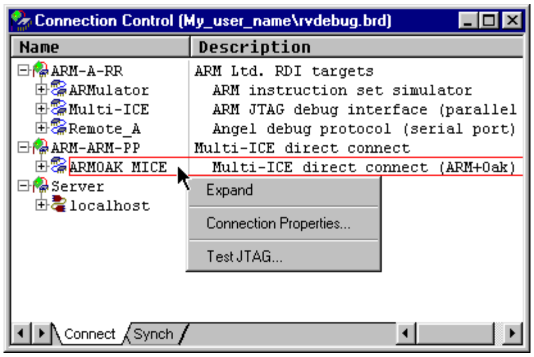
Multi-ICE direct connect is an OCD-based emulator that uses JTAG files to define the devices on the JTAG scan chain and their order. This information might be supplied by the manufacturer or might be configured after installation. RealView Debugger uses JTAG files to access emulator targets on the local 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.

RealView Debugger detects the JTAG files and uses them to complete the configuration settings. However, these files can be disabled. This means that they are included in the configuration settings but are not displayed as available targets in the Connection Control window.

To access the `.jtg` file editor for a Multi-ICE direct connect connection:

1. Start RealView Debugger without connecting to a target.
2. Display the Connection Control window.
3. Right-click on the chosen entry in the Connection Control window to display the context menu, shown in Figure 4-7.



Name	Description
ARM-A-RR	ARM Ltd. RDI targets
ARMulator	ARM instruction set simulator
Multi-ICE	ARM JTAG debug interface (parallel
Remote_A	Angel debug protocol (serial port)
ARM-ARM-PP	Multi-ICE direct connect
ARMOAK MICE	Multi-ICE direct connect (ARM+Oak)
Server	
localhost	

Figure 4-7 Editing JTAG files in the Connection Control window

4. Select **Connection Properties...** to display the Connection Properties window. Right-click on the required `.jtg` file and select **Edit Configuration-File Contents...** from the context menu, shown in Figure 4-8 on page 4-16.

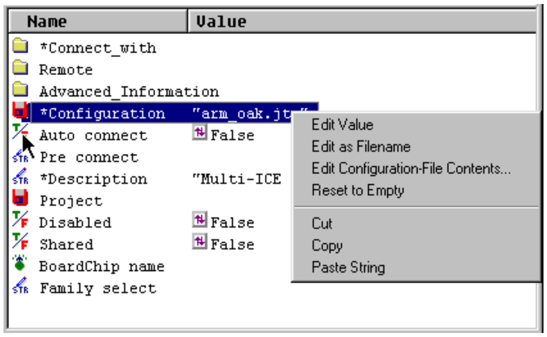


Figure 4-8 Editing JTAG files in the Connection Properties window

This displays the Device JTAG-File Editor dialog box, shown in Figure 4-9.

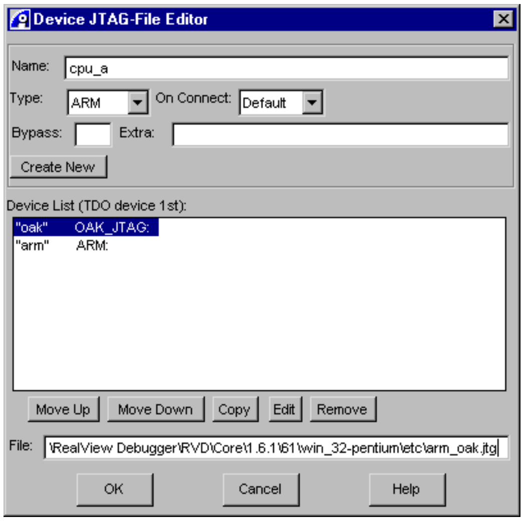


Figure 4-9 Device JTAG-File Editor dialog box

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`, or `TEAKLITE`.

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*). For example, selecting the vehicle `DSG-DSPG-AT` causes the drop-down list to include `OAK` and `TEALKITE`, but not `ARM`.

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 with the top list entry corresponding to the device that has its *Test Data Out* (TDO) connected to the host interface *Test Data In* (TDI) pin and the bottom list entry has its TDO connecting its TDI to TDO of the host interface. This is shown in Figure 4-10.

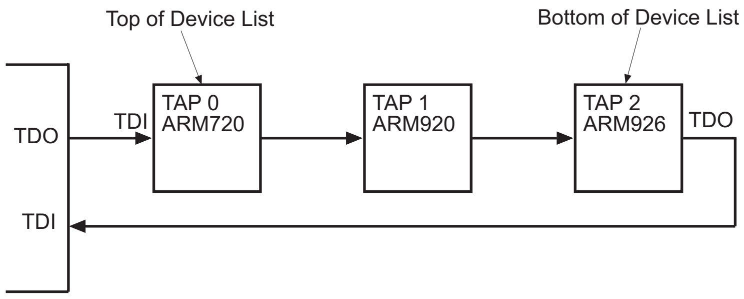


Figure 4-10 JTAG chain ordering

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.

4.3.1 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 `ARMOAK_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.

4.3.2 Defining a DSP target

This example defines a new Oak DSP target. 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`.

To define the new target:

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

2. Right-click on the `...\rvdebug.brd` entry, in the left pane.

3. Select **Make New Group...** from the menu, shown in Figure 4-11.

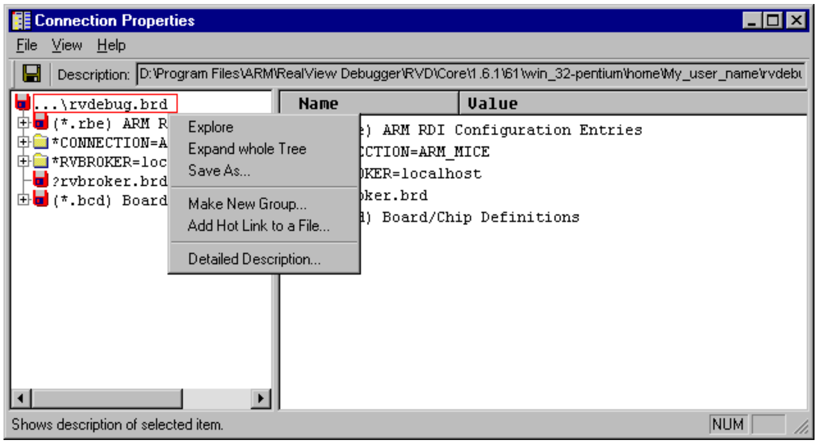


Figure 4-11 Defining a target board

4. This displays the Group Type/Name selector dialog, shown in Figure 4-12.

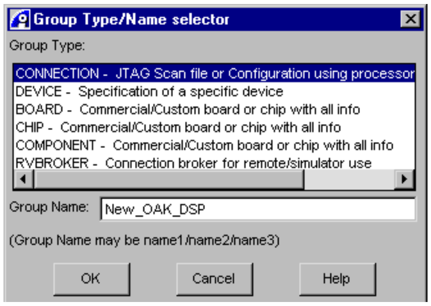

Group Type/Name selector
Group Type:
CONNECTION - JTAG Scan file or Configuration using processor
DEVICE - Specification of a specific device
BOARD - Commercial/Custom board or chip with all info
CHIP - Commercial/Custom board or chip with all info
COMPONENT - Commercial/Custom board or chip with all info
RVBROKER - Connection broker for remote/simulator use
Group Name: New_OAK_DSP
(Group Name may be name1/name2/name3)
OK
Cancel
Help

Figure 4-12 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**.

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

    Your new target board 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.

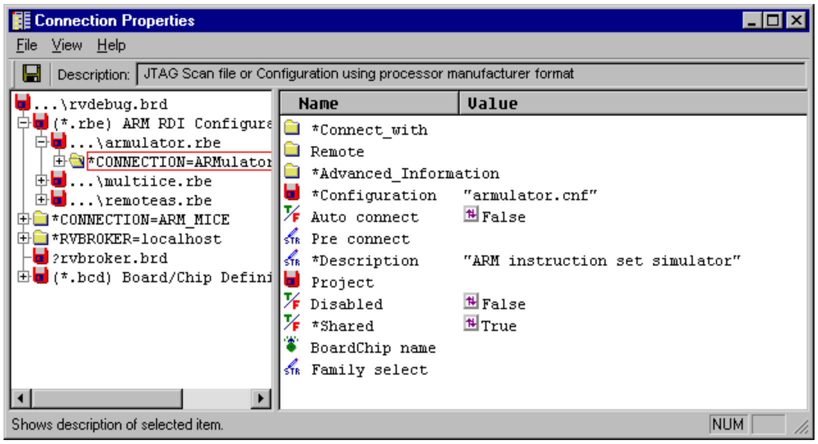
Appendix A
Configuration Settings Reference

This appendix contains reference details about board file entries that define target configurations and custom connections. It contains the sections:

- *Generic settings* on page A-2
- *Target configuration reference* on page A-5
- *Custom connection reference* on page A-21.

A.1 Generic settings

There are many board file entries that are common to several types of settings, shown in Figure A-1.



Description: JTAG Scan file or Configuration using processor manufacturer format

```
...\rvdebug.brd
  (*.rbe) ARM RDI Configure
    ...\armulator.rbe
      *CONNECTION=ARMulator
    ...\multiice.rbe
    ...\remoteas.rbe
  *CONNECTION=ARM_MICE
  *RVBROKER=localhost
  ?rvbroker.brd
  (*.bcd) Board/Chip Defini
```

Name	Value
*Connect_with	
Remote	
*Advanced_Information	
*Configuration	"armulator.cnf"
Auto connect	False
Pre connect	
*Description	"ARM instruction set simulator"
Project	
Disabled	False
*Shared	True
BoardChip name	
Family select	

Shows description of selected item.

Figure A-1 Viewing generic settings

They are described in this section, and referenced from each setting type:

Connect_with This includes the settings values:

Manufacturer The name and the type of the connection.

The values show the available connection types, but you also require an appropriate license and the hardware to use them.

IOdevice This field contains additional information about the hardware. It is not used in this release of the RealView Debugger.

Speed You can set the emulation speed for some emulators. It is not used in this release of the RealView Debugger.

Remote Connections to remote targets are not supported in this release.

Advanced_Information

Provides extended target visibility details about the debug target system. This feature is described in *The Advanced_Information block* on page A-5.

Configuration	Specifies a named JTAG scan 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 RDI targets, or the ARMulator simulator, `.jtg` files are replaced by the `.cnf` configuration files.
Auto_connect	Expands the list of devices on the target board. 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.
Disabled	Stops this entry from being shown in the Connection Control window.
Shared	Enables the sharing of target configurations for remote connections. This setting is not supported in this release.
BoardChip_name	Refers to the `BOARD`, `CHIP`, or `COMPONENT` group this 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`* or as a value.

A.2 Target configuration reference

This section describes in detail the target configuration entries that are supported by RealView Debugger. It is split into the following sections:

- *The Advanced_Information block*
- *BOARD, CHIP, and COMPONENT settings* on page A-19.

For information about connection configuration, including the `CONNECTION` and `DEVICE` entries, see *Custom connection reference* on page A-21.

A.2.1 The Advanced_Information block

The `Advanced_Information` block enables you to provide ETV information, for example about 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 information block. The base block is called `Default` and is used if you provide no other information. Entry names can be the same as processor or device names in `.jtg` or `.cnf` files. 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-6
- *Memory_block* on page A-6
- *Map_rule* on page A-10
- *Register_enum* on page A-10
- *Register* on page A-11
- *Concat_Register* on page A-12
- *Peripherals* on page A-13
- *Register_Window* on page A-14
- *ARM_config* on page A-14
- *Logic_Analyzer* on page A-16
- *Cross_trigger* on page A-16
- *RTOS* on page A-17
- *Pre_connect* on page A-17
- *Commands* on page A-17
- *Connect_mode* on page A-18
- *Disconnect_mode* on page A-18
- *Id_chip* on page A-19
- *Id_match* on page A-19

- *Chip_name* on page A-19
- *Endianess* on page A-19.

Application_Load

Use the `Application_Load` block 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. This block 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` block 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 program counter is initialized during an image load. The default is to set the program counter 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 program counter under any situation, or to set it specifically to address 0.

Memory_block

`Memory_block` entries 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-10). 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 the simulators and guide the debugger in access to 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 in simulators to control timing issues. It is noted if a link command file is generated.

Access_size

This field enables control of how the memory is accessed by the debugger internally. 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 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-10). 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 multi-processor 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 real targets, this only affects how the debugger uses the memory and any generated linker command files.

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 `\flash\examples` directory installed as part of the root installation. 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 ranges of 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-11).

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 needs to 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,... no matter where they are positioned 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 entry 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` entry contains a group called `Bit_fields` which in turn contains the `Default` settings:

Position	Bit position from 0 (LSbit).
Size	Size in bits.
Signed	True if signed, false if unsigned.
Enum	Enumeration name to show values to show values in the Register pane, derived from `Register_enum` group.
Read_only	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.
Gui_name	Optional name for showing in Register pane.

Other settings in the `Register` entry 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 this value is `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 this value is `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 this value is `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 entry 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 needed.
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-11 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-10, *Register* on page A-11, and *Concat_Register* on page A-12 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 entry enables control of ARM processor used for ARM emulators, monitors, or simulators, and emulators. These control features such as semihosting, vector catching, and memory top 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 block enable individual control over each vector.

Semihost_vector

Contains:

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 Angel, Multi-ICE, ARMulator). 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 target 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`.

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.

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 block is used to define settings for external trace analyzer hardware.

RealView Debugger currently supports:

- ARM-based trace targets
- the Oak and TeakLite DSPs
- Motorola MC56600
- 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 3-8) you must configure these settings to enable tracing:

- right-click on `Vendor` and select **ARM**
- right-click on `Load_when` and select **connect**.

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 commands to enable input triggering.
Trig_in_dis	List commands to disable input triggering.
Trig_out_ena	List commands to enable output triggering.
Trig_out_dis	List commands to disable output triggering.

RTOS

This entry enables automatic loading of RTOS and kernel awareness. This entry enables forced loading of RTOS support or symbol hooking. When the RTOS is symbol-hooked, support 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	Select the manufacturer of the RTOS from a popup list.
Load_when	Accepts a keyword from a list to specify when to load RTOS, for example on connection (**connect**), or when image is loaded (**image_load**).
Base_address	Defines a base address for RTOS data structures. See the documentation for your specific RTOS support code.

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

This setting defines the connection mode, described in *Setting connect mode* on page 2-11, when you connect by checking the check box in the Connection Control window, or when the debugger automatically connects on start-up. If you connect using the `CONNECT` command, or using the **File** menu, the connection mode is set in other ways. The options include:

default	Connect in the standard way for this connection type.
prompt	Prompt for the connection mode to use.
stop	Stop the target, placing it in debug state.
reset_stop	Reset the target and place it in debug state
non_stop	Leave the target running, making use of non-stop debug facilities of the connection, if available, to display memory.

The options available for `Connect_mode` settings are generic to all vehicles and supported processors.

Disconnect_mode

This setting defines the disconnection mode, described in *Setting disconnect mode* on page 2-19, when you disconnect by checking the check box in the Connection Control window, or when the debugger automatically disconnects on exit or when making a new connection in single-connection mode. If you disconnect using the `DISCONNECT` command, or using the **File** menu, the disconnection mode is set in other ways. The options include:

default	Disconnect in the standard way for this connection type.
prompt	Prompt for the disconnection mode to use.
as-is	Leave the target in the state it is in. That is, if it is stopped, leave it stopped, if running then leave it running. Breakpoints that are currently set are not deleted.
stopped	Leave the target stopped, in debug state
running	Leave the target running, with breakpoints deleted.

The options available for `Disconnect_mode` settings are generic to all vehicles and supported processors.

Id_chip

The chip-id, or silicon-id, is loaded from the processor normally. 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 ARMulator only. Use it to set the byte order of the simulated processor.

A.2.2 BOARD, CHIP, and COMPONENT settings

You use a `BOARD`, `CHIP`, or `COMPONENT` entry when a standard board or chip, core plus ASICs, exists, either commercial or custom. A `CONNECTION` group can refer to this entry by name or ID. A reference to one of these groups enables automatic use of the `.jtg` file, settings, and advanced information (ASIC, peripherals, and memory). This entry can specify the default connection information which can be overridden by the `CONNECTION` group (see *CONNECTION settings* on page A-23 for details).

The following entries are available, as described in *Generic settings* on page A-2:

- 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.3 Custom connection reference

There are different types of board file entry depending on the kind of targets they describe and the format of configuration files that they access. The board file consists of the following types of entry:

- *Connections and targets*
- *Board file (brd file)* on page A-22
- *CONNECTION settings* on page A-23
- *DEVICE settings* on page A-23
- *ARM RDI Configuration (rbe file)* on page A-24
- *JTAG configuration (jtg file)* on page A-24.

A.3.1 Connections and targets

The board file describes two things about the target hardware:

- the hardware components used by software on the target
- the method used to access the target.

The target access method description 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.

The RealView Debugger board file group type `CONNECTION` is normally used to specify target access method connection details (although occasionally, you might use `DEVICE` instead). Within a specific `CONNECTION`, one or more `BoardChip_name` entries are used to associate the connection with a `BOARD` or `CHIP` 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`.

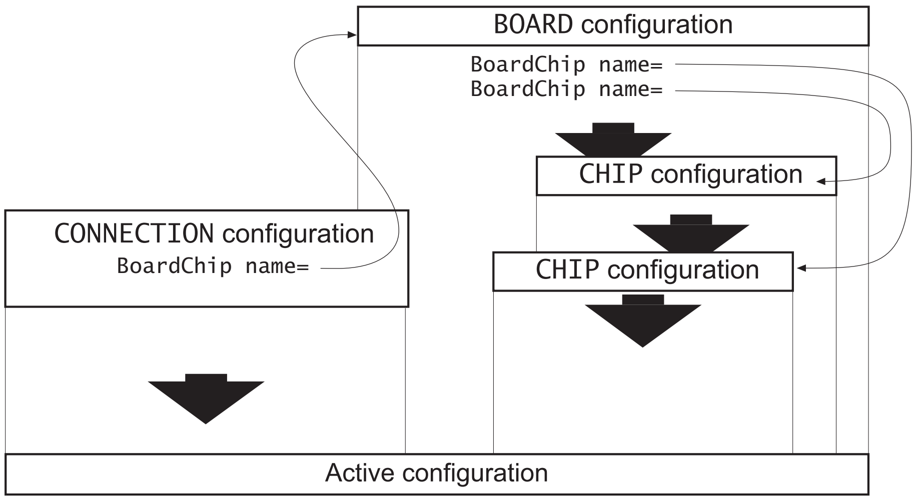


Figure A-2 How Components, Boards, and Chips fit together

A.3.2 Board file (brd file)

RealView Debugger uses a board file to access information about the debugging environment and the debug targets available to you. You can use RealView Debugger with the default board file that is installed for you in your debugger home directory. See *The home directory* on page 1-6 for more information.

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 **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 v1.6 User Guide* for details.

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.

A.3.3 CONNECTION settings

`CONNECTION` entries are used by boards to get a list of one or more 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 group of type `CONNECTION` 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 settings* on page A-2:

- Connect_with
- Remote
- Advanced_information
- Configuration
- Auto_connect
- Pre_connect
- Description
- Project
- Disabled
- Shared
- BoardChip_name
- Family_select.

A.3.4 DEVICE settings

You 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 settings* on page A-2:

- 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.3.5 ARM RDI Configuration (rbe file)

These are RDI configuration entries generated by the RDI configuration utilities.

You can expand this entry to see the second-level entries that list autodetected targets. Expand one of these, for example `armulator.rbe`, to see the `CONNECTION` entry, `CONNECTION=ARMulator`, where the configuration file, `armulator.cnf`, is specified.

A.3.6 JTAG configuration (jtg file)

JTAG configuration files 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`. 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_oak.jtg`	Specifies a DSP Group Oak processor and then an ARM processor on the scan chain.
`oak.jtg`	Specifies a DSP Group Oak processor on the scan chain.
`arm_mp.jtg`	Specifies two ARM processors on the scan chain
`teaklite.jtg`	Specifies a DSP Group TeakLite processor on the scan chain.

You must have the RealView Debugger DSP license to access the DSP Group processors.

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.

ARM state	A processor that is executing ARM (32-bit) instructions is operating in ARM state.
	See also Thumb state
ARMulator	ARMulator is an instruction set simulator. It is a collection of modules that simulate the instruction sets and architecture of various ARM processors.
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.
ATPCS	ARM-Thumb Procedure Call Standard.
Backtracing	*See* Stack Traceback.
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.
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
CPSR	Current Program Status Register.
	See also 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.

Doubleword
A 64-bit unit of information.

DWARF
See Debug With Arbitrary Record Format.

ELF
Executable and Linking Format. ARM code generation tools produce objects and executable images in ELF format.

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*.

ETV
See Extended Target Visibility.

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

Extended Target Visibility (ETV)
Extended Target Visibility enables RealView 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 Compilation Tools

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 Debugger Trace

A software product add-on to RealView Debugger that extends the debugging capability with the addition of real-time program and data tracing.

Remote Debug Interface (RDI)

The *Remote Debug Interface* 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, ARMulator)
- a debug monitor running on ARM-based hardware accessed through a communication link (for example, Angel)
- a debug agent controlling an ARM processor through hardware debug support (for example, Multi-ICE).

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.

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.
SPSR	Saved Program Status Register. *See also* Program Status Register.
Stack traceback	This 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.
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 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, target knowledge, such as memory mapping, lists, rule processing, board-files 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.

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.

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.