MULTI: Configuring Connections for ARM Targets
DISCLAIMER

GREEN HILLS SOFTWARE, INC., MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. Further, Green Hills Software, Inc., reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Green Hills Software, Inc., to notify any person of such revision or changes.

Copyright © 1983-2005 by Green Hills Software, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission from Green Hills Software, Inc.


All other company, product, or service names mentioned in this book may be trademarks or service marks of their respective owners.

PubID: 10345
October 24, 2005
Contents

1 Introduction

The MULTI Integrated Development Environment 2
The MULTI Launcher 4
    MULTI Workspaces 5
The MULTI 4.0 Document Set 6
About This Book: Configuring Connections 7
Supported Target Connections for ARM 8
Online Help
    Viewing Help on Windows Systems 10
    Viewing Help on UNIX Systems 11
Conventions Used in This Book 12

2 Setting Up Your Target Hardware

Installing Your Target Hardware 16
Configuring Your Target Hardware 16
    Using the New Project Wizard to Configure and
    Test Your Target 16
    Creating Customized Setup Script and Linker
    Directives Files 20
Specifying Setup Scripts 28
    Using MULTI (.mbs) Setup Scripts When
    Connecting to Your Target 29
    Using Legacy Debug Server (.dbs) Setup Scripts
    When Connecting to Your Target 30
    Running Setup Scripts Manually 31

3 Green Hills Debug Probe (mpserv)
   Connections 33

4 Agilent Processor Probe (hpserv)
   Connections 35
   Supported Targets and Features for Agilent Processor
   Probe (hpserv) Connections 36
   Agilent Processor Probe Target System Requirements 36
Contents

Checking Your Agilent Firmware 37
Installing Your Agilent Processor Probe 37
Troubleshooting Your Agilent Processor Probe Setup 38
Connecting to Your Target with an Agilent Processor Probe 39
  Creating a Standard Agilent Processor Probe (hpserv) Connection Method 39
  Using the Agilent Processor Probe (hpserv) Connection Editor 40
  Using Custom Agilent Processor Probe (hpserv) Connection Methods 45
Using the Agilent Processor Probe with a Multiple-Device JTAG Scan Chain 49
Trace Support 50
  Additional Setup Required to Collect ETM Trace with a Logic Analyzer 50
  Dual ETM Trace Support 51
Additional Commands for Agilent Processor Probe (hpserv) Connections 52

5 ARM Remote Debug Interface (rdiserv) Connections 55
Supported Remote Debug Interfaces (RDIs) 56
  Angel Debug Protocol (ADP) Targets: Angel Monitor and EPI JEENI Probe 56
  ARM Multi-ICE Probe 56
  ARMulator Simulator 57
Supported Processors for Remote Debug Interface (RDI) Connections 57
ARM Remote Debug Interface (RDI) Hardware Breakpoint Support 58
Installing Your Remote Debug Interface (RDI) 59
  Angel Monitor or JEENI Probe Installation and Setup 59
  ARM Multi-ICE Probe Installation and Setup 60
  ARMulator Installation and Setup 60
Connecting to Your Target with a Remote Debug Interface (RDI) 60
Contents

Creating a Standard Connection Method for Your Remote Debug Interface (RDI) Connection 61

Using the Angel/JEENI (rdiserv) Connection Editor 62
   Angel/JEENI (rdiserv) Connection Settings 63
   Angel/JEENI (rdiserv) Download Settings 64
   Angel/JEENI (rdiserv) Advanced Settings 65
   Angel/JEENI (rdiserv) Debug Settings 66

Using the ARM Multi-ICE (rdiserv) Connection Editor 67
   Multi-ICE (rdiserv) Connection Settings 69
   Multi-ICE (rdiserv) Download Settings 69
   Multi-ICE (rdiserv) Advanced Settings 70
   Multi-ICE (rdiserv) Debug Settings 71

Using the ARMulator (rdiserv) Connection Editor 72
   ARMulator (rdiserv) Connection Settings 74
   ARMulator (rdiserv) Download Settings 74
   ARMulator (rdiserv) Advanced Settings 75
   ARMulator (rdiserv) Debug Settings 76

Using Custom Remote Debug Interface (rdiserv) Connection Methods 77
   Custom Angel Monitor or EPI JEENI Probe Connections with rdiserv 77
   Custom ARM Multi-ICE Probe Connections with rdiserv 78
   Custom ARMulator Connections with rdiserv 79
   Options for Custom Remote Debug Interface (rdiserv) Connections 80
   Example Custom Remote Debug Interface (RDI) Connections 84

Additional Commands for Remote Debug Interface (rdiserv) Connections 85

Troubleshooting Your Remote Debug Interface (rdiserv) Connection 87
   rdiserv Cannot Connect to the Target 87
   Unable to set syscall Breakpoint 88

RDI Error Codes 88

Using the Green Hills Tracer Module for the ARMulator 89
   Profiling with the Green Hills Tracer Module 89
   Configuring the Tracer Module 89
Contents

8 Simulator for ARM (simarm) Connections 127
   Installing the Simulator for ARM (simarm) 128
   Connecting to the Simulator for ARM (simarm) 128
      Creating a Standard Simulator for ARM (simarm) Connection Method 128
      Using the Simulator for ARM (simarm) Connection Editor 129
      Using Custom Simulator for ARM (simarm) Connection Methods 132
   Additional Commands for Simulator for ARM (simarm) Connections 133
   Simulation Modes 133
      OS Simulation Mode 134
      ROM Mode 134
   Profiling with simarm 135
   Using System Calls in Your Code 135
   Connecting to the Simulator for ARM (simarm) as a Stand-alone Debug Server 136

A Green Hills Debug Server Command and Scripting Reference 137
   Green Hills Debug Server Commands 138
      Generic Debug Server Commands 138
      Additional Debug Server Commands 144
   The Green Hills Debug Server Scripting Language 145
      General Notes 146
      Scripting Syntax 146
      Example Scripts 150

B Using the gproxy Network Proxy Utility 155
   Network Proxy File Format 157
   Connecting to your Target 158

Index 159
Chapter 1

Introduction

*This Chapter Contains:*
- The MULTI Integrated Development Environment
- The MULTI Launcher
- The MULTI 4.0 Document Set
- About This Book: Configuring Connections
- Supported Target Connections for ARM
- Online Help
- Conventions Used in This Book
1. Introduction

This chapter provides an overview of the MULTI Integrated Development Environment and its documentation set, and then gives a brief introduction to this book, which describes how to set up your specific debugging interface for use with MULTI and how to configure connections to your ARM target.

The MULTI Integrated Development Environment

MULTI is a complete Integrated Development Environment (IDE) designed especially for embedded systems engineers to assist them in analyzing, editing, compiling, optimizing, and debugging embedded applications.

The MULTI IDE includes graphical tools for each part of the software development process. The following tools can be launched from within the IDE or as separate stand-alone programs:

• IDE launcher
  - **MULTI Launcher (mstart)** — The gateway to the MULTI IDE, which allows you to quickly launch any of the primary MULTI tools, access open windows, and manage MULTI workspaces

• Editing tools
  - **MULTI Editor (me)** — A graphical editor for modifying text files
  - **Checkout Browser (mcobrowse)** — A graphical viewer for files managed under a version control system
  - **Diff Viewer (diffview)** — A graphical viewer that displays differences between two text files
  - **Hex Editor (mhexedit)** — A graphical editor for modifying binary files

• Building tools
  - **MULTI Builder (mbuild)** — A graphical interface for managing and building projects
  - **CodeBalance (codebalance)** — A graphical interface for automating the process of optimizing an executable for size or speed
  - **INTEGRATE (integrate)** — A graphical utility for configuring tasks, connections, and kernel objects across multiple address spaces when using the INTEGRITY RTOS
The MULTI Integrated Development Environment

- **Linker Directives File Editor (mldedit)** — A graphical editor for creating and modifying linker directives files

- **Debugging tools**
  - **MULTI Debugger (multi)** — A graphical source-level debugger
  - **EventAnalyzer (mevgui)** — A graphical viewer for monitoring the complex real-time interactions of an embedded RTOS such as INTEGRITY or ThreadX
  - **ResourceAnalyzer (wperf)** — A graphical viewer for monitoring the CPU and memory usage of an embedded system running the INTEGRITY RTOS
  - **Script Debugger (mscriptdbg)** — A graphical debugger for writing, recording, and debugging scripts containing MULTI commands
  - **Serial Terminal (mterminal)** — A serial terminal emulator for connecting to serial ports on embedded devices

- **Miscellaneous and administrative tools**
  - **Bug Report (gbugrpt)** — A utility for providing system configuration and tool version information to the Green Hills support staff
  - **Green Hills Probe Administrator (gpadmin)** — A graphical interface for configuring and managing Green Hills Debug Probes (Slingshots, Green Hills Probes, and/or SuperTrace Probes)
  - **Graphical Utilities (wgutils)** — A collection of utilities for analyzing and performing various operations on object files, libraries, and executables produced with the Green Hills toolchain
  - **MULTI License Administrator (mlmadmin)** — A graphical utility for managing Green Hills tools licenses
The MULTI Launcher

The MULTI Launcher \texttt{(mstart)} provides a convenient way to launch frequently used tools, to create new or access recently used files and projects, and to manage MULTI workspaces. All of the main MULTI components can be accessed using the following buttons:

- \textbullet{} \hspace{1em} Runs a shortcut or an action sequence in the current workspace. Also allows you to create a new workspace or create or edit a shortcut.
- \textbullet{} \hspace{1em} Opens the Builder on a recent or new project.
- \textbullet{} \hspace{1em} Opens the Debugger on a recent or new executable.
- \textbullet{} \hspace{1em} Opens the Editor on a recent or new file.
- \textbullet{} \hspace{1em} Opens the Checkout Browser on a recent or new checkout.
- \textbullet{} \hspace{1em} Opens the Connection Organizer or a recent or new target connection.
- \textbullet{} \hspace{1em} Opens a Serial Terminal using a recent or new connection.
- \textbullet{} \hspace{1em} Opens the EventAnalyzer (licensed separately).
- \textbullet{} \hspace{1em} Opens the ResourceAnalyzer (licensed separately).
- \textbullet{} \hspace{1em} Closes the MULTI Launcher (UNIX only by default).
- \textbullet{} \hspace{1em} Shows/hides the detail pane of the Launcher.

You can also launch the Green Hills License Administrator and (if installed) the Green Hills Probe Administrator from the \textbf{Utilities} menu.

During development, you can use the MULTI Launcher as a convenient centralized window manager. You can access any of your open MULTI windows from the \textbf{Windows} menu of the Launcher.
MULTI Workspaces

The MULTI Launcher allows you to create and use workspaces. A MULTI workspace is a virtual area where the tools, files, and actions required for a particular project can be organized, accessed, and executed.

A workspace is typically created for each top-level project and includes a working directory and a group of related actions — for example, opening a project in the MULTI Builder, connecting to a target, or performing a shell command. Actions are grouped into action sequences, so that a single mouse click can perform all the actions in the specified action sequence.

For more information, see Chapter 5, “Using MULTI Workspaces and Shortcuts” in the MULTI: Editing Files and Configuring the IDE book.
The MULTI 4.0 Document Set

The primary documentation for using MULTI is provided in the following books:

- **MULTI: Getting Started** — Describes how to install MULTI and obtain a license, and takes you through creating, building, and debugging an example project.

- **MULTI: Editing Files and Configuring the IDE** — Describes how to use the MULTI Editor, how to use a version control system with MULTI, how to use the MULTI Launcher, and how to configure and license the MULTI IDE.

- **MULTI: Building Applications** — Describes how to use the MULTI Builder and compiler drivers and the tools that compile, assemble, and link your code. Also describes the Green Hills implementation of supported high-level languages.

- **MULTI: Configuring Connections** — Describes how to set up your target debugging interface for use with MULTI and how to configure connections to your target.

- **MULTI: Debugging** — Describes how to use the MULTI Debugger and its associated tools.

These books, and any others you may have received (for example, for the INTEGRITY or ThreadX operating system, or for the Green Hills Debug Probes) are available in the following formats:

- A printed book
- Online help, accessible from most MULTI windows via the Help → Manuals menu (see “Online Help” on page 10)
- An electronic PDF, available in the manuals subdirectory of your installation

**Note** New or updated information may have become available while this book was in production. For additional material that was not available at press time, or for revisions that may have become necessary since this book was printed, please check your CD-ROM for Release Notes, README files, and other supplementary documentation.
The MULTI Debugger can be used to debug applications that reside on native, embedded, or simulated targets. To debug these applications, MULTI must first connect to the target through a debug server that runs on the host. Each distribution of the MULTI Integrated Development Environment (IDE) includes multiple debug servers, each of which supports connections to a specific set of probes, in-circuit emulators, monitors, and/or target simulators. This book describes how to set up your specific debugging interface for use with MULTI and how to configure connections to your target using the appropriate debug server. Specifically:

- Chapter 2, “Setting Up Your Target Hardware”, explains procedures you may need to perform if you are using an embedded hardware target. In such cases, you may need to configure your embedded target using a target setup script. MULTI provides default setup scripts for most supported targets. Chapter 2 explains how to access these setup scripts and, if necessary, how to customize them for your particular target.

- Chapters 3-8 each provide connecting instructions for the targets supported by a specific debug server. Because each of the target systems supported by MULTI has different features and limitations, the debug servers that support connections with these various targets also have different features, commands, and options, which are documented in chapters 3-8.

- Appendix A, “Green Hills Debug Server Command and Scripting Reference”, documents the commands that can be issued to Green Hills debug servers and describes the scripting language and conventions used for writing target setup scripts in previous versions of MULTI. The scripting language described in this appendix has been deprecated beginning with MULTI 4.0. The new scripting conventions are described in Chapter 8, “Using MULTI Scripts” in the MULTI: Debugging book.

**Note** The general procedure for connecting to targets using MULTI’s debug servers is described in Chapter 3, “Connecting to Your Target” in the MULTI: Debugging book. The present book supplements that chapter with specific instructions about configuring the debug servers and targets that are supported for ARM processors. (See the next section for a list of theses supported targets.)
1. Introduction

Note In this book, the word target indicates either an embedded or a simulated target. If you are developing in a native environment, MULTI generally “connects” transparently through a simple debug server. Because this happens automatically and because there are usually no configuration options that can be set for such connections, native connections are not discussed in this book. If there are useful connection options available for your particular native connection type, they will be documented in the MULTI: Building Applications book for your specific native environment, or, for native Linux environments, in the MULTI: Developing for Linux and GNU book.

Supported Target Connections for ARM

The following table lists the target connections supported for ARM. For detailed information about each type of target connection, see the appropriate chapter, as identified in the table.

<table>
<thead>
<tr>
<th>Debugging interface, monitor, or simulator</th>
<th>Green Hills debug server</th>
<th>Debug server chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agilent Processor Probe</td>
<td>hpserv</td>
<td>Chapter 4, “Agilent Processor Probe (hpserv) Connections”</td>
</tr>
<tr>
<td>Angel Monitor</td>
<td>rdiserv</td>
<td>Chapter 5, “ARM Remote Debug Interface (rdiserv) Connections”</td>
</tr>
<tr>
<td>ARM Multi-ICE Probe</td>
<td>rdiserv</td>
<td>Chapter 5, “ARM Remote Debug Interface (rdiserv) Connections”</td>
</tr>
<tr>
<td>ARMulator Simulator</td>
<td>rdiserv</td>
<td>Chapter 5, “ARM Remote Debug Interface (rdiserv) Connections”</td>
</tr>
<tr>
<td>EPI JEENI Probe</td>
<td>rdiserv</td>
<td>Chapter 5, “ARM Remote Debug Interface (rdiserv) Connections”</td>
</tr>
<tr>
<td>Green Hills Debug Probe</td>
<td>mpserv</td>
<td>Chapter 3, “Green Hills Debug Probe (mpserv) Connections”</td>
</tr>
<tr>
<td>(Slingshot, Green Hills Probe, or SuperTrace Probe)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDRT</td>
<td>rtserv</td>
<td>Chapter 6, “INDRT (rtserv) Connections”</td>
</tr>
</tbody>
</table>
Supported Target Connections for ARM

<table>
<thead>
<tr>
<th>Debugging interface, monitor, or simulator</th>
<th>Green Hills debug server</th>
<th>Debug server chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macraigor On-Chip Debugging (OCD)</td>
<td>ocdserv</td>
<td>Chapter 7, “Macraigor On-Chip Debugging (ocdserv) Connections”</td>
</tr>
<tr>
<td>(OCDemon Wiggler or OCDemon Raven)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simulator for ARM (simarm)</td>
<td>simarm</td>
<td>Chapter 8, “Simulator for ARM (simarm) Connections”</td>
</tr>
</tbody>
</table>

**Note** This book does not document how to make connections to OS targets or use MULTI to perform OS debugging. If you are using MULTI with the Green Hills INTEGRITY RTOS, see the INTEGRITY document set you received with your INTEGRITY distribution. If you are using another supported OS target, see the appropriate book from the list below.

- *MULTI: Developing for Linux and GNU*
- *MULTI: Developing for OSE*
- *MULTI: Developing for ThreadX*
- *MULTI: Developing for Tornado*

**Note** The lists above may not be comprehensive. Please contact Green Hills Technical Support if your particular debugging interface, monitor, simulator, or OS does not appear in the table above.
Online Help

The MULTI online help system provides three different types of online help:

- **Full online manuals** — You can access an indexed hypertext version of any MULTI manual by selecting it from the list that appears in the Help → Manuals menu.

- **Context-sensitive help** — Many MULTI windows and dialog boxes are linked to specific sections of the online manuals. To view the page in online help that documents an active window or dialog box:
  - (Windows) Press F1.
  - (UNIX) Press F1 or Help.

If you are using the MULTI Editor, you can also open context-sensitive help about a button or a menu item by selecting Help → Identify and then clicking the button or selecting the menu item.

- **Debugger command help** — You can obtain help information about a specific MULTI Debugger command by typing help command_name in the Debugger command pane. This will open the online version of the MULTI: Debugging book on the section that documents the specified command. You can also type usage command_name to print to the command pane the basic syntax of the specified command.

Viewing Help on Windows Systems

Windows systems display online help via the Microsoft HTML Help viewer. You can view only one manual at a time with the HTML Help viewer.

There are two panes in the HTML Help viewer. The right-hand pane displays the contents of a selected help page. You can click any underlined link that appears in right-hand pane to jump to pages about other related topics. The left-hand pane provides the following three navigation tabs:

- **Contents** — Displays the chapter and section headings for the manual being viewed. Click a plus or minus icon to show or hide headings for embedded sections. Click a heading to display the associated help page in the right-hand pane.

- **Index** — Displays index entries for the manual. Double-click an index entry to display its help page in the right-hand pane.
• **Search** — Provides an interface that allows you to search for specific words in the manual. Enter a word or phrase in the text box and click **List Topics** to display a list of pages related to your search string. Double-click any entry in the list to display the entry’s help page in the right-hand pane.

**Viewing Help on UNIX Systems**

Depending on your specific UNIX system, MULTI will automatically choose one of the following two methods for viewing online help:

• **Oracle Help for Java** — Displays hypertext version of manuals, including tables of contents and index entries, and provides full text searching across multiple manuals, with the ability to sort and rank search results.

This viewer may take up to 30 seconds to initialize and is only available on systems running Red Hat Linux 7.1 or higher, Solaris 2.6 or higher, or HP-UX 10.20 or 11.x. Other UNIX systems must use browser-based online help. If problems occur when MULTI is attempting to start the Oracle Help for Java application, you will see a message prompting you to use browser-based help (see below).

The Java Runtime Environment that is included with MULTI has been configured to use fonts common to most X Window servers. However, some X Window servers may not display Oracle Help for Java properly. To configure fonts for the Java Runtime Environment, edit the `font.properties` file that is located in the `jre/lib` directory of your MULTI installation (for more information, see the Sun Microsystems Java web site).

• **Web Browser Help** — Displays hypertext version of manuals, including tables of contents and index entries. In GUI mode, **netscape** is the default browser. If **netscape** is installed on your system and the executable is in your path, no further configuration should be necessary. In non-GUI mode, a copy of **lynx** running inside of an **xterm** will be used as the default browser. If you have **xterm** in your path, no further configuration should be necessary.

For information about using an alternate web browser, see “Using a Custom Web Browser with UNIX” in Chapter 8, “Configuring and Customizing MULTI” in the *MULTI: Editing Files and Configuring the IDE* book.

**Note** To change your online help settings, choose **Config → Options → General** tab, then click **Help**.
1. Introduction

Conventions Used in This Book

All Green Hills documentation assumes that you have a working knowledge of your host operating system and its conventions, including its command line and graphical user interface (GUI) modes. For example, you should know how to use basic commands, how to open, save, and close files, and how to use a mouse and standard menus.

Green Hills documentation uses a variety of notational conventions to present information and describe procedures. These conventions are described below.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold type</strong></td>
<td>Indicates a:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Filename or pathname</td>
<td>C:\MyProjects</td>
</tr>
<tr>
<td></td>
<td>- Command</td>
<td>setup command</td>
</tr>
<tr>
<td></td>
<td>- Option</td>
<td>-G option</td>
</tr>
<tr>
<td></td>
<td>- Window title</td>
<td>the Task window</td>
</tr>
<tr>
<td></td>
<td>- Button name</td>
<td>the Browse button</td>
</tr>
<tr>
<td></td>
<td>- Menu name or menu choice</td>
<td>the File menu</td>
</tr>
<tr>
<td><strong>Italic type</strong></td>
<td>Indicates that the user should replace the text in italics with an appropriate argument, command, filename, or other value.</td>
<td>-o filename</td>
</tr>
<tr>
<td><strong>Ellipsis (...)</strong></td>
<td>Indicates that the preceding argument or option can be repeated zero or more times.</td>
<td>debugbutton [name]...</td>
</tr>
<tr>
<td>(in command line instructions)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Greater than sign ( &gt; )</strong></td>
<td>Represents a prompt. Your actual prompt may be a different symbol or string. The &gt; prompt helps to distinguish input from output in examples of screen displays.</td>
<td>&gt; print Test Test</td>
</tr>
</tbody>
</table>
Conventions Used in This Book

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipe (</td>
<td>)</td>
<td>Indicates that one (and only one) of the parameters or options separated by the pipe or pipes should be specified.</td>
</tr>
<tr>
<td>(in command line instructions)</td>
<td></td>
<td>(The square brackets indicate that an argument is optional. If an argument is specified, however, it must be either a command or a group.)</td>
</tr>
<tr>
<td>square brackets ( [ ] )</td>
<td>Indicate optional arguments, commands, options, and so on. You can either include or omit the enclosed elements. The square brackets should not appear in your actual command.</td>
<td><strong>.macro name [list]</strong></td>
</tr>
<tr>
<td>(in command line instructions)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following command description demonstrates the use of some of these typographical conventions.

**gxyz [-option]... filename**

The formatting of this command indicates that:

- The command **gxyz** should be entered as shown.
- The option **-option** should either be replaced with one or more appropriate options or be omitted.
- The word **filename** should be replaced with the actual filename of an appropriate file.

The square brackets and the ellipsis should not appear in the actual command you enter.
Chapter 2

Setting Up Your Target Hardware

This Chapter Contains:

- Installing Your Target Hardware
- Configuring Your Target Hardware
- Specifying Setup Scripts
This chapter describes how to set up your target hardware for use with MULTI. This chapter is not relevant if you are connecting to a simulated target.

## Installing Your Target Hardware

Before you can configure your embedded target for use with MULTI, you must install your hardware and any necessary software. Your installation procedure will vary according to your particular system. For detailed instructions, see your hardware documentation and the chapter in this book that discusses your particular type of target (see “Supported Target Connections for ARM” on page 8 for a list of supported targets and the relevant chapter for each).

**Note** The Green Hills debug servers documented in this book are installed automatically when you install MULTI. For basic information about installing MULTI, see the *MULTI: Getting Started* book.

## Configuring Your Target Hardware

Before beginning your first debugging session, you should ensure that your target is configured properly. For many embedded targets, you will need to run a target setup script to prepare your target for use with MULTI. You may also need to edit the linker directives files that provide MULTI with information about your target’s memory map. In most cases, you can generate the required setup and linker directives files by creating your projects with the **New Project Wizard**, as described in the next section.

## Using the New Project Wizard to Configure and Test Your Target

The **New Project Wizard** is a tool that creates a new project structure, generates default setup script and linker directives files (if applicable), and Connection Methods based on information you provide about your target system. In most cases, running a default setup script created by the wizard configures your target properly.

After you have run the default setup script (if the wizard created a setup script for your project), you can use one of the example projects available in the **New Project Wizard** to test whether your target is configured properly. If you are able to compile, download, and debug a test application from an example...
Configuring Your Target Hardware

If your debug connection is working correctly and no additional target configuration is required.

Note If no example project exists in the New Project Wizard for your exact processor-board combination, or if no default setup scripts are created by the wizard for your target system, you might need to create customized setup script and/or linker directives files. See “Creating Customized Setup Script and Linker Directives Files” on page 20 for more information.

To use the New Project Wizard to create an example project that contains the necessary setup script and linker directives files (if any) for your specific target, and that you can use to test your target configuration and connection, perform the following steps:

1. From the MULTI Launcher, click and select Create Project. This opens the New Project Wizard.

2. On the first screen of the wizard, select your Processor, Operating System, Endian, and Board Name. If your exact target combination is not listed, select the processor-board combination that is most similar to yours, and click Next.

3. On the second screen of the wizard, select a language and then choose one of the available Example projects from the Project type list. Click Next to accept your language and project type selections and the default project directory and name.
2. Setting Up Your Target Hardware

4. On the remaining screens, select appropriate options for your project. You can click **Finish** any time it is available to accept the default options on the remaining screens. (For more information about all of the **New Project Wizard** screens and options, see “Creating a New Project: The New Project Wizard” in Chapter 2, “The MULTI Builder” in the *MULTI: Building Applications* book for your processor family.)

When you click **Finish**, the **New Project Wizard** closes and a MULTI Builder opens and displays the files created by the wizard for your project. These files may include setup script (**.mbs**) and linker directives (**.ld**) files if they are required for the target you specified.

5. Select the project file for your newly created project (**project_name.gpj**) and click ![Build](image) to build the project.

You can now test your target configuration by downloading and debugging the example program. To connect to your target and begin debugging:

1. With the project file still selected, click ![Open](image) in the Builder to open the MULTI Debugger.

2. In the Debugger window, click ![Connection Chooser](image), which opens a **Connection Chooser** window.
Configuring Your Target Hardware

Select from the pull-down list of available Connection Methods a connection that is appropriate for your target. In the example shown here, a Green Hills Debug Probe (mpserv) connection has been selected.

3. You must edit your selected Connection Method before you use it for the first time. To edit the Connection Method:
   a. Click to open the Connection Editor for the highlighted connection.
   b. Set the required options in the Connection Editor. (See “The Connection Editor” in Chapter 3, “Connecting to Your Target” in the MULTI: Debugging book for more information.)
   c. Click OK to save your settings and close the Connection Editor.

4. Click Connect in the Connection Chooser.

If you encounter problems debugging the example program, you may need to create customized setup and linker directives files for your target, as described in the next section.
Creating Customized Setup Script and Linker Directives Files

If your particular processor-board combination is not available in the New Project Wizard, you may need to create customized setup script and linker directives files for your specific target. You may also need to create customized files if your processor-board combination is available in the New Project Wizard but you experience difficulties compiling, downloading, or debugging a New Project Wizard example program.

If you need to create customized setup and linker directives files for your target, you must generate an example project for the New Project Wizard processor-board combination that is most similar to yours (see “Using the New Project Wizard to Configure and Test Your Target” on page 16). Edit the files created by the wizard to make them suitable for your specific target. To customize the files:

1. Edit the setup script (.mbs) file as necessary.
2. Edit applicable linker directives (.ld) files as necessary.
3. Test your debugging environment.

The following sections provide general instructions for performing these three steps.

Note

Target setup script files in previous versions of MULTI had .dbs extensions. Appendix A, “Green Hills Debug Server Command and Scripting Reference”, demonstrates the revised debug server commands and scripting language. Beginning with MULTI 4.0, the debug server scripting language has been deprecated. Setup script files now have an .mbs extension and use MULTI commands and scripting conventions. The MULTI scripting language and conventions are described in Chapter 8, “Using MULTI Scripts” in the MULTI: Debugging book. The MULTI 4.0 New Project Wizard creates .mbs scripts. The following instructions assume that you are using .mbs scripts. However, this release is backward compatible with the legacy .dbs scripts.

If you have developed projects in an earlier version of MULTI and want to use your legacy .dbs setup scripts with your MULTI 4.0 installation, the procedure for customizing your setup script is similar to the steps provided. However, the commands are different. See the documentation that accompanied the previous version of MULTI you used to create the .dbs scripts for examples that are
appropriate for .dbs scripts. Also, see “Specifying Setup Scripts” on page 28 for information about how to specify use of these scripts in MULTI 4.0.

**Step One: Editing the Setup Script**

After the **New Project Wizard** has created a project for a processor-board combination that is similar to your target system, the MULTI Builder displays the files created for the new project. Navigate to `resource.gpj` and look for a setup script file called `xserv_standard.mbs`, where `xserv` represents the debug server that supports your specific debugging interface, as listed in the table below. The default setup script file for the server that supports Green Hills Debug Probe connections, for example, is `mpserv_standard.mbs`.

<table>
<thead>
<tr>
<th>Debugging interface, monitor, or simulator</th>
<th>Debug server name (substitute for <code>xserv</code>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agilent Processor Probe</td>
<td><code>hpserv</code></td>
</tr>
<tr>
<td>Angel Monitor</td>
<td><code>rdiserv</code></td>
</tr>
<tr>
<td>ARM Multi-ICE Probe</td>
<td><code>rdiserv</code></td>
</tr>
<tr>
<td>ARMulator Simulator</td>
<td><code>rdiserv</code></td>
</tr>
<tr>
<td>EPI JEENI Probe</td>
<td><code>rdiserv</code></td>
</tr>
<tr>
<td>Green Hills Debug Probe</td>
<td><code>mpserv</code></td>
</tr>
<tr>
<td>(Slingshot, Green Hills Probe, or SuperTrace Probe)</td>
<td></td>
</tr>
<tr>
<td>INDRT</td>
<td><code>rtserv</code></td>
</tr>
<tr>
<td>(Run-time OS debugging)</td>
<td></td>
</tr>
<tr>
<td>Macraigor On-Chip Debugging</td>
<td><code>ocdserv</code></td>
</tr>
<tr>
<td>(OCD)</td>
<td></td>
</tr>
<tr>
<td>(OCDemon Wiggler or OCDemon Raven)</td>
<td></td>
</tr>
</tbody>
</table>

If an `.mbs` script for the server you want to use appears in `resource.gpj`, you may still need to edit the file to make it suitable for your specific target. If no `.mbs` script appears for the appropriate debug server, your target may not require a setup script. However, if your processor-board combination is not available in the **New Project Wizard** and you use an alternate combination, you may need to create a setup script for your specific target, even if the wizard does not create one for the example project.
Editing a setup script is a trial-and-error process. When editing your setup script, keep the following points in mind:

- If you have connected MULTI to your target (see Chapter 3, “Connecting to Your Target” in the *MULTI: Debugging* book for full instructions), you can use the MULTI Debugger command pane to confirm the success or failure of each command individually instead of trying to debug an entire script. However, you should not start debugging until you have completely finished setting up your target system. Debugging with a board that has not been properly set up can leave the target in an uncertain state. You can also use the Script Debugger to help you debug your setup scripts. See Chapter 8, “Using MULTI Scripts” in the *MULTI: Debugging* book for more information.

- Some commands cannot be tested individually because they must be executed within a certain time period in relation to other commands. For example, a watchdog timer may need to be disabled seconds after a reset command. It is difficult to type in the two commands that disable the watchdog timer before the timer expires. In such cases, put the relevant commands into a small script and run the script from the command pane using the < command.

- In addition to MULTI Debugger commands, you can use the debug server commands listed in Appendix A, “Green Hills Debug Server Command and Scripting Reference” in an .mbs setup script if you precede each debug server command with the MULTI target command. In addition to the general debug server commands listed in the Appendix, other debug server commands may also be available for your specific debugging interface. See “Additional Debug Server Commands” on page 144 for a listing of where to find descriptions of the additional commands (if any) available for your debug server. These commands also need to follow a target command if used in an .mbs script. (All of these commands can be used directly in .dbs scripts.)

To edit a setup script to make it suitable for your system:

**Note** The commands and examples below are for .mbs scripts.

1. From the Builder, double-click the .mbs setup script to be edited (for example, mpserv_standard.mbs) to open it in the MULTI Editor.
2. Determine whether your board can initialize itself, and then proceed as indicated in the following:
• If your target does not have a valid ROM image that initializes the target upon reset, skip to item number 3.

• If your target has a valid ROM image that initializes the target upon reset, delete the contents of the .mbs setup script you are editing. Replace the .mbs setup script with the command sequence shown in the following (or an equivalent command sequence).

```
# Reset and halt the board
target rst
# Let the ROM image run
target run
# Give the ROM image enough time to set up board
target sleep 3
# Halt the board to get ready for debugging
target halt
```

**Note** The parameter of the `sleep` command might need to be adjusted, depending on how long your board takes to set itself up.

Save the .mbs setup script and skip to “Step Two: Editing Linker Directives Files” on page 25.

3. Verify that your setup script begins with a command that resets the target, such as the MULTI Debugger `reset` or `target rst` commands. All targets must be halted before any debugging activity begins, or unpredictable results might occur.

4. Configure your target’s memory controller based on your board’s memory resources.

If your memory controller and memory resources are already properly configured, skip to item number 6. The following are general steps for configuring memory using a setup script:

a. Determine what memory resources your board has by answering the following:
   - How fast and how big is the board’s memory, and where do you want to map it?
   - Does the board have SRAM? If so, where is it?
   - Does the board have DRAM? If so, where is it, and where is the DRAM controller for it?
2. Setting Up Your Target Hardware

- Does the DRAM controller need refresh timing information or knowledge of any special modes the DRAM chips may have, such as Synchronous DRAM?

- Does the board require a peripheral memory base register to access memory controllers or other on-chip peripherals?

b. If your processor requires you to set up the base register before you can access your memory controllers or other on-chip peripherals, set the base register.

c. Using your processor’s documentation and memory resources, determine which memory-related registers you must set. Additionally, determine what values those registers must have to properly configure your memory resources.

d. Copy the commands necessary to configure your memory resources into your .mbs setup script. (See Chapter 30, “Debugger Commands” in the MULTI: Debugging book for a complete list of available MULTI commands. The debug server commands documented in this book can also be used, if preceded by the MULTI target command.)

5. Disable any interrupt sources that can disrupt the setup or destabilize the board’s memory. The following steps provide a general procedure for disabling interrupt sources:

a. Determine whether your processor has any interrupt sources that might disturb your debugging session.

b. Using your processor’s documentation, determine which registers affect interrupt sources. Then, determine the values those registers must have to disable the interrupt sources.

c. Copy the commands necessary to disable interrupt sources into your .mbs setup script. (See Chapter 30, “Debugger Commands” in the MULTI: Debugging book for a complete list of available MULTI commands. The debug server commands documented in the present book can also be used, if preceded by the MULTI target command.)

6. Verify that your .mbs setup script contains all of the commands required to prepare the target system, and then save it.
Example 1. Setup Script

// MULTI Server Script
// Netsilicon NET+40 setup script

target rst

// Set CPSR such that:
// Supervisor mode is on
// Thumb mode is off
// IRQ and FIQ interrupts are disabled
target reg cpsr=d3

target run

target sleep 2

target halt

// Turn off write protection for the flash memory

target bar0=0xffc00010

target bar0val=m $$bar0

// Flip bit 1 to off

target bar0val=bar0val & 0xfffffffd

target m $$bar0=$$bar0val

Step Two: Editing Linker Directives Files

If your board has a different memory map than the board you selected with the New Project Wizard, you must edit the memory.ld and standalone_config.ld linker directives files. The memory.ld file defines the memory map for the target board. The standalone_config.ld file defines the constants that other linker directives files use to define the program layout. You must define these constants according to the layout of your target. For example, standalone_config.ld should define the size of your target’s stack and heap.

You may also need to edit the standalone_ram.ld, standalone_romcopy.ld, or standalone_romrun.ld files used to build your program.
The general procedure for editing linker directives files is provided in the following. For more information about linker directives files, see the *MULTI: Building Applications* book for your processor family.

1. In the MULTI Builder, navigate to the .ld file you want to edit. The linker directives files are found under resource.gpj.
2. Double-click the .ld file in the MULTI Editor and edit the file as necessary.
3. Save the edited .ld file.
4. Rebuild your project by selecting the project (.gpj) file in the Builder and clicking .

**Step Three: Testing Your Debugging Environment**

After you have customized your setup script (.mbs) and linker directives (.ld) files, perform the subsequent steps to test whether your debugging environment is configured properly.

**Tip** You should test your debugging environment before attempting to download and debug a program.

1. Verify that all hardware connections are tight and secure, and that both the board and your hardware interface (if applicable) are powered.
2. Connect MULTI to your target as follows:
   a. Open the **Connection Chooser** by clicking from the MULTI Launcher and selecting **Connect** from the drop-down menu that appears.

   You can also open the Connection Chooser by clicking the button from a MULTI Debugger or Builder window.
Configuring Your Target Hardware

b. From the drop-down list of available Connection Methods, select a connection that is appropriate for your target. In the following example, a **Green Hills Debug Probe (mpserv)** connection has been selected.

c. You must edit your selected Connection Method before you use it for the first time. To edit the Connection Method:
   
   i. Click **to open the Connection Editor** for the highlighted connection.
   
   ii. Set the required options in the **Connection Editor**. (For more information, see Chapter 3, Connecting to Your Target in the **MULTI: Debugging** book, and the chapter in this book that corresponds to your target.)
   
   iii. Click **OK** to save your settings, and close the **Connection Editor**.
   
   d. Click **Connect** in the **Connection Chooser**.

3. Use the < command to run your setup script file. For example, to run the setup script file **mpserv_standard.mbs**, enter the following command in the MULTI command pane:

   `< src/mpserv_standard.mbs

4. To test whether reads and writes to the register are reaching the target:

   a. Select a general purpose register (for example, **r1**).

   b. Read the register and note its value. For example, enter the following in the Debugger command pane:

      `$r1`

   c. Write a different value to the same register. For example, enter the following in the Debugger command pane:

      `$r1=0xdeadbeef`
2. Setting Up Your Target Hardware

   d. Read the register again and see if it has changed to the new value. If it has, the target connection is able to alter registers.

   5. Test whether you can access the target memory through your debugging interface. We recommend that you test memory locations where you are downloading a program. To test the target memory:

      a. Read a memory location and note its value. For example, enter the following in the Debugger command pane:

         \texttt{memread 4 0x20000}

      b. Write a different value to the same memory location. For example, enter the following in the Debugger command pane:

         \texttt{memwrite 4 0x20000 0xdeadbeef}

      c. Read the memory location again and see if it has changed to the new value. If it has, the debugging interface is successfully accessing the target memory.

   \noindent \textbf{Note} You can also use the graphical Memory Tester to test your target memory. See Chapter 16, “Testing Target Memory” in the \textit{MULTI: Debugging} book for more information.

   If your target configuration appears to be functioning correctly, you are ready to begin debugging. For more information about using the MULTI Debugger, see the \textit{MULTI: Debugging} book.

\section*{Specifying Setup Scripts}

After you have used the \textbf{New Project Wizard} to generate a setup script for your target or have created a customized setup script based on a default script created by the wizard, you should run the script prior to every download to ensure that your target is clean, stable, and properly configured. The procedure for specifying and running target setup scripts varies slightly depending on how you are connecting to your target, and what type of setup script file you are using.

Target setup script files in previous versions of MULTI had .dbs extensions. Appendix A, “Green Hills Debug Server Command and Scripting Reference”, demonstrates the revised debug server commands and scripting language. Beginning with MULTI 4.0, the debug server scripting language has been
Specifying Setup Scripts

deprecated. Setup script files now have an .mbs extension and use MULTI commands and scripting conventions. The MULTI scripting language and conventions are described in Chapter 8, “Using MULTI Scripts” in the MULTI: Debugging book. MULTI 4.0 supports both .dbs and .mbs script files. Therefore, when you specify a setup script you must also specify which scripting language the script uses, so that MULTI interprets the script correctly. You can do this from the Connection Editor by selecting the appropriate scripting language radio button (see “Using MULTI (.mbs) Setup Scripts When Connecting to Your Target” on page 29).

If you are connecting using a command line instruction or a Custom Connection Method, the syntax for specifying a .dbs script is slightly different than that for specifying an .mbs script. The subsequent sections summarize the various ways to specify and run setup scripts. For detailed instructions on locating, creating, or editing a setup script file for your particular system, see “Using the New Project Wizard to Configure and Test Your Target” on page 16.

Using MULTI (.mbs) Setup Scripts When Connecting to Your Target

The method for specifying an .mbs setup script at the time of connection varies depending upon the procedure you use to connect MULTI to your target. The various methods are:

- To run an .mbs setup script every time you connect using a particular Standard Connection Method, specify the filename of the script in the Target Setup Script field of the Connection Editor for the Connection Method and select the MULTI radio button immediately below the field.

![Target Setup Script](image)

If you are using a default Connection Method created by the New Project Wizard, the necessary setup script file for your processor-board combination (if applicable) is specified automatically and the MULTI button will be selected.

- To run an .mbs setup script and connect to your target using a Custom Connection Method:
2. Setting Up Your Target Hardware

- If you are editing the Connection Method using the **Custom Connection Editor**, specify the filename of the target setup script in the **MULTI Target Setup Script** field.

- If you are entering the connection command using the **Start a Custom Connection** field of the **Connection Chooser**, precede the debug server command with the command `setup=filename` (where `filename` is the `.mbs` setup script filename). Click **Connect** to continue.

- To run an `.mbs` setup script from the Debugger command pane, use the following syntax:

  ```
  connect setup=filename.mbs dbserv [args]... [opts]...
  ```

  where `filename.mbs` is the setup script filename, `dbserv` is the name of the debug server to be used, and `args` and `opts` are appropriate arguments for your debug server and target.

**Using Legacy Debug Server (.dbs) Setup Scripts When Connecting to Your Target**

The method for specifying a `.dbs` setup script at the time of connection varies depending upon the procedure you use to connect MULTI to your target. The various methods are:

- To run a `.dbs` setup script every time you connect using a particular Standard Connection Method, specify the filename of the target setup script in the **Target Setup Script** field of the **Connection Editor** for the Connection Method. Select the **Legacy** radio button immediately below the field.

- To run a `.dbs` setup script and connect to your target using a Custom Connection Method, include the `-setup filename.dbs` debug server option in the command you enter into the **Start a Custom Connection** field of the **Connection Chooser** and then click **Connect**. The debug server option for specifying `.dbs` setup scripts must proceed the debug server command. See the appropriate debug server chapter later in this book for more information about connecting to your specific target this way.

- To run a `.dbs` setup script from the Debugger command pane, use the following syntax:
connect dbserv -setup filename.dbs [args]… [opts]…

where filename.dbs is the setup script filename, dbserv is the name of the debug server to be used, and args and opts are appropriate arguments for your debug server and target.

Running Setup Scripts Manually

In addition to running setup scripts as part of the connecting process, you can also run setup scripts manually at other times using any of the following methods:

• (MULTI .mbs scripts) Run your setup script file manually from the Debugger command pane using the < command.

• (MULTI .mbs scripts) Use the MULTI setup=filename.mbs command. If this command is used with a connection command, the setup file is ran prior to downloading and debugging. The setup command can also be used without a specific script name if you are connected and specified a setup script when you connected. The script you specified for the connection is ran if you issue the setup command with no specified file. See Chapter 30, “Debugger Commands” in the MULTI: Debugging book for more information about the setup command.

• (Legacy .dbs scripts) Use the target script filename.dbs command from the Debugger command pane.
Green Hills Debug Probe (mpserv) Connections
The Green Hills family of debug probes includes the Slingshot, the Green Hills Probe, and the SuperTrace Probe. These advanced hardware debugging instruments communicate with your target over a standard JTAG or BDM test port and allow you to use the MULTI Debugger to control, debug, and test your target CPU. Slingshot, Green Hills Probe, and SuperTrace Probe target connections are all supported through the *mpserv* debug server. For complete instructions on how to set up these probes and make target connections using *mpserv*, see the *Green Hills Debug Probes User’s Guide* that came with your probe.
This Chapter Contains:

- Supported Targets and Features for Agilent Processor Probe (hpserv) Connections
- Agilent Processor Probe Target System Requirements
- Checking Your Agilent Firmware
- Installing Your Agilent Processor Probe
- Troubleshooting Your Agilent Processor Probe Setup
- Connecting to Your Target with an Agilent Processor Probe
- Using the Agilent Processor Probe with a Multiple-Device JTAG Scan Chain
- Trace Support
- Additional Commands for Agilent Processor Probe (hpserv) Connections
This chapter supplements the general target connection information contained in Chapter 2, “Setting Up Your Target Hardware” of this book and Chapter 3, “Connecting to Your Target” in the MULTI: Debugging book with specific information for Agilent Processor Probe connections.

**Supported Targets and Features for Agilent Processor Probe (hpserv) Connections**

You can connect MULTI to an Agilent Processor Probe to provide comprehensive debugging capabilities for the following ARM processors:

- ARM/Thumb

When used with an Agilent Processor Probe, MULTI uses the `hpserv` debug server to connect over Ethernet and provide JTAG or BDM debugging on supported microprocessors, as shown in the following diagram.

When used with an E5900B probe and Agilent firmware with support for multiple-device JTAG scan chains, `hpserv` supports debugging of targets with multiple cores. See “Using the Agilent Processor Probe with a Multiple-Device JTAG Scan Chain” on page 49 for more information.

When used with an ARM target with an Embedded Trace Macrocell (ETM) and an appropriate Agilent Processor Probe, MULTI supports trace collection. For more information about collecting and using trace data, see Chapter 19, “Collecting and Using Trace Data” in the MULTI: Debugging book.

**Agilent Processor Probe Target System Requirements**

To use MULTI with an Agilent Processor Probe, your development environment must include:
Checking Your Agilent Firmware

Your Agilent Processor Probe must be loaded with the firmware required for your target CPU. The following table lists the Agilent firmware codes supported by MULTI for ARM processors.

<table>
<thead>
<tr>
<th>CPU</th>
<th>Firmware ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>E3459</td>
</tr>
</tbody>
</table>

You can determine which Agilent firmware code is installed on a probe by using the Agilent Processor Probe command `ver`, which generates output in the form shown below. (For emphasis, the firmware code appears in bold type in this example.)

If your Agilent firmware code does not match one of the supported types listed in the table above, you may need to change the firmware. For information about how to do this, see your Agilent documentation.

Installing Your Agilent Processor Probe

The following is a basic procedure for installing the Agilent Processor Probe. For more detailed instructions, refer to your Agilent documentation.
1. If you are using an E5900A probe, connect the wide ribbon cable from the probe to the target interface module (TIM) board. (If you are using a E5900B probe, the TIM is internal to the probe.)

2. Connect the small ribbon cable from the TIM board to the target debugging connector.

3. Connect the target board and the probe to a power supply.

4. Connect the probe to a terminal using a normal serial cable. Do not use a null modem cable. If you are using an E5900B or E5904B probe you will need to use the serial adapter that came with your probe.

5. Using the `lan` command on the probe serial connection, configure the IP address, gateway, and any other network options that need to be changed.

6. Connect the probe to your Ethernet network.

7. Power cycle the probe so that the Ethernet configuration changes take effect.

You are now ready to configure your target as described in “Using the New Project Wizard to Configure and Test Your Target” on page 16.

**Note** The `hpserv` debug server is automatically installed when you install MULTI. (For basic information about installing MULTI, see the *MULTI: Getting Started* book.)

### Troubleshooting Your Agilent Processor Probe Setup

If you experience problems with your Agilent Processor Probe connection after configuring your system as described in “Using the New Project Wizard to Configure and Test Your Target” on page 16, consider the following troubleshooting questions, which may help you locate the problem:

- Does the probe have power?
- Does the target board have power?
- Are all connections secure?
- Is the cable connecting the TIM board to the target connected to the proper connector in the correct orientation?
- Is the Ethernet cable securely connected to both the Ethernet connector on the probe and to the hub?
• Are the link lights on the probe and the hub on?
• Can you telnet to the probe? If not, you may need to connect to the probe on the serial interface and configure its Ethernet interface. Consult your Agilent documentation.

Connecting to Your Target with an Agilent Processor Probe

After you have installed and configured your system, you are ready to connect to your target and begin debugging. To help you connect to your target quickly and easily, MULTI allows you to create and save Connection Methods that correspond to your particular host and target systems and your desired debugging options.

For general instructions on how to create and use Connection Methods, see Chapter 3, “Connecting to Your Target” in the MULTI: Debugging book. The information in the following sections supplements the instructions provided there with information that is specific to Agilent Processor Probe connections.

Creating a Standard Agilent Processor Probe (hpserv) Connection Method

When creating a new Standard Agilent Processor Probe Connection Method, select Agilent Processor Probe (hpserv) as the connection type in the Create New Connection Method dialog box.

For detailed information about creating new Standard Connection Methods, see “Creating a Standard Connection Method” in Chapter 3, “Connecting to Your Target” in the MULTI: Debugging book.
Using the Agilent Processor Probe (hpserv) Connection Editor

**Note** The **Agilent Processor Probe (hpserv) Connection Editor** does not support multiple-core communications. To communicate with processors on a multiple-device scan chain, you must connect through a Custom Connection Method, as described in “Using Custom Agilent Processor Probe (hpserv) Connection Methods” on page 45. For more information about debugging multiple cores, see “Using the Agilent Processor Probe with a Multiple-Device JTAG Scan Chain” on page 49.

In addition to the generic fields that appear on all Connection Editors for Standard Connection Methods, the **Agilent Processor Probe (hpserv) Connection Editor** includes **Connection**, **Advanced**, and **Debug** tabs that provide settings and options specific to your target and host operating systems.
When the **Connection Editor** is first displayed after you create a new Connection Method, the settings and options on the tabs are set to default values. Settings and options that are not available on your host operating system may appear dimmed. Some of the fields may require user input before the Connection Method functions.

All of the fields on the **Connection**, **Advanced**, and **Debug** tabs of the **Agilent Processor Probe (hpserv) Connection Editor** are described in detail below. (See “The Connection Editor” in Chapter 3, “Connecting to Your Target” in the **MULTI: Debugging** book for a description of the other fields and options, which appear on all **Connection Editors**.)
### Agilent Processor Probe (hpserv) Connection Settings

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probe Name or IP Address</td>
<td>Specifies the hostname or IP address of your Agilent Processor Probe. This is the only field that requires user input.</td>
</tr>
<tr>
<td>Processor</td>
<td>Specifies the target processor type.</td>
</tr>
<tr>
<td>Endian</td>
<td>Specifies the target byte order. Unless you specify <strong>Big</strong> or <strong>Little</strong>, hpserv will use the default endian setting for your processor type. Use this field if you want to override the default setting.</td>
</tr>
</tbody>
</table>

### Agilent Processor Probe (hpserv) Advanced Settings

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Sections:</td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>BSS</td>
<td></td>
</tr>
<tr>
<td>Breakpoint Step</td>
<td></td>
</tr>
<tr>
<td>System Calls</td>
<td></td>
</tr>
<tr>
<td>Ignore Power Failure Reports</td>
<td></td>
</tr>
<tr>
<td>Available with 603, 7xx, and 82x Processors:</td>
<td>Enable Hardware Breakpoint by Disabling Software Breakpoints</td>
</tr>
<tr>
<td>Available with 7400 Processors:</td>
<td>Invalidate Cache on Breakpoint Set or Remove</td>
</tr>
</tbody>
</table>
| Probe Configuration (key1=value1, key2=value2, etc): | }
Warning Use this tab carefully, since changing the advanced options from their default settings can cause problems with your connection.

<table>
<thead>
<tr>
<th>Apply Flash/ROM Debugging Defaults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sets options that are appropriate for debugging code that is burned into ROM on the target. Specifically, clicking this button disables downloading of any program sections and disables system calls by clearing the Text, Data, BSS, and System Calls check boxes.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Load Sections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controls which sections of your program will be downloaded to the target, as follows:</td>
</tr>
<tr>
<td>• <strong>Text</strong> — If this box is checked, the <code>.text</code> (code) sections of your program will be downloaded to the target. This box is checked by default.</td>
</tr>
<tr>
<td>• <strong>Data</strong> — If this box is checked, the <code>.data</code> (initialized data) sections of your program will be downloaded to the target. This box is checked by default.</td>
</tr>
<tr>
<td>• <strong>BSS</strong> — If this box is checked, the <code>.bss</code> (uninitialized data) sections of your program will be cleared. This box is not checked by default.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Breakpoint Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determines the single instruction step method.</td>
</tr>
<tr>
<td>If this box is checked, single instruction steps will be implemented by placing a software breakpoint on the next instruction to be executed. If it is not checked, hardware single step will be used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enables host-based system calls.</td>
</tr>
<tr>
<td>If this box is selected, system calls are enabled and are implemented with a single software breakpoint. This box is checked by default.</td>
</tr>
<tr>
<td>If you are running from ROM and software breakpoints cannot be used, it may be desirable to disable host-based system calls by clearing this box.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ignore Power Failure Reports</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disables detection of loss of target power. By default, <code>hpserv</code> will print an error message and return an error code to MULTI when the probe detects that target power is lost. Selecting this check box disables the error message and makes <code>hpserv</code> ignore the loss of power.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enable Hardware Breakpoint by Disabling Software Breakpoints</th>
</tr>
</thead>
<tbody>
<tr>
<td>This option does not apply to ARM processors.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Invalidate Cache on Breakpoint Set or Remove</th>
</tr>
</thead>
<tbody>
<tr>
<td>This option does not apply to ARM processors.</td>
</tr>
</tbody>
</table>
Agilent Processor Probe (hpserv) Connections

Probes Configuration (key1=value1, key2=value2, etc)

Allows you to implement probe configuration options. Most Agilent probes have configuration options that can be set using the probe cf command. Configuration options specified in this field will be set at the start of the hpserv connection process.

Agilent Processor Probe (hpserv) Debug Settings

<table>
<thead>
<tr>
<th>Connection</th>
<th>Advanced</th>
<th>Debug</th>
</tr>
</thead>
</table>

- **Log Host-Target Debug Output**
  - StdErr/StdOut
  - File

  ```
  hpserv.log
  ```

  Choose...

  Other Options:

  Warning: Do not change the settings on the debug tab unless you are instructed to do so by Green Hills Technical Support.

Log Host-Target Debug Output

Enables logging of all communications between hpserv and the Agilent Processor Probe. Logging is disabled by default.
Connecting to Your Target with an Agilent Processor Probe

<table>
<thead>
<tr>
<th>StdErr/StdOut</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allows you to specify the destination for host-target debug output. These fields will be dimmed unless Log Host-Target Debug Output is selected.</td>
<td></td>
</tr>
<tr>
<td>If you choose StdErr/StdOut, host-target debug output will be directed to the console.</td>
<td></td>
</tr>
<tr>
<td>If you select File, host-target debug output will be directed to the file you specify in the text field. You may enter a filename directly into this text field or click Choose to browse the file system. A default filename will be suggested.</td>
<td></td>
</tr>
</tbody>
</table>

Other Options

Allows you to add other, optional arguments directly to the command line. You should only use this field if directed to do so by Green Hills Technical Support.

Using Custom Agilent Processor Probe (hpserv) Connection Methods

To connect to your Agilent Processor Probe with a Custom Connection Method, type the command given below, with the appropriate parameters and options, into the Start a Custom Connection dialog box and click Connect.

```
[mode=conn_mode] [setup=filename.mbs] hpserv -hostname hostname [options]…
```

where:

- **mode=conn_mode** is optional and specifies the connection mode.
  - Appropriate values for conn_mode are download, attach, and boardsetup. If the mode is not specified, MULTI will connect in download mode. See Chapter 18, “Debugging in Download, Attach, and Board Setup Modes” in the MULTI: Debugging book for more information about connection modes.

- **setup=filename.mbs** is optional and specifies the target setup script. This argument is optional because not all targets require setup scripts.  
  - **Note** This option can only be used to specify an .mbs setup script. If you are using an older .dbs script, you must use the -setup option, which must come after the hpserv command. See “Configuring Your Target Hardware” on page 16 for more information about setup scripts.

- **-hostname hostname** is required and specifies the hostname or IP address of the target.
• `options` can be any non-conflicting combination of the `hpserv` options listed in “Other Options for Custom Agilent Processor Probe (hpserv) Connection Methods” on page 46.

**Note** You can also enter the above connection command from the Debugger’s command pane, where it must be preceded by the `connect` command.

See “Custom Connection Methods” in Chapter 3, “Connecting to Your Target” in the MULTI: Debugging book for more information about Custom Connections.

For example connection commands, see “Example Custom Agilent Processor Probe (hpserv) Connection Methods” on page 48.

**Other Options for Custom Agilent Processor Probe (hpserv) Connection Methods**

In addition to the specific options described in the custom connection section above, you can also use any non-conflicting combination of the options listed below when creating or using a Custom Agilent Processor Probe (hpserv) Connection Method.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-bpstep</code></td>
<td>Implements single stepping with software breakpoints rather than the Agilent probe’s single-step mechanism.</td>
</tr>
<tr>
<td><code>-bss</code></td>
<td>Sets downloading of <code>.bss</code> (uninitialized data) sections. By default, <code>.bss</code> downloading is disabled because the default Green Hills startup code clears <code>.bss</code> sections. This option allows you to clear <code>.bss</code> sections upon download.</td>
</tr>
<tr>
<td><code>-cf key=value</code></td>
<td>Sets a probe configuration item at the start of the <code>hpserv</code> connection process. For example, if your target will not operate reliably with the default JTAG clock speed used by the Agilent Processor Probe, you may need to use the switch <code>-cf speed=2</code> to use a JTAG clock divisor of 2.</td>
</tr>
</tbody>
</table>
**-core**

Configures `hpserv` for a multiple-core JTAG scan chain.

The `-core` option must occur once on the command line for each core on the scan chain that you want to control with the Agilent probe. The relative order of `-core` and `-device` options on the `hpserv` command line must match the order of devices on the JTAG scan chain. Any core-specific command line options that occur after each occurrence of `-core` and before the next apply to the corresponding core on the scan chain. Currently, the only core-specific option is the `-bpstep` option.

For more information about support for multiple-device JTAG scan chains with `hpserv`, see “Using the Agilent Processor Probe with a Multiple-Device JTAG Scan Chain” on page 49.

**-data**

Sets initialized data downloading. By default, initialized data (.data) sections are always downloaded, so this switch should never be used.

**-device ir_bits**

Configures the Agilent probe to set an unknown device on the JTAG scan chain into bypass mode.

This option should be used if your target’s JTAG scan chain includes devices that the Agilent probe does not understand (for example, processors that do not match the firmware installed on the probe or Ethernet controllers). The `ir_bits` argument specifies the size of the unknown device’s JTAG instruction register. The relative order of `-core` and `-device` options on the `hpserv` command line must match the order of devices on the JTAG scan chain.

For more information about support for multiple-device JTAG scan chains with `hpserv`, see “Using the Agilent Processor Probe with a Multiple-Device JTAG Scan Chain” on page 49.

**- endian [0|1]**

Controls the byte order of the target. Normally, `hpserv` can detect the byte order of the target. With some targets, however, the byte order must be specified on the command line. The switch `- endian 0` forces little endian operation, and `- endian 1` forces big endian operation.

**-loadall**

Enables downloading of all program sections. By default, all sections are loaded except for uninitialized data (.bss) sections, which are cleared by the default Green Hills startup code. The `-loadall` option allows you to download all sections, including .bss.

**-log filename**

Creates a log of actions performed by `hpserv` and stores it in the specified file. If you have problems using `hpserv` with your target, this log file may help Green Hills Technical Support find a solution.
### -nobss
Disables uninitialized data (.bss) clearing. This option is enabled by default and is documented for backward compatibility only.

### -nodata
Disables downloading of initialized data (.data) sections.

### -noload
Disables downloading of any of the program sections in an executable file.
This option is useful when debugging an executable that is already loaded on the target. For example, use this option when debugging an executable in ROM or flash memory. This option does not disable stack setup processes, such as setting the stack pointer, writing the stack to memory, or setting the system call breakpoint.

### -notext
Disables downloading of program sections containing executable code (.text sections).

### -nozeropic
Does not allow the position independent code (PIC) base address to be 0.

### -p port
Specifies the port with which to connect on the probe. This defaults to port 6470.

### -setup filename
Specifies a setup script file that is run by the hpserv command interpreter before each download.
This option can only be used to specify .dbs setup scripts. See “Specifying Setup Scripts” on page 28 for information about specifying .mbs scripts.

### -zeropic
Allows the position independent code (PIC) base address to be 0.

---

### Example Custom Agilent Processor Probe (hpserv) Connection Methods

The following are examples of commands that can be entered into the Start a Custom Connection dialog box. (These example commands can also be entered, preceded by the connect command, from the MULTI command pane.)

#### Example 1.

The following command connects hpserv to an Agilent Processor Probe with the hostname yak5 using the default options.
Example 2.

The following command connects `hpserv` to an Agilent Processor Probe with the IP address 192.168.101.88 that is listening on port 10000 and ensures that the commands in the setup file `init.mbs` are automatically run immediately prior to every host-to-target download.

```
setup=init.mbs hpserv -hostname 192.168.101.88 -p 10000
```

Example 3.

The following command connects `hpserv` to an Agilent Processor Probe with the hostname `kabob2` and configures the probe for a JTAG scan chain with a core followed by an unknown device and then another core. The first core is configured to use breakpoint single stepping instead of the Agilent probe’s built-in single-step mechanism.

```
hpserv -hostname kabob2 -core -bpstep -device 4 -core
```

Example 4.

The following command connects `hpserv` to an Agilent E5904B probe with the hostname `aztec2`, configures the probe to debug code on an ARM 966ES Integrator core module, and ensures that the commands in the Legacy-style setup file `integrator-ap.dbs` are automatically run immediately prior to every host-to-target download.

```
hpserv -hostname aztec2 -cf proc=ARM966ES -cf speed=rtck -setup integrator-ap.dbs
```

Using the Agilent Processor Probe with a Multiple-Device JTAG Scan Chain

Recent firmware for the Agilent E5900B and E5904B probes allows `hpserv` to configure these probes to communicate with processors on a multiple-device JTAG scan chain. This is accomplished by communicating with one device at a time and placing all other devices in bypass mode.

Although the scan chain can contain any number of different types of CPUs, the Agilent probes are not capable of debugging different types of CPUs on one scan chain. For example, if your target’s scan chain includes both a MIPS and
an ARM processor, you can either use an Agilent probe with MIPS firmware and control the MIPS processor while the ARM processor is placed in bypass mode, or use an Agilent probe with ARM firmware and control the ARM processor while the MIPS processor is placed in bypass mode. All processors you want to control with the Agilent probe must use the same byte order.

For information about the options available for configuring hpserv to debug a target with a multiple-device JTAG scan chain, see -core and -device in “Other Options for Custom Agilent Processor Probe (hpserv) Connection Methods” on page 46. For an example of the use of these options, see “Example Custom Agilent Processor Probe (hpserv) Connection Methods” on page 48.

**Note** The Agilent Processor Probe (hpserv) Connection Editor does not support multiple-core communications. To communicate with processors on a multiple-device scan chain, you must connect through a Custom Connection Method, as described in “Using Custom Agilent Processor Probe (hpserv) Connection Methods” on page 45.

**Trace Support**

MULTI supports trace data collection from ARM ETM targets with the following Agilent hardware:

- An Agilent E5904B integrated Processor Probe and Trace Port Analyzer.
- An Agilent 16700 series Logic Analyzer with E3456 high density probe adapter and an Agilent Processor Probe.
- An Agilent E5900A Processor Probe and an Agilent E5903A Trace Port Analyzer with an ARM ETM target adapter.

For information about using the MULTI trace features see Chapter 19, “Collecting and Using Trace Data” in the MULTI: Debugging book.

**Additional Setup Required to Collect ETM Trace with a Logic Analyzer**

To use an Agilent 16700 Logic Analyzer to collect ETM trace data with hpserv, the logic analyzer must be set up to work with the ETM before it can be used to collect trace information for the first time.

To configure the logic analyzer, use the Setup Assistant from a logic analyzer session to perform the following steps.
1. Select **ARM → ARM ETM → HP E9595A Option 002**.

2. Choose the options appropriate for your configuration in the rest of the setup dialogs.

3. Save the setup with the appropriate filename for the options you have selected, as listed in the table below.

<table>
<thead>
<tr>
<th>Setup Assistant options</th>
<th>Configuration filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>4- or 8-bit packets</td>
<td>/hplogic/configs/etmconfig</td>
</tr>
<tr>
<td>No time tags</td>
<td></td>
</tr>
<tr>
<td>16-bit packets</td>
<td>/hplogic/configs/etmconfig-16bit</td>
</tr>
<tr>
<td>No time tags</td>
<td></td>
</tr>
<tr>
<td>4- or 8-bit packets</td>
<td>/hplogic/configs/etmconfig-timetags</td>
</tr>
<tr>
<td>Time tags</td>
<td></td>
</tr>
<tr>
<td>16-bit packets</td>
<td>/hplogic/configs/etmconfig-timetags-16bit</td>
</tr>
<tr>
<td>Time tags</td>
<td></td>
</tr>
</tbody>
</table>

It is a good idea to close the window created by the **Setup Assistant** (the **ETM Data** window) before saving the setup, so that it will not be opened every time **hpserv** reconfigures the logic analyzer.

4. Complete steps 1-3 for each set of options (from those listed in the table in step 3) that you want to use.

When you use the logic analyzer, **hpserv** will load the setup configurations you have saved in steps 1-4 above. This process takes a few seconds and may result in a noticeable delay if the logic analyzer configuration must be changed as a result of setting a trigger.

In order to connect to the logic analyzer and collect trace information, a logic analyzer session must be running and the host name of the analyzer must be given as an argument to the **MULTI trace** command. For more information about the **trace** command, see Chapter 30, “Debugger Commands” in the **MULTI: Debugging** book.

**Dual ETM Trace Support**

MULTI now supports simultaneous tracing of dual ETM-enabled cores with the ARM trace multiplexer and dual Agilent E5904B probes. The following hardware is required:
4. Agilent Processor Probe (hpserv) Connections

- 2 Agilent E5904B Option 300 probes
- 1 Agilent E5903–66508 dual ETM buffer board
- A target with dual ETM-enabled ARM cores and an ARM trace multiplexer

The MULTI **dualtrace** command is used to start a dual core trace session. See Chapter 30, “Debugger Commands” in the **MULTI: Debugging** book for more information about the **dualtrace** command.

**Additional Commands for Agilent Processor Probe (hpserv) Connections**

The following commands, in addition to the commands listed in “Generic Debug Server Commands” on page 138, are available to **hpserv**, the Green Hills debug server that supports Agilent Processor Probe connections.

You can enter all of these commands directly into the **hpserv Target** window. You can also enter these commands into the MULTI Debugger command pane using the **target** command. All of the commands for **hpserv** are case-insensitive.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>**bpstep [on</td>
<td>off]**</td>
</tr>
<tr>
<td></td>
<td>If <strong>on</strong> is specified: Specifies that software breakpoints will be used to implementation single instruction steps.</td>
</tr>
<tr>
<td></td>
<td>If <strong>off</strong> is specified: Specifies that the default probe single-step mechanism will be used.</td>
</tr>
<tr>
<td></td>
<td>If no argument is specified: Prints the current single-step method.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>core [pid [command]]</strong></td>
<td>If only <strong>pid</strong> is specified: Changes the default active core to the processor identified by <strong>pid</strong>, where <strong>pid</strong> is the Object ID number in the MULTI <strong>Task Window</strong>.</td>
</tr>
<tr>
<td></td>
<td>If <strong>pid</strong> and <strong>command</strong> are specified: Sends the command to the processor identified by <strong>pid</strong>, where <strong>pid</strong> is the Object ID number in the MULTI <strong>Task Window</strong>.</td>
</tr>
<tr>
<td></td>
<td>If no arguments are specified: Displays the current default active core for the target window.</td>
</tr>
</tbody>
</table>
**hp command**
Sends the specified *command* directly to the probe. You need this command to access probe commands that are shadowed by *hpserv* commands.

When you give a command to **hpserv** that **hpserv** does not recognize, it passes the command to the Agilent probe’s command interpreter. In some cases (for example, with the **help** and **m** commands) the Agilent probe implements commands with the same name as **hpserv** commands. To use these probe commands from **hpserv**, you must use the **hp** command.

**reg [reg=[val]]**
Reads or writes to the specified registers.
If no arguments are specified: Lists the names and values of all target registers.
If a register name is specified: Prints the value of the specified register.
If a value (=val) is specified for a register: Sets the register to the specified value.
Values should be given in hexadecimal form with no leading 0x.

Example:

```plaintext
> reg r1
reg r1 is 0x0001a000
> reg r1=1
reg r1 is now 0x00000001
```

**rst [-m]**
Resets the target. The optional -m switch specifies that the target should enter the monitor after the reset.

**stepint [0 | 1]**
Turns on or off the processing of interrupts while performing single instruction steps.
If 0 is specified: Disables interrupts.
If 1 is specified: Enables interrupts.
If no argument is specified: Prints the current single-step interrupt processing setting.

**zeropic [on | off]**
Specifies whether or not the position independent code (PIC) base address can be 0.
If on is specified: Allows the position independent code (PIC) base address to be 0.
If off is specified: Does not allow the position independent code (PIC) base address to be 0.
If no argument is specified: Displays the current **zeropic** setting.
Chapter 5

ARM Remote Debug Interface (rdiserv) Connections

This Chapter Contains:

- Supported Remote Debug Interfaces (RDIs)
- Supported Processors for Remote Debug Interface (RDI) Connections
- ARM Remote Debug Interface (RDI) Hardware Breakpoint Support
- Installing Your Remote Debug Interface (RDI)
- Connecting to Your Target with a Remote Debug Interface (RDI)
- Creating a Standard Connection Method for Your Remote Debug Interface (RDI) Connection
- Using the Angel/JEENI (rdiserv) Connection Editor
- Using the ARM Multi-ICE (rdiserv) Connection Editor
- Using the ARMulator (rdiserv) Connection Editor
- Using Custom Remote Debug Interface (rdiserv) Connection Methods
- Additional Commands for Remote Debug Interface (rdiserv) Connections
- Troubleshooting Your Remote Debug Interface (rdiserv) Connection
- RDI Error Codes
- Using the Green Hills Tracer Module for the ARMulator
This chapter supplements the general target connection information contained in Chapter 2, “Setting Up Your Target Hardware” of this book and Chapter 3, “Connecting to Your Target” in the MULTI: Debugging book with specific connection information for the following ARM Remote Debug Interfaces (RDIs): Angel Monitors, EPI JEENI Probes, ARM Multi-ICE Probes, and the ARMulator Simulator.

**Supported Remote Debug Interfaces (RDIs)**

MULTI uses the `rdiserv` debug server to support target connections using the following types of Remote Debug Interfaces (RDIs): Angel Monitors, EPI JEENI Probes, ARM Multi-ICE Probes, and the ARMulator Simulator. These interfaces are briefly described below.

**Angel Debug Protocol (ADP) Targets: Angel Monitor and EPI JEENI Probe**

The Angel Debug Protocol (ADP) is a debug protocol used to communicate with the Angel Monitor and JTAG probes, such as the EPI JEENI, that have an ADP interface. ADP can be used to communicate with a target through a serial line, a combination of a serial and a parallel line, or over Ethernet. MULTI’s `rdiserv` debug server supports ADP connections to Angel Monitor and EPI JEENI Probe targets. For more information about these devices, see your hardware documentation.

**ARM Multi-ICE Probe**

Multi-ICE is a parallel port debug probe from ARM. It consists of a small box that connects to the parallel port of a Windows PC. This box has a JTAG interface that connects directly to the target JTAG port via a ribbon cable (and possibly an adapter). In order to use Multi-ICE with `rdiserv`, the Multi-ICE server must be running on the PC that is connected to the Multi-ICE. For more information about the Multi-ICE server, see your Multi-ICE documentation.

**Note** Multi-ICE is not supported by `rdiserv` on UNIX hosts.
ARMulator Simulator

The ARMulator is an instruction-accurate simulator of ARM processors. While the ARMulator is not fully cycle accurate, it does provide approximate cycle counts and other performance statistics. See the description of the `cycles` command in “Additional Commands for Remote Debug Interface (rdiserv) Connections” on page 85 for more information.

When simulating simple uncached ARM cores, the ARMulator generally provides accurate cycle counts, but with more complex cached ARM cores, cycle counts will not be accurate.

Supported Processors for Remote Debug Interface (RDI) Connections

The `rdiserv` debug server supports Angel Monitor, EPI JEENI Probe, ARM Multi-ICE Probe, and ARMulator target connections for the processors listed below.

<table>
<thead>
<tr>
<th>Processor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM6</td>
<td>ARM6 core model</td>
</tr>
<tr>
<td>ARM60</td>
<td>ARM60 model</td>
</tr>
<tr>
<td>ARM61</td>
<td>ARM61 model</td>
</tr>
<tr>
<td>ARM610</td>
<td>ARM6 core with MMU, WB, and cache</td>
</tr>
<tr>
<td>ARM7</td>
<td>ARM7 core model</td>
</tr>
<tr>
<td>ARM7D</td>
<td>ARM7 core with debug hardware</td>
</tr>
<tr>
<td>ARM7DM</td>
<td>ARM7 core with debug and multiplier</td>
</tr>
<tr>
<td>ARM7T-S</td>
<td>Synthesized ARM7 with thumb core model</td>
</tr>
<tr>
<td>ARM7TDM</td>
<td>ARM7 core with thumb, debug, and multiplier</td>
</tr>
<tr>
<td>ARM7TDI</td>
<td>ARM7 core with thumb, debug, and multiplier</td>
</tr>
<tr>
<td>ARM7TDI-S</td>
<td>Synthesized ARM7 core with thumb, debug, and multiplier</td>
</tr>
<tr>
<td>ARM70T</td>
<td>ARM70 with thumb and multiplier</td>
</tr>
<tr>
<td>ARM70T-S</td>
<td>Synthesized ARM70 with thumb and multiplier</td>
</tr>
<tr>
<td>ARM70</td>
<td>ARM70 model</td>
</tr>
<tr>
<td>ARM70D</td>
<td>ARM70 with debug hardware</td>
</tr>
<tr>
<td>ARM70DM</td>
<td>ARM70 with debug and multiplier</td>
</tr>
</tbody>
</table>
## ARM Remote Debug Interface (RDI) Hardware Breakpoint Support

The RDI API has no support for hardware instruction breakpoints. When used with RDI, the MULTI Debugger asks the RDI library to set a breakpoint on a particular instruction. The RDI library chooses whether to use a hardware or software breakpoint for each breakpoint. However, many RDI targets use...
a simple algorithm for choosing which breakpoints are set with hardware breakpoints. If you use the \texttt{-exechwbp rdiserv} command line argument, \texttt{rdiserv} attempts to take advantage of this algorithm to allow the user to set hardware breakpoints. This option will not work with all RDI targets. For example, the Multi-ICE RDI library uses a different hardware breakpoint allocation algorithm, so \texttt{-exechwbp} should not be used with Multi-ICE. The \texttt{-exechwbp} option has been tested with several Angel monitor targets and the EPI JEENI, but it is not guaranteed to work with any particular target.

The RDI API does support hardware data breakpoints, and \texttt{rdiserv} allows the use of hardware data breakpoints if the target supports them.

### Installing Your Remote Debug Interface (RDI)

Your particular RDI target may require hardware installation or other setup procedures. Consult your hardware documentation for detailed instructions. The sections below briefly describe some of the target system requirements and installation procedures for the interfaces supported by \texttt{rdiserv}.

#### Note

The \texttt{rdiserv} debug server and any necessary RDI target libraries are automatically installed when you install MULTI.

### Angel Monitor or JEENI Probe Installation and Setup

To make target connections and perform debugging using \texttt{rdiserv} and the Angel Debug Protocol (ADP), your target system must include:

- An Angel Monitor and a serial, serial + parallel, or Ethernet interface
  
  — or —

- A JTAG interface and an EPI JEENI probe connected to the target JTAG interface

Install your Angel Monitor or EPI JEENI Probe according to your hardware documentation instructions.
ARM Multi-ICE Probe Installation and Setup

To make target connections and perform debugging with MULTI and a Multi-ICE device, your target system must include:

- A JTAG interface
- A Multi-ICE connected to the target JTAG interface and the parallel port of a Windows host
- A Multi-ICE server running on the host that is connected to the Multi-ICE

You will need to install the Multi-ICE server software on the PC that is connected to the Multi-ICE. After installing the Multi-ICE server software, you must copy `multi-ice.dll` from the Multi-ICE server installation directory to your Green Hills tools directory. Make sure that you start the Multi-ICE server application before you attempt to connect to your target system with `rdiserv`. Also ensure that the Multi-ICE server is successfully communicating with the target system. The server should detect and display what type of ARM CPU is running on the target system. If it does not, refer to the ARM Multi-ICE documentation for information about configuring the server and troubleshooting.

ARMulator Installation and Setup

There are no additional target requirements or installation steps for ARMulator connections.

Connecting to Your Target with a Remote Debug Interface (RDI)

After you have installed and configured your system, you are ready to connect to your target and begin debugging. To help you connect to your target quickly and easily, MULTI allows you to create and save Connection Methods that correspond to your particular host and target systems and your desired debugging options.

For general instructions on how to create and use Connection Methods, see Chapter 3, “Connecting to Your Target” in the MULTI: Debugging book. The information in the following sections supplements the instructions provided there with information that is specific to the following types of ARM Remote Debug Interface (RDI) connections:
Creating a Standard Connection Method for Your Remote Debug Interface (RDI) Connection

When creating a new Standard Connection Method for your RDI connection, select the appropriate connection type in the Create New Connection Method dialog box, as indicated in the table below.

<table>
<thead>
<tr>
<th>RDI Connection type</th>
<th>Connection type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Angel Monitor</td>
<td>Angel/JEENI (rdiserv)</td>
</tr>
<tr>
<td>ARM Multi-ICE Probe</td>
<td>Multi-ICE (rdiserv)</td>
</tr>
<tr>
<td>ARMulator Simulator</td>
<td>ARMulator (rdiserv)</td>
</tr>
<tr>
<td>EPI JEENI Probe</td>
<td>Angel/JEENI (rdiserv)</td>
</tr>
</tbody>
</table>

For example, to create a new connection method for an Angel Monitor target, enter a name for your connection method and select Angel/JEENI (rdiserv) from the list of available connection types, as shown below.

For detailed instructions on creating new Standard Connection Methods, see “Creating a Standard Connection Method” in Chapter 3, “Connecting to Your Target” in the MULTI: Debugging book.
Using the Angel/JEENI (rdiserv) Connection Editor

In addition to the generic fields that appear on all Connection Editors for Standard Connection Methods, the Angel/JEENI (rdiserv) Connection Editor includes Connection, Download, Advanced, and Debug tabs that provide settings and options specific to your target and host operating systems.

When the Connection Editor is first displayed after you create a new Connection Method, the settings and options on the tabs are set to default values. Settings and options that are not available on your host operating system may appear dimmed. Some of the fields may require user input before the Connection Method functions.
All of the fields on the **Connection**, **Download**, **Advanced**, and **Debug** tabs of the **Angel/JEENI (rdiserv) Connection Editor** are described in detail below. (See “The Connection Editor” in Chapter 3, “Connecting to Your Target” in the *MULTI: Debugging* book for a description of the other fields and options, which appear on all **Connection Editors**.)

### Angel/JEENI (rdiserv) Connection Settings

<table>
<thead>
<tr>
<th>Processor</th>
<th>Specifies the target processor.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Little Endian</strong></td>
<td>Specifies the target byte order. By default, this box is checked, which sets little endian byte ordering.</td>
</tr>
</tbody>
</table>

**Connection**

- **Processor**: Specifies the target processor.
- **Little Endian**: Specifies the target byte order. By default, this box is checked, which sets little endian byte ordering.
5. ARM Remote Debug Interface (rdiserv) Connections

Serial Connection
Specifies that a serial connection to the target should be used. This radio button is mutually exclusive with the Ethernet connection to host button. If you select a serial connection (this is the default), the following option fields will also be available:

- **Serial Port** — Specifies which host serial port to use for Angel communications. The default serial port (used if you do not specify a port) for each supported operating system is listed below:
  - (Windows) COM1
  - (Solaris) /dev/ttya
  - (HP-UX) /dev/tty00
  - (Linux) /dev/ttyS0
- **Baud** — Specifies the baud rate for Angel communications. The default baud rate is 9600 baud.
- **With Parallel connection** — Enables serial + parallel Angel mode. If this box is selected, the Parallel Port field specifies which host parallel port to use.

Ethernet connection to host
Specifies that an Ethernet connection to the target should be used. This radio button is mutually exclusive with the Serial connection button. If you select an Ethernet connection, you must specify the IP hostname of the JEENI in the text box.

Angel/JEENI (rdiserv) Download Settings

<table>
<thead>
<tr>
<th>Connection</th>
<th>Download</th>
<th>Advanced</th>
<th>Debug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Sections:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ] Text</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ] Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ] BSS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Warning** Use this tab carefully, since changing the download options from their default settings can sometimes cause problems with your connection.

**Load Sections**
Controls which sections of your program will be downloaded to the target, as follows:

- **Text** — If this box is checked, the `.text` (code) sections of your program will be downloaded to the target. This box is checked by default.
- **Data** — If this box is checked, the `.data` (initialized data) sections of your program will be downloaded to the target. This box is checked by default.
- **BSS** — If this box is checked, the `.bss` (uninitialized data) sections of your program will be cleared. This box is not checked by default.

**Angel/JEENI (rdiserv) Advanced Settings**

<table>
<thead>
<tr>
<th>Connection</th>
<th>Download</th>
<th>Advanced</th>
<th>Debug</th>
</tr>
</thead>
</table>

- **Display configuration GUI**
  (Windows only) Invokes the RDI library’s configuration window. This window may be necessary to configure rarely used options that are not accessible from the Angel/JEENI (rdiserv) Connection Editor.

- **Use armd.map settings**
  Enables use of an `armd.map` file for describing target memory resources to the RDI library. This is rarely necessary.
  If this option is selected, `rdiserv` will load `armd.map` from the current working directory. The format of the `armd.map` file is specified by ARM.

**Warning** Use this tab carefully, since changing the advanced options from their default settings can cause problems with your connection.
### 5. ARM Remote Debug Interface (rdiserv) Connections

<table>
<thead>
<tr>
<th><strong>Use breakpoints when stepping</strong></th>
<th>Determines the single instruction step method. If this box is checked, single instruction steps will be implemented by placing a software breakpoint on the next instruction to be executed. If it is not checked, hardware single step will be used if it is available.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Service system calls</strong></td>
<td>Enables host-based system calls. If this box is selected, system calls are enabled and are implemented with a single software breakpoint. This box is checked by default. If you are running from ROM and software breakpoints cannot be used, it may be desirable to disable host-based system calls by clearing this box.</td>
</tr>
<tr>
<td><strong>Memory size</strong></td>
<td>Provides the RDI library with information about the size of target memory. This field is optional and is not supported by all RDI targets.</td>
</tr>
<tr>
<td><strong>Use RDI DLL</strong></td>
<td>Specifies an alternate RDI library for rdiserv to use instead of the default Angel RDI library.</td>
</tr>
</tbody>
</table>

### Angel/JEENI (rdiserv) Debug Settings

<table>
<thead>
<tr>
<th>Connection</th>
<th>Download</th>
<th>Advanced</th>
<th>Debug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use compatibility-node profiling</td>
<td>Force halt on these exceptions:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Error</td>
<td>Branch Through Zero</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Undefined</td>
<td>SWI</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Address Exception</td>
<td>FIQ</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data Abort</td>
<td>IRQ</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Prefetch Abort</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other Options:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Warning** Do not change the settings on the debug tab unless you are instructed to do so by Green Hills Technical Support.

<table>
<thead>
<tr>
<th>Use compatibility-mode profiling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Causes <code>rdiserv</code> to revert to the old <code>rdiserv</code> profiling method.</td>
</tr>
<tr>
<td>In some previous versions, <code>rdiserv</code> collected profiling data by periodically halting the target and sampling the PC. Checking this box will cause <code>rdiserv</code> to revert to that behavior. The new default profiling method requires profiling code to run on the target, collect PC samples, and then upload them to <code>rdiserv</code>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Force halt on these exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configures which exceptions should cause the target to halt and return control to the MULTI Debugger. Exceptions that are not checked in this list will be handled by an exception handler.</td>
</tr>
<tr>
<td>This feature may not work with all RDI targets. By default, it is disabled and the RDI library's default behavior is used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allows you to add other, optional arguments directly to the command line. You should only use this field if directed to do so by Green Hills Technical Support.</td>
</tr>
</tbody>
</table>

---

**Using the ARM Multi-ICE (rdiserv) Connection Editor**

In addition to the generic fields that appear on all Connection Editors for Standard Connection Methods, the **Multi-ICE (rdiserv) Connection Editor** includes **Connection, Download, Advanced**, and **Debug tabs** that provide settings and options specific to your target and host operating systems.
When the **Connection Editor** is first displayed after you create a new Connection Method, the settings and options on the tabs are set to default values. Settings and options that are not available on your host operating system may appear dimmed. Some of the fields may require user input before the Connection Method functions.

All of the fields on the **Connection**, **Download**, **Advanced**, and **Debug** tabs of the **Multi-ICE (rdiserv) Connection Editor** are described in detail below. (See “The Connection Editor” in Chapter 3, “Connecting to Your Target” in the *MULTI: Debugging* book for a description of the other fields and options, which appear on all **Connection Editors**.)
Multi-ICE (rdiserv) Connection Settings

<table>
<thead>
<tr>
<th>Processor</th>
<th>&lt;default&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Little Endian</td>
<td>✔</td>
</tr>
<tr>
<td>Host:</td>
<td></td>
</tr>
</tbody>
</table>

**Processor**
Specifies the target processor.

**Little Endian**
Specifies the target byte order. By default this box is checked to indicate little endian byte ordering.

**Host**
Specifies the hostname of the Windows computer to which the Multi-ICE Probe is connected. The Multi-ICE server software must be running on the specified computer.

If this field is left blank, rdiserv will assume the Multi-ICE server is running on the local computer.

Multi-ICE (rdiserv) Download Settings

<table>
<thead>
<tr>
<th>Connection</th>
<th>Download</th>
<th>Advanced</th>
<th>Debug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Section:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>✔ Text</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>✔ Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>✔ BSS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Green Hills Software, Inc. 69
Warning  Use this tab carefully, since changing the download options from their default settings can sometimes cause problems with your connection.

**Load Sections**
Controls which sections of your program will be downloaded to the target, as follows:
- **Text** — If this box is checked, the `.text` (code) sections of your program will be downloaded to the target. This box is checked by default.
- **Data** — If this box is checked, the `.data` (initialized data) sections of your program will be downloaded to the target. This box is checked by default.
- **BSS** — If this box is checked, the `.bss` (uninitialized data) sections of your program will be cleared. This box is not checked by default.

---

**Multi-ICE (rdiserv) Advanced Settings**

(Windows only) Invokes the RDI library’s configuration window. This window may be necessary to configure rarely used options that are not accessible from the Angel/JEENI (rdiserv) Connection Editor.

- **Display configuration GUI**
- **Use armsd.map settings**
- **Use breakpoints when stepping**
- **Service system calls**

Memory Size:

- **Use RDI DLL:**

---

Warning  Use this tab carefully, since changing the advanced options from their default settings can cause problems with your connection.

**Display configuration GUI**
(Windows only) Invokes the RDI library’s configuration window. This window may be necessary to configure rarely used options that are not accessible from the Angel/JEENI (rdiserv) Connection Editor.

**Use armsd.map settings**
Enables use of an `armsd.map` file for describing target memory resources to the RDI library. This is rarely necessary.

If this option is selected, `rdiserv` will load `armsd.map` from the current working directory. The format of the `armsd.map` file is specified by ARM.
### Use breakpoints when stepping

Determines the single instruction step method.

If this box is checked, single instruction steps will be implemented by placing a software breakpoint on the next instruction to be executed. If it is not checked, hardware single step will be used if it is available.

### Service system calls

Enables host-based system calls.

If this box is selected, system calls are enabled and are implemented with a single software breakpoint. This box is checked by default.

If you are running from ROM and software breakpoints cannot be used, it may be desirable to disable host-based system calls by clearing this box.

### Memory size

Provides the RDI library with information about the size of target memory. This field is optional and is not supported by all RDI targets.

### Use RDI DLL

Specifies an alternate RDI library for `rdiserv` to use instead of the default Angel RDI library.

---

## Multi-ICE (rdiserv) Debug Settings

<table>
<thead>
<tr>
<th>Connection</th>
<th>Download</th>
<th>Advanced</th>
<th>Debug</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="checkbox" alt="Use compatibility-mode profiling" /></td>
<td><img src="checkbox" alt="Force halt on these exceptions" /></td>
<td><img src="input" alt="Other Options" /></td>
<td></td>
</tr>
</tbody>
</table>

- **Force halt on these exceptions:**
  - Error
  - Undefined
  - Address Exception
  - Data Abort
  - Prefetch Abort
  - Branch Through Zero
  - SWI
  - FIQ
  - IRQ

---

*Green Hills Software, Inc.*
Warning Use this tab carefully, since changing the troubleshooting options from their default settings can sometimes cause problems with your connection.

<table>
<thead>
<tr>
<th>Use compatibility-mode profiling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Causes <code>rdiserv</code> to revert to the old <code>rdiserv</code> profiling method.</td>
</tr>
<tr>
<td>In some previous versions, <code>rdiserv</code> collected profiling data by periodically halting the target and sampling the PC. Checking this box will cause <code>rdiserv</code> to revert to that behavior. The new default profiling method requires profiling code to run on the target, collect PC samples, and then upload them to <code>rdiserv</code>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Force halt on these exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configures which exceptions should cause the target to halt and return control to the MULTI Debugger. Exceptions that are not checked in this list will be handled by an exception handler.</td>
</tr>
<tr>
<td>This feature may not work with all RDI targets. By default, it is disabled and the RDI library’s default behavior is used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allows you to add other, optional arguments directly to the command line. You should only use this field if directed to do so by Green Hills Technical Support.</td>
</tr>
</tbody>
</table>

Using the ARMutator (rdiserv) Connection Editor

In addition to the generic fields that appear on all Connection Editors for Standard Connection Methods, the ARMutator (rdiserv) Connection Editor includes Connection, Download, Advanced, and Debug tabs that provide settings and options specific to your target and host operating systems.
When the Connection Editor is first displayed after you create a new Connection Method, the settings and options on the tabs are set to default values. Settings and options that are not available on your host operating system may appear dimmed. Some of the fields may require user input before the Connection Method functions.

All of the fields on the Connection, Download, Advanced, and Debug tabs of the ARMulator (rdiserv) Connection Editor are described in detail below. (See “The Connection Editor” in Chapter 3, “Connecting to Your Target” in the MULTI: Debugging book for a description of the other fields and options, which appear on all Connection Editors.)
5. ARM Remote Debug Interface (rdiserv) Connections

ARMulator (rdiserv) Connection Settings

<table>
<thead>
<tr>
<th>Connection</th>
<th>Download</th>
<th>Advanced</th>
<th>Debug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>&lt;default&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Little Endian</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Processor**
Specifies the type of ARM processor that the ARMulator will simulate.

**Little Endian**
Specifies the byte order of the ARM processor that the ARMulator will simulate.

ARMulator (rdiserv) Download Settings

<table>
<thead>
<tr>
<th>Connection</th>
<th>Download</th>
<th>Advanced</th>
<th>Debug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Sections:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Warning** Use this tab carefully, since changing the download options from their default settings can sometimes cause problems with your connection.

**Load Sections**
Controls which sections of your program will be downloaded to the target, as follows:

- **Text** — If this box is checked, the `.text` (code) sections of your program will be downloaded to the target. This box is checked by default.

- **Data** — If this box is checked, the `.data` (initialized data) sections of your program will be downloaded to the target. This box is checked by default.

- **BSS** — If this box is checked, the `.bss` (uninitialized data) sections of your program will be cleared. This box is not checked by default.
ARMulator (rdiserv) Advanced Settings

<table>
<thead>
<tr>
<th>Display configuration GUI</th>
<th>Use armsd.map settings</th>
<th>Use breakpoints when stepping</th>
<th>Service system calls</th>
<th>Memory Size:</th>
</tr>
</thead>
</table>

**Warning** Use this tab carefully, since changing the advanced options from their default settings can cause problems with your connection.

### Display configuration GUI
(Windows only) Invokes the RDI library’s configuration window. This window may be necessary to configure rarely used options that are not accessible from the Angel/JEENI (rdiserv) Connection Editor.

### Use armsd.map settings
Enables use of an armsd.map file for describing target memory resources to the RDI library. This is rarely necessary.

If this option is selected, rdiserv will load armsd.map from the current working directory. The format of the armsd.map file is specified by ARM.

### Use breakpoints when stepping
Determines the single instruction step method.

If this box is checked, single instruction steps will be implemented by placing a software breakpoint on the next instruction to be executed. If it is not checked, hardware single step will be used if it is available.

### Service system calls
Enables host-based system calls.

If this box is selected, system calls are enabled and are implemented with a single software breakpoint. This box is checked by default.

If you are running from ROM and software breakpoints cannot be used, it may be desirable to disable host-based system calls by clearing this box.

### Memory size
Provides the RDI library with information about the size of target memory. This field is optional and is not supported by all RDI targets.
5. ARM Remote Debug Interface (rdiserv) Connections

Use RDI DLL
Specifies an alternate RDI library for rdiserv to use instead of the default Angel RDI library.

ARMulator (rdiserv) Debug Settings

<table>
<thead>
<tr>
<th>Connection</th>
<th>Download</th>
<th>Advanced</th>
<th>Debug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use compatibility-mode profiling</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Force halt on these exceptions:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error</td>
<td>Branch Through Zero</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Undefined</td>
<td>SWI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address Exception</td>
<td>HIQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Abort</td>
<td>IFQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prefetch Abort</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Other Options: [ ]

⚠️ **Warning** Use this tab carefully, since changing the troubleshooting options from their default settings can sometimes cause problems with your connection.

**Use compatibility-mode profiling**
Causes rdiserv to revert to the old rdiserv profiling method.

In some previous versions, rdiserv collected profiling data by periodically halting the target and sampling the PC. Checking this box will cause rdiserv to revert to that behavior. The new default profiling method requires profiling code to run on the target, collect PC samples, and then upload them to rdiserv.

**Force halt on these exceptions**
Configures which exceptions should cause the target to halt and return control to the MULTI Debugger. Exceptions that are not checked in this list will be handled by an exception handler.

This feature may not work with all RDI targets. By default, it is disabled and the RDI library’s default behavior is used.

**Other Options**
Allows you to add other, optional arguments directly to the command line. You should only use this field if directed to do so by Green Hills Technical Support.
Using Custom Remote Debug Interface (rdiserv) Connection Methods

You can create a Custom Remote Debug Interface (rdiserv) Connection Method by manually entering commands and options into the Connection Chooser instead of using the graphical Connection Editor.

The connection command syntax for your Custom Remote Debug Interface (rdiserv) connection will vary according to your type of RDI. The sections below give the exact command syntax for the following RDI targets:

- Angel Monitor or EPI JEENI Probe
- ARM Multi-ICE Probe
- ARMulator Simulator

For example custom connection commands, see “Example Custom Remote Debug Interface (RDI) Connections” on page 84.

Custom Angel Monitor or EPI JEENI Probe Connections with rdiserv

To connect to an Angel Monitor or EPI JEENI Probe with a Custom Connection Method, type the command given below, with the appropriate parameters and options, into the Start a Custom Connection dialog box and click Connect.

```
[mode=conn_mode] [setup=filename.dbs] rdiserv -adp [device] [options]…
```

where:

- **mode=conn_mode** is optional and specifies the connection mode. Appropriate values for conn_mode are download, attach, and boardsetup. If the mode is not specified, MULTI will connect in download mode. See Chapter 18, “Debugging in Download, Attach, and Board Setup Modes” in the MULTI: Debugging book for more information about connection modes.

- **setup=filename.mbs** is optional and specifies the target setup script. This argument is optional because not all targets require setup scripts.

**Note** This option can only be used to specify an .mbs setup script. If you are using an older .dbs script, you must use the -setup option, which must come after the rdiserv command. See “Configuring Your Target Hardware” on page 16 for more information about setup scripts.
5. ARM Remote Debug Interface (rdiserv) Connections

- **-adp** is required and specifies an Angel Debug Protocol connection.

- **device** is optional for serial and serial + parallel connections and specifies the serial port name. If no port is specified, **rdiserv** assumes (Windows) **COM1**, (Solaris) **/dev/ttya**, (HP-UX) **/dev/tty00**, or (Linux) **/dev/ttyS0**.

- **device** is required for Ethernet connections and specifies the hostname of the target or probe.

- **options** can be any non-conflicting combination of the **rdiserv** options listed in “Options for Custom Remote Debug Interface (rdiserv) Connections” on page 80.

**Note** You can also enter the above connection command from the Debugger’s command pane, where it must be preceded by the **connect** command.

See “Custom Connection Methods” in Chapter 3, “Connecting to Your Target” in the **MULTI: Debugging** book for more information about Custom Connections.

For example Angel Monitor and EPI JEENI Probe connection commands, see “Example Custom Remote Debug Interface (RDI) Connections” on page 84.

### Custom ARM Multi-ICE Probe Connections with rdiserv

To connect to a Multi-ICE Probe with a Custom Connection Method, type the command given below, with the appropriate parameters and options, into the **Start a Custom Connection** dialog box and click **Connect**.

```
[mode=conn_mode] [setup=filename.dbs] rdiserv -ice [hostname] [options]...
```

where:

- **mode=conn_mode** is optional and specifies the connection mode. Appropriate values for **conn_mode** are **download**, **attach**, and **boardsetup**. If the mode is not specified, MULTI will connect in download mode. See Chapter 18, “Debugging in Download, Attach, and Board Setup Modes” in the **MULTI: Debugging** book for more information about connection modes.

- **setup=filename.mbs** is optional and specifies the target setup script. This argument is optional because not all targets require setup scripts.
**Note** This option can only be used to specify an `.mbs` setup script. If you are using an older `.dbs` script, you must use the `-setup` option, which must come after the `rdiserv` command. See “Configuring Your Target Hardware” on page 16 for more information about setup scripts.

- `-ice` is required and specifies a Multi-ICE connection.
- `hostname` is optional. If no hostname is specified, `rdiserv` attempts to connect to a Multi-ICE server running on the local host.
- `options` can be any non-conflicting combination of the `rdiserv` options listed in “Options for Custom Remote Debug Interface (rdiserv) Connections” on page 80.

**Note** You can also enter the above connection command from the Debugger’s command pane, where it must be preceded by the `connect` command.

See “Custom Connection Methods” in Chapter 3, “Connecting to Your Target” in the `MULTI: Debugging` book for more information about Custom Connections.

For an example Multi-ICE Probe connection command, see “Example Custom Remote Debug Interface (RDI) Connections” on page 84.

### Custom ARMulator Connections with rdiserv

To connect to an ARMulator with a Custom Connection Method, type the command given below, with the appropriate parameters and options, into the Start a Custom Connection dialog box and click Connect.

```
[mode=conn_mode] rdiserv [-cpu processor] [options]…
```

where:

- `mode=conn_mode` is optional and specifies the connection mode. Appropriate values for `conn_mode` are `download`, `attach`, and `boardsetup`. If the mode is not specified, MULTI will connect in download mode. See Chapter 18, “Debugging in Download, Attach, and Board Setup Modes” in the `MULTI: Debugging` book for more information about connection modes.
- `-cpu processor` is optional and specifies which processor to simulate. For a list of valid values for `processor`, see the description of `-cpu` in “Options for
Custom Remote Debug Interface (rdiserv) Connections” on page 80. If the 
-cpu processor option is not used, an ARM7TDMI is simulated by default.

- options can be any non-conflicting combination of the rdiserv options listed 
in “Options for Custom Remote Debug Interface (rdiserv) Connections” 
on page 80.

Note You can also enter the above connection command from the Debugger’s 
command pane, where it must be preceded by the connect command.

See “Custom Connection Methods” in Chapter 3, “Connecting to Your 
Target” in the MULTI: Debugging book for more information about Custom 
Connections.

For an example ARMulator connection command, see “Example Custom 
Remote Debug Interface (RDI) Connections” on page 84.

Options for Custom Remote Debug Interface (rdiserv) Connections

The following options are available when creating a Custom Remote Debug 
Interface (rdiserv) Connection Method.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-adp</td>
<td>Specifies an Angel Debug Protocol (ADP) target connection.</td>
</tr>
</tbody>
</table>
| -baud rate | (For ADP connections only) Specifies the baud rate for an ADP serial connection. Valid baud rates are:  
• 9600 (Default)  
• 19200  
• 38400  
• 57600  
• 115200 |
| -bss    | Sets downloading of .bss (uninitialized data) sections. By default, .bss downloading is disabled because the default Green Hills startup code clears .bss sections. This option allows you to clear .bss sections upon download. |
### Using Custom Remote Debug Interface (rdiserv) Connection Methods

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-bigendian</strong></td>
<td>Specifies that the target is big endian. If this option is not used, little endian operation is assumed.</td>
</tr>
<tr>
<td><strong>-bpstep</strong></td>
<td>Implements single stepping with software breakpoints. Some RDI targets support the RDI API single step function. With these targets, the <strong>-bpstep</strong> option can be used to force rdiserv to implement single instruction steps with breakpoints.</td>
</tr>
<tr>
<td><strong>-config</strong></td>
<td>Launches the config dialog, which allows you to modify configuration settings using a GUI instead of command line switches. Some rarely used options can be set only by using this dialog. For other options, command line switches are recommended because MULTI will display a server timeout message if the config window is open for longer than the server timeout period. The timeout message can be ignored and should be dismissed after the RDI config window has been closed.</td>
</tr>
<tr>
<td><strong>-cpu processor</strong></td>
<td>Specifies the processor to simulate or the processor of the target system. In ADP mode, rdiserv will attempt to detect the CPU if this flag is omitted. In ARMulator and Multi-ICE modes, ARM7TDMI is assumed if this flag is omitted. Valid values for processor are: ARM6, ARM60, ARM61, ARM610, ARM7, ARM7D, ARM7DM, ARM7T-S, ARM7TDM, ARM7TDMI (Default), ARM7TDMI-S, ARM70T, ARM70T-S, ARM710, ARM710T, ARM720T, ARM730T, ARM8, ARM810, ARM9, ARM9E, ARM9TDMI, ARM920T, ARM922T, ARM925T, ARM940T, ARM946E-S, ARM966E-S, ARM10, SA-110, StrongARM, XScale</td>
</tr>
<tr>
<td><strong>-dll filename</strong></td>
<td>(For Windows hosts only) Causes rdiserv to attempt to use the specified RDI shared library instead of the default shared library for the selected mode.</td>
</tr>
<tr>
<td><strong>-ethernet</strong></td>
<td>(For ADP connections only) Specifies an Ethernet Angel connection. If this flag is used, the hostname can be specified anywhere on the command line.</td>
</tr>
</tbody>
</table>
### ARM Remote Debug Interface (rdiserv) Connections

#### -exechwp
Causes `rdiserv` to exploit an RDI algorithm to allow the user to set hardware breakpoints. This option will not work with all RDI targets. See “ARM Remote Debug Interface (RDI) Hardware Breakpoint Support” on page 58 for more information.

#### -ice
Specifies a Multi-ICE connection.

#### -memsizer size
Has the same effect as `-set MEMORYSIZE=size`. If your debug target needs this key to be set in the RDI configuration database, you can use this option to set it.

#### -nobrkError
- `nobrkUndefined`
- `nobrkAddressException`
- `nobrkDataAbort`
- `nobrkPrefetchAbort`
- `nobrkBranchThroughZero`
- `nobrkSWI`
- `nobrkFIQ`
- `nobrkIRQ`

These `-nobr` options cause `rdiserv` to attempt to configure the RDI target to halt on all exceptions except those you identify with `-nobr` options. If no `-nobr` options are used, `rdiserv` will not change the RDI target’s default behavior.

It is possible to configure some RDI targets so that they halt and give control to the MULTI Debugger when certain exceptions occur. The default behavior when an exception occurs is not specified by the RDI specification and can vary between targets. There is also no mechanism for querying the RDI target to determine which exceptions it is configured to halt on. In practice, most RDI targets do not halt on exceptions by default.

#### -nodata
Disables downloading of initialized data (.data) sections.
Using Custom Remote Debug Interface (rdiserv) Connection Methods

- **-noload**
  Disables downloading of any of the program sections in an executable file.
  This option is useful when debugging an executable that is already loaded on the target. For example, use this option when debugging an executable in ROM or flash memory. This option does not disable stack setup processes, such as setting the stack pointer, writing the stack to memory, or setting the system call breakpoint.

- **-notext**
  Disables downloading of program sections containing executable code (.text sections).

- **-oldprofiling**
  Some older versions of rdiserv perform profiling by periodically halting the target and sampling the program counter. rdiserv now supports a different type of profiling that uses program counter samples collected by a timer interrupt on the target. Use the **-oldprofiling** option to enable the old profiling method.

- **-parallel port**
  (For ADP connections only)
  Specifies ADP serial + parallel communications with the specified parallel port.
  Valid parallel port names are:
  - LPT1
  - LPT2
  In serial + parallel mode, and in the default (serial-only) mode, the serial port can be specified anywhere on the command line. If no serial port is specified the default of (Windows) COM1, (Solaris) /dev/ttya, (HP-UX) /dev/tty00, or (Linux) /dev/ttyS0 will be used.

- **-rom**
  Configures rdiserv to debug programs that are located in ROM. Currently, the **-rom** option only disables a breakpoint that is used to implement host I/O.

- **-set key=value**
  The RDI specification includes a basic database that is used to store configuration options. With some RDI targets, the database is loaded from a configuration file when the RDI target is initialized. Both rdiserv and the RDI target can read and write to the database. This mechanism is used by some targets to allow the user to configure various target-dependent settings that are not accessible through dedicated RDI interfaces.
  The **-set key=value** option allows you to set the value associated with any key in the database when rdiserv starts.
5. ARM Remote Debug Interface (rdiserv) Connections

- **setup file**
  Specifies a setup script file that is run by the rdiserv command interpreter before each download.
  This option can only be used to specify .dbs setup scripts. See “Specifying Setup Scripts” on page 28 for information about specifying .mbs scripts.

- **-usemapfile**
  Causes rdiserv to attempt to parse the armsd.map file and register its settings with the RDI target. The format of the armsd.map file is defined by ARM.

Example Custom Remote Debug Interface (RDI) Connections

The following are examples of commands that can be entered into the Start a Custom Connection dialog box. (These example commands can also be entered, preceded by the connect command, from the MULTI command pane.)

**Example 1. Angel/JEENI Serial Mode Connection**

The following command launches rdiserv in ADP mode and connects to an ADP target using serial communications on COM1 at 115200 baud.

```
rdiserv -adp COM1 -baud 115200
```

**Example 2. Angel/JEENI Ethernet Mode Connection**

The following command launches rdiserv in ADP mode and connects, using Ethernet communications, to the ADP target with the hostname myprobe.

```
rdiserv -adp -ethernet myprobe
```

**Example 3. Angel/JEENI Serial + Parallel Mode Connection**

The following command launches rdiserv in ADP mode and connects to an ADP target using serial communications on COM1 at 57600 baud and parallel communications on LPT1.

```
rdiserv -adp COM1 -parallel LPT1 -baud 57600
```

**Example 4. ARM Multi-ICE Probe Connection**

The following command launches rdiserv in Multi-ICE mode and connects to the Multi-ICE server running on the local machine, excludes all program sections from every host-to-target download, and ensures that the commands
Additional Commands for Remote Debug Interface (rdiserv) Connections

in the setup file setup.mbs are automatically run immediately prior to every host-target download.

setup=setup.mbs rdiserv -ice -noload

Example 5. ARMuulator Connection

The following command launches rdiserv in ARMuulator mode such that it simulates an ARM920T CPU.

rdiserv -cpu ARM920T

Additional Commands for Remote Debug Interface (rdiserv) Connections

The following commands, in addition to the commands listed in “Generic Debug Server Commands” on page 138, are available to rdiserv, the Green Hills debug server that supports Remote Debug Interface (RDI) connections.

You can enter all of these commands directly into the rdiserv Target window. You can also enter these commands into the MULTI Debugger command pane using the target command. All of the commands for rdiserv are case-insensitive.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bpstep [on</td>
<td>off]</td>
</tr>
<tr>
<td></td>
<td>Most RDI targets support an RDI API function for single stepping the target. If this function is supported by your target, it will be used by default. However, you can use the command bpstep on to force rdiserv to implement single instruction steps with breakpoints.</td>
</tr>
<tr>
<td></td>
<td>Using the bpstep command with no arguments will print the current bpstep setting.</td>
</tr>
<tr>
<td>cpuspeed [speed]</td>
<td>Allows the user to set the speed in MHz of the CPU simulated by the ARMuulator. This value does not affect the operation of the ARMuulator and is currently only used when making profiling calculations.</td>
</tr>
<tr>
<td></td>
<td>This command is only available in ARMuulator mode when the ARMuulator includes the Green Hills Tracer module. For more information about the Green Hills Tracer module, see “Profiling with the Green Hills Tracer Module” on page 89.</td>
</tr>
</tbody>
</table>
cycles
Prints the current values of all RDI cycle counters (if the RDI target supports this). For example:

```bash
>cycles
Instructions: 0x000044eec
S_Cycles: 0x00005d670
N_Cycles: 0x000030cbb
I_Cycles: 0x000011fd3
C_Cycles: 0x00000000
Total: 0x0000a02fe
```

rdiprop [group]... [prop_name[=value]]
Displays or sets properties exported by the RDI library. RDI properties are used to provide additional information and controls to the user.

If no arguments are given, the `rdiprop` command lists all top-level properties and property groups.

If a `group` argument is specified, the `rdiprop` command lists all properties and sub-groups in that group.

If a property name is given as an argument without a value, the current value of that property is printed.

If a property name is given and a value is also specified, the property is assigned that value.

reg [regname[=val]] [regname2[=val2]]
Reads or writes to the specified registers.

If no arguments are specified, this command lists the names of all registers and their values.

If register names are specified, only the specified registers are listed.

If a value is specified for a register, the register is set to that value. Values should be specified as a hexadecimal with no leading 0x. Only registers available in the current mode can be used. For example:

```bash
reg r0
reg r0=0001a000
reg r0=1
reg r0
reg r0=00000001
```
regsdm [[[module:bank:register [=value]]]

Displays or sets RDI self-describing module registers. Self-describing modules allow the RDI library to inform the MULTI Debugger about peripherals on the target and their registers.

If no arguments are given, the regsdm command prints a list of all of the self-describing modules, all of their register banks, and the names and current values of each register.

If a register is specified without a value, all of the available information about that register, including its current value, is printed.

If a register is specified and a value is also provided, the register will be set to that value.

rst

Resets the target.

Troubleshooting Your Remote Debug Interface (rdiserv) Connection

Two common problems and troubleshooting suggestions are described below.

rdiserv Cannot Connect to the Target

If rdiserv cannot connect to the target, consider these troubleshooting questions:

• Does the target board have power?
• Are all connections secure?
• (For Multi-ICE connections only) Did you copy multi-ice.dll from the Multi-ICE server installation to the Green Hills tools directory?
• (For Multi-ICE connections only) Is the Multi-ICE server running on the Windows PC that is connected to the Multi-ICE?
• (For Multi-ICE connections only) Does the Multi-ICE have power?
• (For ADP connections only) Are all communications settings correct?
Unable to set syscall Breakpoint

This message occurs after program download and almost always indicates that the target board’s memory was not set up correctly or that the linker directives file for the downloaded program is not correct. To troubleshoot this problem:

- Use the `m` command to ensure that your target memory has been correctly set up on the board. For example:
  
  1. Read a memory locations and note its value. For example:
    
    RDI> m a0020000
    
    f9f7fbbb
  
  2. Write a different value to the location. For example:
    
    RDI> m a0020000=deadbeef
  
  3. Read the memory location again and see if it changes to the new value.
     For example:
    
    RDI> m a0020000
    
    deadbeef

- Check your linker directives files to make sure that all program contents are linked to run in valid memory. You can confirm that all program contents are in valid memory with a link map of your executable.

RDI Error Codes

All RDI API functions return a status code. When `rdiserv` encounters an unexpected RDI error, it will display the error code and any information it has about the error.

The RDI API reserves the first 256 error codes for general target-independent errors. Error codes above 256 are target-specific. The RDI API includes a call that, when implemented by the target, converts an RDI error code to a text string describing the error condition. When `rdiserv` encounters an unexpected error, it attempts to use this API call and then prints the descriptive string. If the error code is defined in the ARM RDI header files, `rdiserv` will also print the name of the error code. Unfortunately, most RDI targets do not return descriptive strings for all the error codes they return.
Using the Green Hills Tracer Module for the ARMuator

The standard ARMuator includes an ARMuator module called the Tracer module. In the standard ARMuator, this module can be configured to output instruction and memory trace information to the console, to a file in text format, or to a binary file. Source code for the ARM Tracer module is provided with the ARMuator extension kit. Green Hills has created a custom version of the Tracer module that provides instruction and memory trace information to \texttt{rdiserv} for use with the MULTI trace tools and MULTI profiling. For more information about using the MULTI trace tools, see Chapter 19, “Collecting and Using Trace Data” in the \textit{MULTI: Debugging} book.

Profiling with the Green Hills Tracer Module

If the ARMuator you are using includes the Green Hills Tracer module, \texttt{rdiserv} will detect it and automatically take advantage of it when profiling. Refer to Chapter 15, “Using the Profiler” in the \textit{MULTI: Debugging} book for information about profiling your application with MULTI.

When the Green Hills Tracer module is available, profiling with the ARMuator will be very accurate and precise because every instruction execution and its approximate cycle count will be taken into account. The ARMuator is not strictly cycle accurate, so cycle counts are not always correct. However, the ARM documentation states that the ARMuator models for simple uncached ARM cores are cycle accurate.

Without the Tracer module, \texttt{rdiserv} falls back on its standard profiling mechanism.

Configuring the Tracer Module

The Tracer module is configured using the ARMuator configuration files. Previous versions of the ARMuator were configured with a single file named \texttt{armul.cnf}. The ARMuator is now configured using many more specialized configuration files with \texttt{.ami} and \texttt{.dsc} extensions. The file \texttt{peripherals.ami} should contain a section with all of the Tracer configuration options.

The Green Hills Tracer module is based on the standard ARM Tracer module. For information about configuration options for the ARM Tracer module, refer to the relevant ARM documentation.
In addition to the ARM Tracer module options, the following options have been added by Green Hills:

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GHSTrace</td>
<td>Boolean</td>
<td>If GHSTrace is true, the Tracer module will collect data and send it to rdiserv. If GHSTrace is false, the Tracer module will revert to the default ARM behavior.</td>
</tr>
<tr>
<td>GHSTraceIgnoreFetch</td>
<td>Boolean</td>
<td>If GHSTraceIgnoreFetch is true, memory accesses generated by instruction fetches will not be traced. If GHSTraceIgnoreFetch is false, every instruction fetch will show up as a separate row in the Trace List. Usually this does not provide much information and clutters the display.</td>
</tr>
</tbody>
</table>

A default set of ARMulator configuration files is installed by the Green Hills installation process. These files contain default Tracer configuration options that should work well for most users.

**Integrating the Tracer Module with Your Custom ARMulator**

The ARMulator has been reorganized so that the rebuild kit has been replaced with an extension kit. Rather than customizing the ARMulator by rebuilding it, the extension kit creates custom ARMulator module shared libraries. The ARMulator’s configuration files instruct it to load these custom libraries.

The Green Hills Tracer module is distributed as the pre-built ARMulator extension kit module Tracer.dll (Windows), Tracer.so (Solaris and Linux), or Tracer.sl (HP-UX) and comes preconfigured to allow trace data collection.

The ARMulator extension kit included on the Green Hills CD includes the source for the Green Hills Tracer module.
Chapter 6

INDRT (rtserv) Connections

This Chapter Contains:

- Connecting to Your Target with INDRT
INDRT is a software interface provided by Green Hills Software that facilitates run-mode debugging of real-time operating systems. INDRT and rtserv, the debug server that supports INDRT connections, are installed automatically when you install a distribution of MULTI that supports INDRT connections.

This chapter supplements the general target connection information contained in Chapter 2, “Setting Up Your Target Hardware” of this book and Chapter 3, “Connecting to Your Target” in the MULTI: Debugging book with specific information for INDRT connections.

**Connecting to Your Target with INDRT**

To help you connect to your target quickly and easily, MULTI allows you to create and save Connection Methods that correspond to your particular host and target systems and your desired debugging options.

For general instructions on how to create and use Connection Methods, see Chapter 3, “Connecting to Your Target” in the MULTI: Debugging book. The information in the following sections supplements the instructions provided there with information that is specific to INDRT connections.

The gproxy utility allows you to make INDRT connections to a target that is not directly accessible from your debugging host. See Appendix B, “Using the gproxy Network Proxy Utility”.

**Creating a Standard INDRT (rtserv) Connection Method**

When creating a new Standard INDRT Connection Method, select INDRT (rtserv) as the connection type in the Create New Connection Method dialog box.
For detailed instructions on creating new Standard Connection Methods, see “Creating a Standard Connection Method” in Chapter 3, “Connecting to Your Target” in the *MULTI: Debugging* book.

**Using the INDRT (rtserv) Connection Editor**

In addition to the generic fields that appear on all Connection Editors for Standard Connection Methods, the **INDRT (rtserv) Connection Editor** includes **Connection**, **Advanced**, and **Debug** tabs that provide settings and options specific to your target and host operating systems.

When the **Connection Editor** is first displayed after you create a new Connection Method, the settings and options on the tabs are set to default values. Settings and options that are not available on your host operating system...
may appear dimmed. Some of the fields may require user input before the Connection Method functions.

All of the fields on the **Connection**, **Advanced**, and **Debug** tabs of the **INDRT (rtserv) Connection Editor** are described in detail below. (See “The Connection Editor” in Chapter 3, “Connecting to Your Target” in the *MULTI: Debugging* book for a description of the other fields and options, which appear on all **Connection Editors**.)

### INDRT (rtserv) Connection Settings

<table>
<thead>
<tr>
<th>Connection</th>
<th>Advanced</th>
<th>Debug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet / IP Connection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Target Name(s) or Address(es) in the form: string1, string2, ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TFTP Load Directory</td>
<td>Choose...</td>
<td></td>
</tr>
<tr>
<td>Serial Connection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Serial Port:</td>
<td>default</td>
<td>Baud Rate:</td>
</tr>
</tbody>
</table>

---

94 MULTI: Configuring Connections for ARM Targets
Connecting to Your Target with INDRT

**Ethernet/IP Connection**

Sets your desired connection type as Ethernet/IP. This radio button is mutually exclusive with the **Serial Connection** button. If you select an Ethernet/IP connection (this is the default), the following fields will also be available:

- **Target Name(s) or Address(es)** — Specifies the hostname(s) or IP address(es) of your target. You must specify a hostname or IP address to create a valid Ethernet/IP connection.

  Currently, connecting to multiple targets is only supported by the Green Hills INTEGRITY RTOS. To specify multiple targets, enter the target names or addresses separated by commas. See the *INTEGRITY Development Guide* for more information.

- **TFTP Load Directory** — Specifies a load directory for dynamic downloading via TFTP. This is the directory to which *rtserv* will copy a file before requesting a download. Enter the name of your desired TFTP load directory or click **Choose** to browse to it. The specified directory must be accessible by the TFTP server.

  Currently, dynamic downloading via TFTP is only supported by the Green Hills INTEGRITY RTOS. See the *INTEGRITY Development Guide* for more information.

**Serial Connection**

Specifies that a serial connection to the target should be used. This radio button is mutually exclusive with the **Ethernet/IP Connection** button. If you select a serial connection, the following option fields will also be available:

- **Serial Port** — Specifies which host serial port to use for your serial INDRT connection. The default serial port (used if you do not specify a port) for each supported host operating system is listed below:
  - (Windows) **COM1**
  - (Solaris) **/dev/ttya**
  - (HP-UX) **/dev/tty00**
  - (Linux) **/dev/cua0**

- **Baud Rate** — Specifies the serial port communication speed. The default baud rate is 9600.

**Note** If you need to connect to your target using a non-standard UDP port number (i.e., not 2220), you must use a Custom Connection Method. See “Custom INDRT (rtser) Ethernet Connections” on page 97 for instructions.
INDRT (rtserv) Advanced Settings

<table>
<thead>
<tr>
<th>Connection</th>
<th>Advanced</th>
<th>Debug</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BSD Exclusive Serial Port Access</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Always Use TFTP for Load</td>
<td></td>
</tr>
</tbody>
</table>

Warning Use this tab carefully, since changing the advanced options from their default settings can cause problems with your connection.

BSD Exclusive Serial Port Access
Enables exclusive serial port access.
By default, the serial port is opened in exclusive, or locked, mode. If you are connecting through a terminal server, clear this box to disable exclusive serial port access.

Always Use TFTP for Load
Forces rtserv to use TFTP for downloads, even when the system defaults to a different download method (for example, host I/O).

INDRT (rtserv) Debug Settings

<table>
<thead>
<tr>
<th>Connection</th>
<th>Advanced</th>
<th>Debug</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Log Host-Target Debug Output</td>
<td></td>
</tr>
<tr>
<td></td>
<td>StdErr/StdOut</td>
<td></td>
</tr>
<tr>
<td></td>
<td>File</td>
<td></td>
</tr>
</tbody>
</table>

rtserv.log
Choose...

Other Options:

Other Options:
Warning Do not change the settings on the debug tab unless you are instructed to do so by Green Hills Technical Support.

<table>
<thead>
<tr>
<th>Log Host-Target Debug Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enables logging of all communications between rtserv and your target. Logging is disabled by default.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>StdErr/StdOut File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allows you to specify the destination for host-target debug output. These fields will be dimmed unless Log Host-Target Debug Output is selected.</td>
</tr>
<tr>
<td>If you choose StdErr/StdOut, host-target debug output will be directed to the console.</td>
</tr>
<tr>
<td>If you select File, host-target debug output will be directed to the file you specify in the text field. You may enter a filename directly into this text field or click Choose to browse the file system. A default filename will be suggested.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allows you to add other, optional arguments directly to the command line. You should only use this field if directed to do so by Green Hills Technical Support.</td>
</tr>
</tbody>
</table>

Using Custom INDRT (rtserv) Connection Methods

You can create a Custom INDRT (rtserv) Connection Method by manually entering commands and options into the Connection Chooser instead of using the graphical Connection Editor.

The appropriate commands for Custom INDRT Connection Methods for Ethernet and serial connections are described in the sections below.

Custom INDRT (rtserv) Ethernet Connections

To establish an INDRT connection over Ethernet using a Custom Connection Method, type the command given below, with the appropriate parameters and options, into the Start a Custom Connection dialog box and click Connect.

```
rtserv [-d7 [-log filename] ] [-loaddir tftp_dir ] [-forcetftp ] -port protocol@hostname [portnumber]
```
where:

- **-d7 [-log filename]** is optional and enables rtserv protocol debugging. If used without the -log filename option, -d7 causes communication to be logged to stderr/stdout, which is usually inhibited on Windows hosts. To direct the output to a file, use the additional -log filename option. You should not need to use the -d7 option unless directed to do so by Green Hills Technical Support.

- **-loaddir tftp_dir** is optional and specifies the TFTP load directory for dynamic downloading.

- **-forcetftp** is optional and causes rtserv to always use TFTP to load the program.

- **-port protocol@hostname** is required and specifies the target protocol and hostname for your Ethernet connection. The only protocol currently supported is udp.

- **portnumber** is optional and indicates the UDP port number. The port number defaults to 2220 if not specified.

**Note** You can also enter the above connection command from the Debugger’s command pane, where it must be preceded by the connect command.

**Note** Currently, dynamic downloading via TFTP and connecting to multiple targets are only supported by the Green Hills INTEGRITY RTOS. See the INTEGRITY Development Guide for information about using these features with rtserv.

For example connection commands for INDRT Ethernet connections, see “Example Custom INDRT (rtserv) Connection Methods” on page 100.

See “Custom Connection Methods” in Chapter 3, “Connecting to Your Target” in the MULTI: Debugging book for more information about Custom Connections.
Custom INDRT (rtserv) Serial Connections

To establish a serial INDRT connection using a Custom Connection Method, type the command given below, with the appropriate parameters and options, into the Start a Custom Connection dialog box and click Connect.

\texttt{rtserv [-d7 [-log filename] ] [-serialdev devicepath [-baud baud_rate ]] [-noexcl ] [-forcetftp ]}

where:

- \texttt{-d7 [-log filename]} is optional and enables \texttt{rtserv} protocol debugging. If used without the \texttt{-log filename} option, \texttt{-d7} causes communication to be logged to \texttt{stderr/stdout}, which is usually inhibited on Windows hosts. To direct the output to a file, use the additional \texttt{-log filename} option. You should not need to use the \texttt{-d7} option unless directed to do so by Green Hills Technical Support.

- \texttt{-serialdev devicepath} is optional and specifies the serial port device to use when connecting to the target. If no serial device is specified, the default serial port device is used, as listed below.
  - (Windows) COM1
  - (Solaris) /dev/ttya
  - (HP-UX) /dev/tty00
  - (Linux) /dev/cua0

- \texttt{-baud baud_rate} is optional and sets the serial port communication speed. The default baud rate is 9600. If a baud rate is specified, a \texttt{devicepath} must also be specified.

- \texttt{-noexcl} is optional and disables exclusive serial port access. By default, the serial port is opened in exclusive, or locked, mode. Specify the \texttt{-noexcl} option if you are connecting through a terminal server.

- \texttt{-forcetftp} is optional and causes \texttt{rtserv} to always use TFTP to load the program.

\textbf{Note} You can also enter the above connection command from the Debugger’s command pane, where it must be preceded by the \texttt{connect} command.
**Note** When connecting through a terminal server, pass the `-noexcl` option to `rtserv`. This disables the default locked serial port mode. Failure to pass this option may result in `rtserv` aborting with the message:

```
server: device devicepath is busy
```

For example connection commands for serial INDRT connections, see the next section.

See “Custom Connection Methods” in Chapter 3, “Connecting to Your Target” in the *MULTI: Debugging* book for more information about Custom Connections.

### Example Custom INDRT (rtserv) Connection Methods

The following are examples of commands that can be entered into the **Start a Custom Connection** dialog box. (These example commands can also be entered, preceded by the `connect` command, from the MULTI command pane.)

#### Example 1. Ethernet Connection

To connect over Ethernet to UDP port 2229 on a host named `mike` with no protocol debugging messages, enter the following in the **Start a Custom Connection** field of the **Connection Chooser**:

```
rtserv -port udp@mike 2229
```

#### Example 2. Serial Connection

To connect using serial port B at 19200 baud with protocol debugging messages, enter the following in the **Start a Custom Connection** field of the **Connection Chooser**:

```
rtserv -d7 -serialdev /dev/ttyb -baud 19200
```
This Chapter Contains:

- Macraigor OCD Interfaces
- Supported Macraigor OCD Target Systems
- Installing Your Macraigor OCD Interface
- Troubleshooting Your Macraigor OCD Hardware Setup
- Connecting to Your Target with a Macraigor OCD Interface
- Running MacDemon
- Additional Commands for Macraigor OCD (ocdserv) Connections
- Common OCD Connection Errors
This chapter supplements the general target connection information contained in Chapter 2, “Setting Up Your Target Hardware” of this book and Chapter 3, “Connecting to Your Target” in the MULTI: Debugging book with specific information for Macraigor on-chip debugging (OCD) connections.

Macraigor OCD Interfaces

On-chip debugging (OCD) refers to a group of cost-efficient development interfaces built into many embedded processors. OCD allows software developers to read and write registers, read and write memory, single step through code, and set breakpoints—all without using a software monitor or operating system. This allows real application code to be downloaded and debugged on a target board without a developer having to write a monitor program for the target board.

Many different hardware interfaces are available for connecting a host computer to an OCD target. These interfaces vary in implementation type, cost, download speed, and interaction speed. Some interfaces exist only for certain hosts and certain targets. MULTI supports Macraigor OCD interfaces through the debug server ocdserv. This chapter describes how to use ocdserv with the following Macraigor OCD interfaces: OCDemon Wiggler and OCDemon Raven.

OCDemon Wiggler

The OCDemon Wiggler, the lowest-cost and slowest of these interfaces, connects a target board to a PC parallel port. The PC directly toggles (wiggles) the control lines on the OCD interface to communicate with the target. You can use MULTI with the OCDemon Wiggler to connect to an ARM7 or ARM7T target, as shown in the following diagram.

The OCDemon Wiggler connects directly only to Windows hosts. It is possible, however, to use an Ethernet connection, a Windows PC, and the MacDemon software included with ocdserv to debug targets from a UNIX host with the
OCDemon Wiggler. For more information, see “On-Chip Debugging Using MacDemon” on page 103.

**OCDemon Raven**

The OCDemon Raven, a more expensive interface that is up to thirty times faster than the OCDemon Wiggler, connects a target board to a PC parallel port. The Raven interface contains specialized logic to control the OCD interface without intervention from the PC. You can use MULTI with the OCDemon Raven to connect to an ARM7, ARM7T, or XScale target, as shown in the following diagram.

The OCDemon Raven connects directly only to Windows hosts. It is possible, however, to use an Ethernet connection, a Windows PC, and the MacDemon software included with `ocdserv` to debug targets from a UNIX host with the OCDemon Raven. For more information, see “On-Chip Debugging Using MacDemon” on page 103.

**On-Chip Debugging Using MacDemon**

MacDemon is a program included with `ocdserv` that provides additional OCD versatility by allowing you to control an OCDemon Wiggler or OCDemon Raven over an Ethernet connection, which can be helpful if your host is remote from your target.

MacDemon also allows you to use an Ethernet connection and a Windows machine to control an OCDemon Wiggler or OCDemon Raven from a UNIX host. An OCDemon Wiggler or OCDemon Raven interface can only connect directly to a Windows machine. You can, however, use a Windows machine that is connected to a Wiggler or Raven and is running MacDemon to communicate over Ethernet with a UNIX machine, thus allowing the UNIX box to control the OCD device and debug the target, as shown in the diagram below.
Supported Macraigor OCD Target Systems

MULTI supports on-chip debugging for the CPU-interface pairings listed in the table below. (See “Macraigor OCD Interfaces” on page 102 for a description of each OCD interface.)

<table>
<thead>
<tr>
<th>CPU</th>
<th>Macraigor OCD interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM7</td>
<td>OCDemon Wiggler or OCDemon Raven</td>
</tr>
<tr>
<td>ARM7T</td>
<td>OCDemon Wiggler or OCDemon Raven</td>
</tr>
<tr>
<td>NetARM</td>
<td>OCDemon Wiggler or OCDemon Raven</td>
</tr>
<tr>
<td>XScale</td>
<td>OCDemon Raven</td>
</tr>
</tbody>
</table>

Installing Your Macraigor OCD Interface

Your hardware installation procedure will vary according to your OCD interface. The following sections describe how to set up the hardware for parallel port (OCDemon Wiggler and OCDemon Raven) interfaces.

The `ocdserv` debug server that supports Macraigor OCD connections is installed automatically when you install MULTI.
OCDemon Wiggler and OCDemon Raven Installation

Your OCDemon Wiggler or OCDemon Raven includes a ribbon cable with the appropriate connector for your target system hardware, as described in the following table.

<table>
<thead>
<tr>
<th>Target system</th>
<th>Cable and connector</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM7</td>
<td>Ribbon cable with a 14-pin or 20-pin Berg connector with pin 1 marked by a red stripe.</td>
</tr>
<tr>
<td>ARM7T</td>
<td>Ribbon cable with a 14-pin or 20-pin Berg connector with pin 1 marked by a red stripe.</td>
</tr>
<tr>
<td>NetSilicon ARM</td>
<td>Ribbon cable with a 14-pin or 20-pin Berg connector with pin 1 marked by a red stripe.</td>
</tr>
<tr>
<td>XScale</td>
<td>Ribbon cable with a 20-pin Berg connector with pin 1 marked by a red stripe.</td>
</tr>
</tbody>
</table>

To install an OCDemon Wiggler or an OCDemon Raven, connect the OCD cable to your target board using the ribbon cable’s red stripe to align with pin 1 on the target board. We recommend that you connect the OCD cable directly to the PC’s parallel port without using a parallel extension cable, as electromagnetic interference can enter the parallel extension cable and cause unreliable operation of the OCD interface.

⚠️ **Warning** We strongly recommend making all connections before applying power to the target board.

For Raven connections, your parallel port must be set to EPP mode. EPP mode is usually set in the PC’s BIOS setup.

You are now ready to configure your target as described in “Using the New Project Wizard to Configure and Test Your Target” on page 16.

Troubleshooting Your Macraigor OCD Hardware Setup

If you experience problems with your hardware setup, the following questions may help you locate the problem.

- Is the OCDemon power supply connected to the wall outlet and powered? This is necessary for any OCDemon device with an external power supply.
- Is the target connected to the wall outlet and powered?
- Are all connections secure?
7. Macraigor On-Chip Debugging (ocdserv) Connections

- Does the OCDemon device work when it is directly connected to the parallel port without a parallel port extension cable? (Some PC parallel ports may not be able to drive the supplied 6-foot extension cable, so the OCDemon device needs to be connected directly to the parallel port. The nuts on the device may need to be removed before connecting it to some ports. Use a small pair of pliers to unscrew the nuts.)

- Is the OCDemon device’s OCD cable connected to the correct pins on the target board in the correct orientation, with the red stripe on the OCD cable indicating pin 1?

- (Windows hosts) Is the parallel port BIOS setting set to EPP? Setting the parallel port to non-EPP modes, such as ECP or Auto, may cause problems on certain systems.

- (Windows hosts) Is your Windows parallel port driver correctly installed? Some software may disable or change the parallel port driver. We have tested the OCDemon devices only on the default parallel port driver that ships with Windows.

- (When testing registers) Has the `debug 8` command been executed to turn off register caching?

If you experience problems with making a target connection or downloading or debugging, see “Common OCD Connection Errors” on page 125 also.

### Connecting to Your Target with a Macraigor OCD Interface

After you have installed and configured your system, you are ready to connect to your target and begin debugging. To help you connect to your target quickly and easily, MULTI allows you to create and save Connection Methods that correspond to your particular host and target systems and your desired debugging options.

For general instructions on how to create and use Connection Methods, see Chapter 3, “Connecting to Your Target” in the *MULTI: Debugging* book. The information in the following sections supplements the instructions provided there with information that is specific to Macraigor OCD connections.
Connecting to Your Target with a Macraigor OCD Interface

Creating a Standard Macraigor OCD (ocdserv) Connection Method

When creating a new Standard Macraigor OCD Connection Method, select **Macraigor OCD (ocdserv) for ARM** as the connection type in the Create New Connection Method dialog box.

For detailed information about creating new Standard Connection Methods, see “Creating a Standard Connection Method” in Chapter 3, “Connecting to Your Target” in the *MULTI: Debugging* book.

Using the Macraigor OCD (ocdserv) Connection Editor

In addition to the generic fields that appear on all Connection Editors for Standard Connection Methods, the **Macraigor OCD (ocdserv) Connection Editor** includes **Connection**, **Advanced**, and **Debug** tabs that provide settings and options specific to your target and host operating systems.
When the **Connection Editor** is first displayed after you create a new Connection Method, the settings and options on the tabs are set to default values. Settings and options that are not available on your host operating system may appear dimmed. Some of the fields may require user input before the Connection Method functions.

All of the fields on the **Connection**, **Advanced**, and **Debug** tabs of the **Macraigor OCD (ocdserv) Connection Editor** are described in detail below. (See “The Connection Editor” in Chapter 3, “Connecting to Your Target” in the **MULTI: Debugging** book for a description of the other fields and options, which appear on all **Connection Editors**.)
Connecting to Your Target with a Macraigor OCD Interface

Macraigor OCD (ocdserv) Connection Settings

<table>
<thead>
<tr>
<th>Connection</th>
<th>Advanced</th>
<th>Debug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethernet OCDemon</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MacDemon</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host Name:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Port:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Local</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wiggler</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Raven</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallel Port:</td>
<td>LPT1</td>
<td></td>
</tr>
</tbody>
</table>

Processor: ARM7T  Endian: <default>

Ethernet OCDemon

MacDemon

Wiggler

Raven

Specifies the type of OCD connection. Select one of these radio buttons according to your OCD interface.

Select **MacDemon** if you want to use an Ethernet connection and a Windows machine to control an OCDemon Raven or OCDemon Wiggler from another, remote Windows host or from a UNIX machine.

Ethernet OCDemon is not supported for ARM processors.

Host Name

Specifies the hostname or IP address of your remote Windows machine (for MacDemon connections). This field requires user input if you have selected a MacDemon connection.

Port

Specifies which TCP port to use when connecting to MacDemon. This field only applies if you have selected a MacDemon connection.

When using MacDemon, the TCP port defaults to 10000. It is not necessary to set this field unless you are running MacDemon on a non-standard port.
7. Macraigor On-Chip Debugging (ocdserv) Connections

**Parallel Port**
Specifies the parallel port to use when connecting to the target. The default is LPT1. This field only applies to Wiggler and Raven connections.

**Processor**
Specifies the target processor type. This field defaults to ARM7T.

**Endian**
Specifies the target byte order. By default, ocdserv will use the default endian setting for your processor type. Use this field if you want to override the default setting.

---

**Macraigor OCD (ocdserv) Advanced Settings**

![Advanced Settings Tab]

- **DLL Name**: Specifies an alternate DLL to use instead of the default, wigglers.dll. To use an alternate DLL, enter a the filename in the text box or click **Choose** to browse to it.

---

**Warning** Use this tab carefully, since changing the advanced options from their default settings can cause problems with your connection.

---

**DLL Name**
Specifies an alternate DLL to use instead of the default, wigglers.dll. To use an alternate DLL, enter a the filename in the text box or click **Choose** to browse to it.
Apply Flash/ROM Debugging Defaults
Sets options that are appropriate for debugging code that is burned into ROM on the target. Specifically, clicking this button disables downloading of any program sections and disables system calls by clearing the Text, Data, BSS, and System Calls check boxes.

System Calls
Enables host-based system calls.
If this box is selected, system calls are enabled and are implemented with a single software breakpoint. This box is checked by default.
If you are running from ROM and software breakpoints cannot be used, it may be desirable to disable host-based system calls by clearing this box.

Attach To Running Process
Sets passive attachment mode. When this box is checked, the debug server can attach to a running target without disturbing the target. This box is cleared by default.

Hardware Breakpoints
Enables hardware breakpoints.
This box is checked and hardware breakpoints are on by default. However, some RTOS kernels may use the hardware breakpoints. Clearing this check box prevents conflicts between ocdserv and such kernels.

Software Breakpoints
Enables software breakpoints.
This box is checked and software breakpoints are on by default. However, some RTOS kernels may use the software breakpoint trap. Clearing this check box prevents conflicts between ocdserv and such kernels.

Breakpoint Trap
Enables traps for breakpoints. This box is selected by default, since the breakpoint trap option must always be set for ARM targets.

Auto Hardware Breakpoint Resume
Enables automatic hardware breakpoint continuation.
By default, this box is cleared and MULTI continues hardware breakpoints for ocdserv. Setting this option, which is intended for use with older versions of MULTI, allows ocdserv to continue off its own hardware breakpoints.
Breakpoint Step
Determines the single instruction step method.
If this box is checked, single instruction steps will be implemented by placing a software breakpoint on the next instruction to be executed. If it is not checked, hardware single step will be used.
This box is cleared by default.

Load Sections
Controls which sections of your program will be downloaded to the target, as follows:

- **Text** — If this box is checked, the `.text` (code) sections of your program will be downloaded to the target. This box is checked by default.
- **Data** — If this box is checked, the `.data` (initialized data) sections of your program will be downloaded to the target. This box is checked by default.
- **BSS** — If this box is checked, the `.bss` (uninitialized data) sections of your program will be cleared. This box is not checked by default.

Pass Through
Specifies the string to pass through to the DLL `passthrough` command. Some OCD target interfaces accept strings for further configuration. This option passes such configuration strings to the target interface.

Macraigor OCD (ocdserv) Debug Settings

<table>
<thead>
<tr>
<th>Connection</th>
<th>Advanced</th>
<th>Debug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log Macraigor DLL Communications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cable Version Check</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For Out-Of-Sync Connections:
- Perform Frequent Checks To Verify Connection Status
- Attempt To Fix Connection Failures

Other Options: 

**Warning** Do not change the settings on the debug tab unless you are instructed to do so by Green Hills Technical Support.

<table>
<thead>
<tr>
<th>Log Macraigor DLL Communications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enables <code>ocdserv</code> logging. Logging is disabled by default.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cable Version Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enables OCD cable version checking. This check box is selected by default.</td>
</tr>
<tr>
<td>Some Macraigor DLLs may report incorrect cable version numbers and cause <code>ocdserv</code> to disable some of its features. Using this check box to disable version checking will work around this problem.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Perform Frequent Checks To Verify Connection Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enables frequent polling of your OCDemon device.</td>
</tr>
<tr>
<td>This check box is cleared by default. If you select this option the follow option also becomes available:</td>
</tr>
<tr>
<td>• <strong>Attempt to Fix Connection Failures</strong> — Causes <code>ocdserv</code> to automatically retry certain OCDemon actions that fail. This option is only available if <strong>Perform Frequent Checks</strong> has been selected, and is off by default.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allows you to add other, optional arguments directly to the command line. You should only use this field if directed to do so by Green Hills Technical Support.</td>
</tr>
</tbody>
</table>

**Using Custom Macraigor OCD (ocdserv) Connection Methods**

To connect to your Macraigor OCD device with a Custom Connection Method, type the command given below, with the appropriate parameters and options, into the **Start a Custom Connection** dialog box and click **Connect**.

```
[mode=conn_mode] [setup=filename.mbs] ocdserv connection -cpu cpu_type [options]...
```

where:

- **mode=conn_mode** is optional and specifies the connection mode. Appropriate values for `conn_mode` are **download**, **attach**, and **bareboard**. If the mode is not specified, MULTI will connect in download mode. See Chapter 18, “Debugging in Download, Attach, and Board Setup Modes” in the **MULTI: Debugging** book for more information about connection modes.

- **setup=filename.mbs** is optional and specifies the target setup script. This argument is optional because not all targets require setup scripts.
Note This option can only be used to specify an .mbs setup script. If you are using an older .dbs script, you must use the -setup option, which must come after the ocdserv command. See “Configuring Your Target Hardware” on page 16 for more information about setup scripts.

- **connection** is required and specifies the connection parameters for your particular OCD interface. Appropriate values for connection are listed in “Connection Parameters for Custom Macraigor OCD Connection Methods” on page 115.

- **-cpu cpu_type** is required and specifies the type of CPU to be debugged. Valid arguments for cpu_type are listed in “CPU Types for Custom Macraigor OCD Connection Methods” on page 115.

- **options** can be any non-conflicting combination of the options listed in “Other Options for Custom Macraigor OCD (ocdserv) Connection Methods” on page 116.

Note You can also enter the above connection command from the Debugger’s command pane, where it must be preceded by the connect command.

See “Custom Connection Methods” in Chapter 3, “Connecting to Your Target” in the MULTI: Debugging book for more information about Custom Connections.

For example Macraigor OCD connection commands, see “Example Custom Macraigor OCD (ocdserv) Connection Methods” on page 118.
### Connection Parameters for Custom Macraigor OCD Connection Methods

<table>
<thead>
<tr>
<th>Value for connection (case-insensitive)</th>
<th>Device</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>-port lpt1</td>
<td>OCDemon Wiggler</td>
<td>Specifies the parallel port to which the OCDemon Wiggler is connected.</td>
</tr>
<tr>
<td>-port lpt2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-port lpt3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-port rlpt1</td>
<td>OCDemon Raven</td>
<td>Specifies the parallel port to which the OCDemon Raven is connected.</td>
</tr>
<tr>
<td>-port rlpt2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-port rlpt3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-macd hostname [[:port]]</td>
<td>MacDemon</td>
<td>Specifies the host and port of the Windows PC that is running MacDemon.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* hostname is either the name of the MacDemon Windows host PC or its IP address in dotted-decimal form.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* :port is an optional argument that specifies the port on the MacDemon host to which ocdserv connects. This setting is usually not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>specified and defaults to 10000.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MacDemon must be running before ocdserv can connect to it. See “Running Mac Demon” on page 119 for more information about</td>
</tr>
<tr>
<td></td>
<td></td>
<td>starting MacDemon.</td>
</tr>
</tbody>
</table>

### CPU Types for Custom Macraigor OCD Connection Methods

<table>
<thead>
<tr>
<th>Value for cpu_type (case-insensitive)</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>arm7</td>
<td>All ARM7 processors *</td>
</tr>
<tr>
<td>arm7t</td>
<td>All ARM7 processors with Thumb *</td>
</tr>
<tr>
<td>netarm</td>
<td>All NetSilicon ARM cores</td>
</tr>
<tr>
<td>xscale</td>
<td>XScale processors</td>
</tr>
</tbody>
</table>

* ARM7 cores without Thumb do not have 16-bit memory access capabilities, so 16-bit memory accesses are simulated with two 8-bit memory accesses in ARM7 mode. This limitation may slow cores that do not have 16-bit memory.
access capabilities or may affect access to 16-bit volatile memory regions. Use ARM7T mode if your ARM core is capable of 16-bit memory accesses.

Other Options for Custom Macraigor OCD (ocdserv) Connection Methods

In addition to the specific options described in the custom connection section above, you can also use any non-conflicting combination of the options listed below when creating or using a custom Macraigor OCD (ocdserv) Connection Method.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-attach</td>
<td>Sets passive attachment mode. This option allows ocdserv to attach to a running target without disturbing the target.</td>
</tr>
<tr>
<td>-autobp</td>
<td>Enables automatic hardware breakpoint continuation. By default, MULTI continues hardware breakpoints for ocdserv. This option, intended for use with older versions of MULTI, allows ocdserv to continue off its own hardware breakpoints.</td>
</tr>
<tr>
<td>-big</td>
<td>Sets big endian target mode. This option is needed only on processors that can be either big or little endian. Processors that are big endian by default do not need this switch.</td>
</tr>
<tr>
<td>-bpsc</td>
<td>Sets the system call instruction for software breakpoints. This option is useful when debugging an application that uses the exceptions normally used for software breakpoints by ocdserv.</td>
</tr>
<tr>
<td>-bpstep</td>
<td>Sets software breakpoint single-stepping mode. When this option is set, ocdserv will simulate single stepping with software breakpoints. This option is useful when debugging an application that uses the hardware single-stepping facility. This option is set by default for ARM targets because ARM processors lack any form of hardware single step.</td>
</tr>
<tr>
<td>-bss</td>
<td>Sets downloading of .bss (uninitialized data) sections. By default, .bss downloading is disabled because the default Green Hills startup code clears .bss sections. This option allows you to clear .bss sections upon download.</td>
</tr>
<tr>
<td>-d n</td>
<td>Sets debugging level to n. Do not use this option unless instructed to do so by Green Hills Technical Support.</td>
</tr>
</tbody>
</table>
**-data**
Sets initialized data downloading. By default, initialized data (.data) sections are always downloaded, so this switch should never be used.

**-little**
Sets little endian target mode. This option is needed only on processors that can be either big or little endian. Processors that are little endian by default, such as ARM7, do not need this switch.

**-loadall**
Enables downloading of all program sections. By default, all sections are loaded except for uninitialized data (.bss) sections, which are cleared by the default Green Hills startup code. The **-loadall** option allows you to download all sections, including .bss.

**-log**
Enables ocdserv logging. Do not use this option unless instructed to do so by Green Hills Technical Support.

**-nobss**
Disables uninitialized data (.bss) clearing. This option is enabled by default and is documented for backward compatibility only.

**-nodata**
Disables downloading of initialized data (.data) sections.

**-nodlcheck**
Disables pre-download checks. ocdserv tests the beginning and ending of all sections being downloaded before the download, so that invalid memory regions can be found more quickly than when downloading the entire executable. However, some memory may be mapped to volatile memory regions that should not be probed; the **-nodlcheck** option allows the user to disable the checks.

**-nohwbp**
Disables hardware breakpoints. By default, hardware breakpoints are on. However, some RTOS kernels may use the hardware breakpoints. Setting this option prevents conflicts between ocdserv and such kernels.

**-noload**
Disables downloading of any of the program sections in an executable file.

This option is useful when debugging an executable that is already loaded on the target. For example, use this option when debugging an executable in ROM or flash memory. This option does not disable stack setup processes, such as setting the stack pointer, writing the stack to memory, or setting the system call breakpoint.
-nosyscalls
Disables system calls. By default, ocdserv places a breakpoint on the .syscall section to service system calls on the target. However, some applications may place the .syscall in read-only memory, while others may implement their own system call mechanism with .syscall. The -nosyscalls switch prevents conflicts between ocdserv and such applications.

-notext
Disables downloading of program sections containing executable code (.text sections).

-noversion
Disables OCD cable version checking. Some Macraigor DLLs may report incorrect cable version numbers and cause ocdserv to disable some of its features. Using this option to disable version checking will work around these DLLs.

-ocddll dllname
Sets dllname as the Macraigor interface DLL instead of the default, wigglers.dll. This option is helpful when using new, experimental DLLs that have not yet been integrated into the default .dll.

-passthru string
Specifies the string to pass through to the DLL passthrough command. Some OCD target interfaces accept strings for further configuration. This option passes such configuration strings to the target interface.

-rom
Disables setting of the system call breakpoint. This option should be used when debugging an executable in ROM or flash memory, so that ocdserv will not set a software breakpoint in a read-only program section to facilitate its system call mechanism.

-setup filename
Specifies a setup script file that is run by the ocdserv command interpreter before each download.

This option can only be used to specify .dbs setup scripts. See “Specifying Setup Scripts” on page 28 for information about specifying .mbs scripts.

Example Custom Macraigor OCD (ocdserv) Connection Methods

The following are examples of commands that can be entered into the Start a Custom Connection dialog box. (These example commands can also be entered, preceded by the connect command, from the MULTI command pane.)
Example 1.

In the following example, `ocdserv` connects through the LPT1 parallel port and an OCDemon Wiggler to an ARM7 target and a script file named `arm7_ocd.mbs` is run prior to every program download.

```
setup=arm7_ocd.mbs ocdserv -port lpt1 -cpu arm7
```

Example 2.

In the following example, `ocdserv` connects through the LPT1 parallel port and an OCDemon Raven to an XScale target.

```
ocdserv -port rlpt1 -cpu xscale
```

Example 3.

In the following example, `ocdserv` connects to MacDemon on a computer named `ut` that is connected through an OCDemon device to an ARM7.

```
ocdserv -macd ut -cpu arm7
```

Note MacDemon must be running before `ocdserv` can connect to it. For information about starting MacDemon, see “Running MacDemon” on page 119.

**Running MacDemon**

MacDemon must be running before `ocdserv` can connect to it. To run MacDemon, enter the following in a DOS command window or the Windows Run dialog box:

```
macdemon connection [options]...
```

where:

- `connection` is required and specifies the connection parameters for the MacDemon connection. Valid arguments for `connection` are listed in “MacDemon Connection Parameters” on page 120.
- `options` can be any non-conflicting combination of the arguments listed in “MacDemon Options” on page 120.
You may have to specify the full pathname of MacDemon so that Windows can find it.

### MacDemon Connection Parameters

<table>
<thead>
<tr>
<th>Values for connection (case-insensitive)</th>
<th>Device</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpt1</td>
<td>Wiggler</td>
<td>Specifies the parallel port to which the Wiggler is connected.</td>
</tr>
<tr>
<td>lpt2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lpt3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rlpt1</td>
<td>Raven</td>
<td>Specifies the parallel port to which the Raven is connected.</td>
</tr>
<tr>
<td>rlpt2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rlpt3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### MacDemon Options

- **-log [filename]**
  
  Enables Macraigor API logging. If *filename* is not specified, the output goes to standard out.

- **-port port**
  
  Specifies the port on the MacDemon host on which MacDemon listens. The default port is 10000 and should not be changed unless another program is already using port 10000. If a different port number is used, it must also be specified in the *macd* option to *ocdserv*.

### Example MacDemon Commands

The following are example commands that can be entered in a DOS command window or the Windows **Run** dialog box to start MacDemon.

**Example 1.**

To run MacDemon to serve a Wiggler on LPT2, enter:

```
macdemon lpt2
```
Additional Commands for Macraigor OCD (ocdserv) Connections

Example 2.

To run MacDemon to serve a Raven on LPT1, enter:

macdemon rlpt1

For examples of how to connect to MacDemon with ocdserv, see “Example Custom Macraigor OCD (ocdserv) Connection Methods” on page 118.

Additional Commands for Macraigor OCD (ocdserv) Connections

The following commands, in addition to the commands listed in “Generic Debug Server Commands” on page 138, are available to ocdserv, the Green Hills debug server that supports Macraigor OCD connections.

You can enter all of these commands directly into the ocdserv Target window. You can also enter these commands into the MULTI Debugger command pane using the target command. All of the commands for ocdserv are case-insensitive.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>api bytes</td>
<td>Passes bytes as Macraigor protocol bytes to the OCDemon device. The protocol checksum will be computed automatically and must be omitted from bytes. Do not use this command unless directed to do so by Green Hills Technical Support.</td>
</tr>
<tr>
<td>autobp [on</td>
<td>off]</td>
</tr>
<tr>
<td>blockfill byte</td>
<td>short</td>
</tr>
</tbody>
</table>
### bpstep [on | off]

Turns single stepping using breakpoints on or off.

- If **on** is specified: Makes `ocdserv` read and analyze the instruction it is about to single step and set a breakpoint at the instruction that will be reached after the execution of the current instruction. `ocdserv` also reads any necessary condition registers to predict where to set the breakpoint. Only one software breakpoint is set for `bpstep`. Turning `bpstep` on allows `ocdserv` to debug an application that uses the default single-stepping mechanism, like an RTOS.

- If **off** is specified: Makes `ocdserv` use the default single-stepping mechanism of the target CPU.

If no argument is specified: Displays the current setting of `bpstep`.

### breakdump

Lists all software breakpoints currently set.

### delay [n | slow | fast]

- If **n** is specified: Sets the current communications delay value to `n`. The delay value is a 16-bit number that divides the communication clock speed. Therefore, `n=1` sets the fastest communication speed.

- If **slow** is specified: `ocdserv` sets the communication delay to a value that increases the likelihood of successful communications between the target CPU and the OCDemon device when the CPU clock speed is not set.

- If **fast** is specified: `ocdserv` sets the communication delay so that the OCDemon device communicates with the target CPU at the highest possible speed. On some processors, the **fast** setting may not be usable until the clock speed of the target CPU is set appropriately.

If no argument is specified: Displays the current communications delay value.

### dlcheck [on | off]

- If **on** or **off** is specified: Turns download checking on or off.

- If no arguments are specified: Displays the current setting of `dlcheck`.

Download checking tests the beginning and ending addresses of every program section being downloaded to make sure the corresponding memory addresses can be written. Download checking is on by default and is useful because `ocdserv` does not have to download the entire program to discover that memory cannot be written. You may want to turn off download checking if you are downloading to volatile memory regions that should not be written more than once.
### haltnum [num]

If *num* is specified: Sets the number of times *ocdserv* will retry a failed halt before reporting an error.

If *num* is not specified: Displays the current setting of *haltum*. Some targets do not halt the first time *ocdserv* tries to halt them, but will halt after some number of attempted halts. The default *haltum* setting is 5. Almost all targets will halt after the first attempted halt.

### inticache

For XScale only.

Moves the ROM/flash interrupt table into the mini instruction cache. This is useful for applications that use their own interrupts.

### listsec

Lists the program sections that have been loaded onto the target.

### log [on | off | message]

Controls OCDemon logging.

If *on* is specified: Turns OCDemon logging on. All logging goes to a file named *wiggles.log*. Any existing *wiggles.log* is overwritten.

If *off* is specified: Turns OCDemon logging off.

If *message* is specified: Writes *message* to the log file.

If no arguments are specified: Displays the current logging status.

Do not use this command unless directed to do so by Green Hills Technical Support.

### memchunk [sr | sw | lr | lw size]

If the first argument is specified: Sets the memory access size as follows:

- *sr* sets small memory read size.
- *sw* sets small memory write size.
- *lr* sets large memory read size.
- *lw* sets large memory write size.

This command affects how large the packets for memory access are that *ocdserv* sends to the OCDemon device.

If no arguments are specified: Displays the current memory access size settings.

Do not use this option unless directed to do so by Green Hills Technical Support.
### memread byte | short | long addr

Performs a sized memory read or write to target memory.

The size of the access is 1, 2, or 4 bytes when the first argument is `byte`, `short`, or `long`, respectively.

For `memwrite`, `expr` specifies the value written to the address.

Both `expr` and `addr` may contain MULTI expressions using debug symbol labels. However, the address expression may not contain spaces. Using labels for which no debugging information exists may lead to unpredictable results.

### minmemaccess 1 | 2 | 4

Sets the minimum memory access size to 1, 2, or 4 bytes. By default, `ocdserv`'s smallest memory access size is 1 byte. However, some targets or boards may not support memory access sizes this small, and the `minmemaccess` command provides a way for `ocdserv` to work with such targets.

### reg [regname[expr]]

Displays or sets the register set of the target processor.

If no arguments are specified: Displays the entire register set.

If `regname` is specified: Displays the specified register.

If `regname` and `expr` are specified: Sets the specified register to `expr`. The expression `expr` may be a combination of labels, numbers, and so on, such as those accepted by the MULTI `print` command, and may contain internal white spaces.

### rom [on | off]

If `on` or `off` is specified: Turns ROM debugging mode on or off. If ROM debugging mode is set, `ocdserv` will not set the system call breakpoint.

If no argument is specified: Displays the current ROM setting.

### row byte | short | long addr expr

Performs a sized read of `addr`, logically ORs `expr` with the value that was read, and then performs a sized write back to the address.

### rrow reg expr

Reads the value from the specified register, logically ORs `expr` with the value that was read, and then writes the result back to the register.

### rst

 Resets the processor and forces it into debugging mode.
**Common OCD Connection Errors**

This section describes some common OCD connection problems. See also “Troubleshooting Your Macraigor OCD Hardware Setup” on page 105 for hardware setup problems that can affect your connection.

**The OCDemon Device Is Not Responsive**

If your Macraigor debug connection does not respond, work through the suggestions in “Troubleshooting Your Macraigor OCD Hardware Setup” on page 105.

**ocdserv Is Unable to Set syscall Breakpoint**

If you receive the error message:

```
ocdserv: Unable to set syscall breakpoint
```

after program download, it probably indicates that the target board memory was not set up correctly or that a linker directives file is not correct. To troubleshoot, use the `memread` and `memwrite` commands to verify that memory has been correctly set up on the board. For example:

| **secmap** [on | off] |
|-----------------------|
| If **on** or **off** is specified: Turns section map listing on or off. Section map listing must be on to use the `listsec` command. |
| If no arguments are specified: Displays the current setting of **secmap**. |

| **syscalls** [on | off] |
|-----------------------|
| If **on** or **off** is specified: Turns target system calls on or off. System calls are on by default. |
| If no argument is specified: Reports the current state of system calls. |
| System calls for performing input and output on the host are implemented with a software breakpoint. The **syscalls** command can be used to disable and enable this breakpoint and therefore system call servicing by the host. When running code from ROM, it may not be possible to set the system call breakpoint and you may need to use **syscalls off** to disable it. The default mode is **syscalls on**. |
OCD> memread long 0xa0020000
memread: long at 0xA0020000 = 0xF9F7FBFF succeeded.
OCD> memwrite long 0xa0020000 0xdeadbeef
memwrite: long at 0xA0020000 = 0xDEADBEEF succeeded.
OCD> memread long 0xa0020000
memread: long at 0xA0020000 = 0xDEADBEEF succeeded.

Also recheck your linker directives file to make sure that all program contents are linked to run in valid memory. You can use a link map of your executable to confirm that all program contents are in valid memory.

For more information about editing linker directives files, see “Step Two: Editing Linker Directives Files” on page 25 and the MULTI: Building Applications book.

**Windows Programs Are Slow to Load and Run**

If your Windows programs are slow to load and run, you may need to remove an obsolete dynamically linked library (DLL) file that is slowing performance. The Macraigor API loads this particular Macraigor-supplied Windows DLL if it is present, and it can slow down Windows considerably. However, **ocdserv** no longer uses this DLL, so it may be safely removed. To remove the DLL, go into your MULTI installation directory and remove or rename the file **mac_enet.dll**. Once the DLL is removed, the Macraigor API will no longer load the DLL, and Windows program loading and running will behave normally.
This Chapter Contains:

- Installing the Simulator for ARM (simarm)
- Connecting to the Simulator for ARM (simarm)
- Additional Commands for Simulator for ARM (simarm) Connections
- Simulation Modes
- Profiling with simarm
- Using System Calls in Your Code
- Connecting to the Simulator for ARM (simarm) as a Stand-alone Debug Server
MULTI includes a target simulator called simarm that allows you to execute and debug ARM code on your host system. This chapter supplements the general target connection information contained in Chapter 2, “Setting Up Your Target Hardware” of this book and Chapter 3, “Connecting to Your Target” in the MULTI: Debugging book with information about how to connect to and use the simarm Simulator.

Features available when using simarm with MULTI include:

- Symbolic disassembly
- Interactive breakpoint debugging
- Single-step execution into procedure calls
- Single-step execution over procedure calls

### Installing the Simulator for ARM (simarm)

The simarm Simulator is installed automatically when you install MULTI. No other installation steps are necessary.

### Connecting to the Simulator for ARM (simarm)

To help you connect to targets (including simulators) quickly and easily, MULTI allows you to create and save Connection Methods that specify the parameters of your connection.

For general instructions on how to create and use Connection Methods, see Chapter 3, “Connecting to Your Target” in the MULTI: Debugging book. The information in the following sections supplements the instructions provided there with information that is specific to Simulator for ARM (simarm) connections.

### Creating a Standard Simulator for ARM (simarm) Connection Method

When creating a new Standard Simulator for ARM (simarm) Connection Method, select Simulator for ARM (simarm) as the connection type in the Create New Connection Method dialog box.
Connecting to the Simulator for ARM (simarm)

For detailed information about creating new Standard Connection Methods, see “Creating a Standard Connection Method” in Chapter 3, “Connecting to Your Target” in the MULTI: Debugging book.

Using the Simulator for ARM (simarm) Connection Editor

In addition to the generic fields that appear on all Connection Editors for Standard Connection Methods, the Simulator for ARM (simarm) Connection Editor includes Connection and Debug tabs that provide settings and options specific to your simulated target and your host operating system.
When the **Connection Editor** is first displayed after you create a new Connection Method, the settings and options on the tabs are set to default values. Settings and options that are not available on your host operating system may appear dimmed. Some of the fields may require user input before the Connection Method functions.

All of the fields on the **Connection** and **Debug** tabs of the **Simulator for ARM (simarm) Connection Editor** are described in detail below. (See “The Connection Editor” in Chapter 3, “Connecting to Your Target” in the *MULTI: Debugging* book for a description of the other fields and options, which appear on all **Connection Editors**.)
Simulator for ARM (simarm) Connection Settings

<table>
<thead>
<tr>
<th>Processor</th>
<th>ROM mode</th>
<th>Simulate FPU</th>
</tr>
</thead>
</table>

**Processor**
Specifies which target processor to simulate. This field defaults to **ARM7**.

**ROM mode**
Specifies ROM Simulation Mode (as opposed to OS Simulation Mode). For more information about OS and ROM Simulation Modes, see “Simulation Modes” on page 133.

**Simulate FPU**
Specifies that the simulated target has a hardware floating point coprocessor.

Simulator for ARM (simarm) Debug Settings

<table>
<thead>
<tr>
<th>Alarm Timeout</th>
<th>Other Options</th>
</tr>
</thead>
</table>

**Warning** Do not change the settings on the debug tab unless you are instructed to do so by Green Hills Technical Support.

**Alarm Timeout**
Instructs the simulator to abort simulation using `alarm()` after the specified number of seconds.

**Other Options**
Allows you to add other, optional arguments directly to the command line. You should only use this field if directed to do so by Green Hills Technical Support.
Using Custom Simulator for ARM (simarm) Connection Methods

To connect to the Simulator for ARM with a Custom Connection Method, type the command given below, with the appropriate parameters and options, into the Start a Custom Connection dialog box and click Connect.

```
simarm [options]…
```

where options can be any non-conflicting combination of the simarm options listed in “Options for Custom simarm Simulator Connection Methods” on page 132.

**Note** You can also enter the above connection command from the Debugger’s command pane, where it must be preceded by the connect command.

See “Custom Connection Methods” in Chapter 3, “Connecting to Your Target” in the MULTI: Debugging book for more information about Custom Connections.

Options for Custom simarm Simulator Connection Methods

The following options are available when creating a Custom simarm Simulator Connection Method.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-cpu=cpu</td>
<td>Specifies which target processor to simulate. The default is ARM7.</td>
</tr>
<tr>
<td>-fpu</td>
<td>Specifies that the simulated target has a hardware floating point coprocessor.</td>
</tr>
<tr>
<td>-rom</td>
<td>Enables ROM mode on the simulator and sets the simulator to start from the reset vector. For more information, see “ROM Mode” on page 134.</td>
</tr>
<tr>
<td>-rom_use_entry</td>
<td>Enables ROM mode on the simulator and sets the simulator to start from the program’s entry point. For more information, see “ROM Mode” on page 134.</td>
</tr>
<tr>
<td>-Tseconds</td>
<td>Instructs the simulator to abort simulation using alarm() after the specified number of seconds.</td>
</tr>
</tbody>
</table>
Additional Commands for Simulator for ARM (simarm) Connections

The following commands, in addition to the commands listed in “Generic Debug Server Commands” on page 138, are available to simarm.

You can enter all of these commands directly into the simarm Target window. You can also enter these commands into the MULTI Debugger command pane using the target command. All of the commands for simarm are case-insensitive.

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>alloc addr size</td>
<td>Allocates size bytes at addr</td>
</tr>
<tr>
<td>assert_fiq</td>
<td>Assert FIQ interrupt</td>
</tr>
<tr>
<td>assert_irq</td>
<td>Assert IRQ interrupt</td>
</tr>
<tr>
<td>cycles</td>
<td>Displays number of cycles executed</td>
</tr>
<tr>
<td>cycles reset</td>
<td>Resets number of cycles executed</td>
</tr>
<tr>
<td>dcache command arg</td>
<td>Issue data cache command</td>
</tr>
<tr>
<td>freq f</td>
<td>Sets clock frequency to f (default is 20 MHz)</td>
</tr>
<tr>
<td>help</td>
<td>Lists available commands</td>
</tr>
<tr>
<td>icache command arg</td>
<td>Issue instruction cache command</td>
</tr>
<tr>
<td>insts</td>
<td>Displays instruction profile</td>
</tr>
<tr>
<td>reset</td>
<td>Causes processor to reset trap</td>
</tr>
<tr>
<td>timer count fiq</td>
<td>Assert FIQ every count instructions</td>
</tr>
<tr>
<td>timer count irq</td>
<td>Assert IRQ every count instructions</td>
</tr>
</tbody>
</table>

Simulation Modes

The simarm Simulator has two simulation mode: OS and ROM. These two modes are described in the following sections. OS Simulation Mode is the default mode. To enable ROM simulation, select the ROM mode option in the Connection Editor or pass the -rom or -rom_use_entry option.
OS Simulation Mode

In OS simulation mode (which is the default mode), the simulator provides an environment similar to a user process running under UNIX. The simulator allocates memory for the text and initialized and uninitialized data of the simulated program, loads it into memory, and starts executing at the entry point specified at link time. The stack pointer is initialized and any command line arguments listed after the user program name on the simulator command line are passed to the main program via the normal `argc/argv` convention. A limited set of system calls is supported for I/O purposes. This level of simulation is suitable for debugging user-level programs, but if your programs need a more accurate simulation of actual CPU behavior, you should run in ROM mode.

The simulator accepts the following system calls.

<table>
<thead>
<tr>
<th>access</th>
<th>alarm</th>
<th>brk</th>
<th>close</th>
</tr>
</thead>
<tbody>
<tr>
<td>creat</td>
<td>exit</td>
<td>fstat</td>
<td>getpid</td>
</tr>
<tr>
<td>link</td>
<td>lseek</td>
<td>open</td>
<td>read</td>
</tr>
<tr>
<td>stat</td>
<td>time</td>
<td>unlink</td>
<td>write</td>
</tr>
</tbody>
</table>

Note While only these system calls are implemented, many library functions use a combination of these calls to achieve their goals.

To implement these system calls, the simulator cooperates with the Green Hills Libraries by setting a breakpoint in the `.syscall` section. The simulator is then able to trap any call into this special section and treat it as a system call request by the library.

In OS simulation mode, only memory that was allocated initially, plus memory returned by the `brk` system call, is available. References to other memory addresses will be trapped by the simulator and cause the program to abort. Similarly, attempts to use exception-generating instructions also cause the program to abort. If the simulation must closely reflect actual hardware behavior, consider using ROM mode.

ROM Mode

ROM Mode simulates only the bare minimum hardware, such as CPU and memory systems.
In ROM mode, you can set the simulator to start at either the reset vector or at the program’s entry point. (The stack pointer must be set manually.) To cause the simulator to always start execution at the address of the system reset vector (address 0), as if the CPU were given a hard reset, and to ignore any program startup address specified at link time, select ROM mode on the Connection tab of the Connection Editor or include the -rom option in your Custom Connection Method. To start at the program’s entry point instead, use a Custom Connection Method and include the -rom_use_entry option in the Connection Method.

Other features of ROM mode include:

- Handling of exceptions as in the hardware, with the program counter transferred to the appropriate address. User code must be supplied to handle the exception appropriately.
- Prevention of arguments being passed to the running program.
- Requirement that user startup code initialize the stack pointer.
- Support of a limited set of system calls for I/O.

You can use ROM mode for writing and testing exception handlers and even parts of a real operating system. You should write at least some start-up code in pure assembly language, since this mode simulates a bare bones processor.

**Profiling with simarm**

You can use MULTI to profile with the simarm simulator. The simulated clock rate is 20 MHz, but the clock cycles used by each instruction are approximations. For more information about profiling, see Chapter 15, “Using the Profiler” in the MULTI: Debugging book.

**Using System Calls in Your Code**

The simulator can handle the following UNIX-style system calls in your code:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>close</td>
<td>creat</td>
<td>exit</td>
<td>lseek</td>
</tr>
<tr>
<td>open</td>
<td>read</td>
<td>rename</td>
<td>time</td>
</tr>
<tr>
<td>unlink</td>
<td>write</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Any attempt to use an unimplemented system call will cause the simulator to print an error message of the form:

syscall #nnnn not implemented

The value #nnnn is the number of the system call that was attempted. On most UNIX systems, these numbers can be found in the file /usr/include/syscall.h.

Connecting to the Simulator for ARM (simarm) as a Stand-alone Debug Server

To connect to the Simulator for ARM from the command line, independently of MULTI, enter:

```
simarm [options] image_file
```

where:

- **options** — Can be any non-conflicting combination of the options listed in “Options for Custom simarm Simulator Connection Methods” on page 132.
- **image_file** — Specifies and ELF-style executable file.

Connecting to the simulator in this way allows you to simulate your program, but does not offer the interactive debugging capabilities available through MULTI.
Appendix A

Green Hills Debug Server Command and Scripting Reference

This Appendix Contains:

- Green Hills Debug Server Commands
- The Green Hills Debug Server Scripting Language
This chapter describes generic debug server commands and a full scripting language that can be used with most Green Hills debug servers.

These commands and the scripting language are used only in legacy .dbs scripts, and are deprecated.

Green Hills Debug Server Commands

The commands described below are available to most Green Hills debug servers, but do not apply to simulator or operating system connections.

Most Green Hills debug servers also support additional commands. See “Additional Debug Server Commands” on page 144 for a listing of where to find descriptions of the additional commands (if any) available for the debug server that supports your specific target.

Generic Debug Server Commands

Unless otherwise noted, the commands listed in this section are available for all of the debug servers in this book.

You can enter all of these commands directly into the debug server Target window. You can also enter these commands into the MULTI Debugger command pane using the target command. All of the Green Hills debug server commands are case-insensitive.

```
addressof symbol
```

Returns the address of symbol. This command requires that an image be loaded in the MULTI Debugger.
**amask [mask] [value]**

If no arguments are specified, the current settings of the download mask and value are displayed. The default settings of `mask` and `value` are `0xffffffff` and `0`, respectively.

If `mask` and `value` are specified, `amask` sets the download mask and value to `mask` and `value`. When the debug server downloads a program, it will bitwise AND `mask` and bitwise OR `value` with the download addresses, as follows:

\[
\text{address} = (\text{address} \& \text{mask}) | \text{value}
\]

The `amask` command is useful for shifting download target addresses without relinking a program. However, the program being downloaded still retains its original relocations and might not run correctly at its new destination address without further support on the target.

**close fd**

Closes the specified file descriptor, `fd`.

**debug n**

Sets the debugging bit flags.

Do use this command unless directed to do so by Green Hills Technical Support.

**delrange addr**

Deletes a checked address range starting at `addr`. See `setrange` for more information about checked memory ranges.

**delrangeall**

Deletes all checked address ranges. See `setrange` for more information about checked memory ranges.

**echo [on | off]**

If no argument is specified, the current echo mode is printed.

If `on` or `off` is specified, printing of commands executed in a script are enabled or disabled.

**fprint fd string**

Prints `string` to the specified file, `fd`, with script variable expansion.

**fprintb fd integer**

Prints `integer` to the specified file, `fd`, in binary mode.

**fread fd identifier**

Reads one line of text from the specified file, `fd`. The line is stored as a variable with the specified `identifier`. The number of bytes read is returned. If there is an error, `-1` is returned.
freadb  fd identifier
Reads an integer from the specified file, *fd*, in binary mode. The integer is stored as a variable with the specified identifier. The number of bytes read is returned. If there is an error, -1 is returned.

getenv envName identifier
Stores the value of the environment variable, envName, in the script variable with the specified identifier.

halt
Halts execution of the target CPU and forces it into debugging mode.

help [command | group]
If no argument is specified, general help information and instructions for finding more detailed and specific help is printed.

If a command is specified, detailed help information about the specified command is printed.

If a group is specified, a list of all of the commands in the group with information about the arguments each command takes is printed. Valid command groups include **target**, **server**, and **scripting**.

Example 1:

```
> help server
help [<command> | <group>]
ddebug <n>
playdead
```

Example 2:

```
> help help
help [<command> | <group>]
Prints help information
```

listrange
Lists all checked ranges. See **setrange** for more information about checked memory ranges.
### listvars

Prints all variable identifiers in no particular order.

**Example:**

```plaintext
> str1="foo"
> str2="bar"
> i=100
> listvars
i
str1
str2
```

### load [all] [text] [data] [bss]

If no argument is specified: Displays the current `load` settings.

If one or more arguments are specified, `load` instructs which sections to include in host-to-target downloads. `.text` sections contain code, `.data` sections contain initialized variables, and `.bss` sections contain uninitialized data. Setting the `all` option includes all of these sections in the download.

You can combine the `load` command with the `noload` command.

Green Hills startup code clears all `.bss` sections so you do not need to download them. In most cases, the default setting is `text data`, but the default varies according to your debug server.

**Standard Example:**

```plaintext
> load all
Download Options: text data bss
```

**Advanced Example:**

```plaintext
> load all noload bss
Download Options: text data
```
**m [-d <size>] address[=val]**

If =val is not specified, the indicated memory address on the target is read.

If =val is specified, val to the specified memory address on the target is written.

Memory addresses and values must be specified in hexadecimal (with or without a leading 0x).

The optional -d argument can be used to set the access size to byte (-d1), short (-d2), or long (-d4). The default is -d4.

Examples:

> m 1000
7ca62b78
> m 1000=12345678
> m -d2 1000
1234

**nofail command**

Executes the specified command and always returns success.

**noload [all] [text] [data] [bss]**

If no argument is specified, the current noload settings are displayed.

If one or more arguments are specified, noload specifies which sections to exclude from host-to-target downloads. .text sections contain code, .data sections contain initialized variables, and .bss sections contain uninitialized data. Setting the all option excludes all of these sections from the download.

Use the command noload bss to exclude .bss sections from downloads. These sections are cleared to all zeros by programs compiled with Green Hills tools; therefore, downloading them to the target is usually unnecessary.

**open file**

Opens the specified file for writing and returns a file descriptor.

**print string**

Prints string with script variable expansion.

Example:

> print Test
Test

**random max**

Generates and returns a pseudo-random number between 0 and max-1.
**regnum**  *reg*[=val]

If =val is not specified, the specified register, *reg* is read and returned.

If =val is specified, **regnum** writes val to the specified register, *reg*.

Registers are specified by MULTIregister number and values in hexadecimal with no leading 0x.

Example:

```
> regnum 0=deadbeef
> regnum 0
Register 0=deadbeef
```

**rg**  *regname*[=val]

If *val* is not specified: Reads and returns the specified register, *regname*.

If *val* is specified: Writes *val* to the specified register, *regname*.

**run**  *address*

If *address* is not specified, the processor is run from the current program counter(PC).

If *address* is specified, it sets the program counter to *address* and then runs the processor.

Use this command very carefully. It is possible to run a program on the board when MULTI thinks it is halted, which causes unpredictable results.

A common use for this command is to run a ROM monitor program so that the monitor can set up the board properly before MULTI downloads a program.

**script**  *file*

Executes the commands in the specified script file, *file*, as if they were typed in line by line.

**setrange**  *addr*  *length*

Sets a checked address range beginning at *addr* and extending for *length* bytes.

A checked address range is a range of addresses that are considered inaccessible. Any memory access, read, or write to a checked address range can fail.

The **setrange** command is useful for restricting access to target memory that might be sensitive or volatile.

**setup**  *file*

Specifies a script file, *file*, to be automatically run immediately prior to every host-to-target download.

**sleep**  *n*

Suspends the debug server for *n* seconds.
status
Prints and returns the current status of the debug server using the following status codes:
0 Running
1 Stopped by breakpoint
2 Single step completed
3 Exception
4 Halted
5 Process exited
6 Process terminated
7 No process
8 (Unassigned)
9 Stopped by hardware breakpoint
10 Failure
11 Process ready to run
12 Host system call in progress
13 Target reset

step
Single steps the target CPU from the current program counter location.

undef variable
Removes variable and releases any memory associated with it.
Example:
> x=5
> undef x
> print $x
Error: variable undefined!

Additional Debug Server Commands
The following table indicates where you can find information about additional commands (if any) available for your particular debugging interface.
In addition to the commands listed in “Generic Debug Server Commands” on page 138 and the additional commands available to your specific debug server (see “Additional Debug Server Commands” on page 144), a full scripting language is available from the debug server command line for all of the debug servers in this book except the Simulator for ARM (simarm).

You can run scripts by:

- Entering scripts one line at a time at the command prompt. When entering a script line by line, nothing is executed until the top-level enclosing while or if statement is terminated.
- Storing commands in a script file and then running the file using the script command.
- Storing commands in a script file and then using the -setup option or setup command (or the Target Setup Script field in some Connection Editors)
to ensure that the script file is automatically executed immediately prior to every host-to-target download.

**General Notes**

When writing scripts, keep the following points in mind:

- There can be no more than one statement per line.
- Any line that has the # character as the first non-whitespace character is treated as a comment.
- Variables do not need to be declared before they are used.
- Variable types are determined automatically. For example, an identifier that is bound to an integer variable can later be assigned to a string variable.
- Variable and function names are not case-sensitive.

**Scripting Syntax**

The following sections describe and give examples of some features of the Green Hills debug server scripting language.

**Expressions**

Expressions in the Green Hills debug server scripting language are similar to C language expressions. The following table contains, in order of precedence, the valid operators you can use in expressions. Note that a string is treated like an integer if it contains a string representation of an integer.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Integer Function</th>
<th>String Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>Grouping of operators to ensure desired evaluation</td>
<td>Grouping of operators to ensure desired evaluation</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>Invalid</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>Invalid</td>
</tr>
<tr>
<td>%</td>
<td>Modulus</td>
<td>Invalid</td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
<td>Concatenation</td>
</tr>
</tbody>
</table>
The Green Hills Debug Server Scripting Language

<table>
<thead>
<tr>
<th>Operator</th>
<th>Integer Function</th>
<th>String Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>Invalid</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>Bitwise left shift</td>
<td>Invalid</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Bitwise right shift</td>
<td>Invalid</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>Alphabetic less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
<td>Alphabetic less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>Alphabetic greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
<td>Alphabetic greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>Equality test</td>
<td>Equality test</td>
</tr>
<tr>
<td>!=</td>
<td>Inequality test</td>
<td>Inequality test</td>
</tr>
<tr>
<td>&amp;</td>
<td>Bitwise AND</td>
<td>Invalid</td>
</tr>
<tr>
<td>^</td>
<td>Bitwise XOR</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bitwise OR</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Logical AND</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Assignments

The syntax for an assignment is:

```
identifier = expression
```

The expression is evaluated and the result is stored as a variable with the given identifier. String, integer, and array variables are supported. Identifiers can contain alphanumeric characters and the underscore (_) character, but cannot begin with a number.

### Arrays

Arrays are indexed lists of variables. Each cell in an array can contain a string, an integer, or an array. Array indexing begins with the index 0. An array can be created by assigning an entire array to an identifier or by assigning a string, an integer, or an array to one cell of an array. To reference a cell, follow the array identified by the index contained in square brackets. If an array cell contains another array, the elements in the second array are accessed by appending an
additional index in square brackets. The following example demonstrates the various methods of array access.

```
endl="\n"
bar[3] = 42
foo = { bar, 7, "hello" }
print $foo[2] world.$endl
if(foo[0][3]==bar[2+1])
   print Array indexing works.$endl
endif
```

Arrays are dynamically allocated in a sparse fashion. For example, making an assignment to `foo[0]` and then to `foo[100]` only allocates two array cells, and no space is used for the undefined array cells 1 through 99. After an array element has been allocated, it can only be deallocated by using the `undef` command on the top-level array identifier.

## Conditionals

The following table explains the syntax for conditionals.
The Green Hills Debug Server Scripting Language

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>if expression</td>
<td>If expression evaluates to zero, nothing happens.</td>
</tr>
<tr>
<td>statements</td>
<td>Otherwise, the block of statements between the if and endif lines are executed.</td>
</tr>
<tr>
<td>endif</td>
<td></td>
</tr>
<tr>
<td>if expression</td>
<td>If expression evaluates to zero, the debug server executes the block of statements between the else and endif lines.</td>
</tr>
<tr>
<td>statements</td>
<td>Otherwise, the block of statements between the if and else lines are executed.</td>
</tr>
<tr>
<td>else</td>
<td></td>
</tr>
<tr>
<td>statements</td>
<td></td>
</tr>
<tr>
<td>endif</td>
<td></td>
</tr>
<tr>
<td>if expression1</td>
<td>If expression1 does not evaluate to zero, the debug server executes the block of statements between the if and elif lines.</td>
</tr>
<tr>
<td>statements</td>
<td></td>
</tr>
<tr>
<td>elif expression2</td>
<td>If expression1 does not evaluate to zero and expression2 does not evaluate to zero, the debug server executes the block of statements between the elif and endif lines.</td>
</tr>
<tr>
<td>statements</td>
<td></td>
</tr>
<tr>
<td>[elif expressionX</td>
<td>If both expression1 and expression2 evaluate to zero, the debug server continues evaluating each of the subsequent elif expressions (if any) that occur before the endif statement and will execute the block of statement associated with each elif that does not evaluate to zero.</td>
</tr>
<tr>
<td>statements]...</td>
<td></td>
</tr>
<tr>
<td>endif</td>
<td>If none of the elif expressions evaluates to non-zero, the debug server will not execute anything.</td>
</tr>
</tbody>
</table>

Loops

The following table explains the syntax for loops.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>while expression</td>
<td>The statements between the while and endwhile lines are executed as long as expression does not evaluate to zero.</td>
</tr>
<tr>
<td>statements</td>
<td></td>
</tr>
<tr>
<td>endwhile</td>
<td>Be careful to avoid infinite loops. If an infinite loop occurs, you must shut down and restart the debug server.</td>
</tr>
</tbody>
</table>

Variable Expansion

To use script variables as arguments to Green Hills debug server commands, you must prepend the variable name with one or two $ characters.
• To pass a variable to a command in its default text representation, prepend the variable name with a single $ character. This passes a decimal string for integer variables and passes the string itself for string variables. An entire array cannot be given as an argument to a command.

• To pass an integer variable to a command as a hexadecimal string, prepend the variable name with two $ characters. Use this method with commands such as m that require arguments in hexadecimal form.

Variable expansion must be unambiguous. The script parser attempts to use the longest legal identifier name following the $ character. In the following example, the user has attempted to print the string bar after the expansion of the variable foo. The parser interprets this as printing the value of the variable foobar and reports that the variable is not defined:

```bash
>foo="foo"
>print $foobar
Error: variable undefined!
```

**Example Scripts**

You can use the following examples of the Green Hills debug server scripting language as a guide when writing your own scripts.
Example 1.

For this example, assume that the file `test.ascii` contains the following text:

This is a test of script file access in ascii mode.
This should print 3 lines of which this is the second.
And this is the third.

The following script commands access the file `test.ascii`:

```plaintext
file = open test.ascii
filecontents=""
totalchars=0
while(numlinechars=fread file line)
   filecontents=filecontents+line
   totalchars=totalchars+numlinechars
endwhile
close file
endl="\n"
print Read: $filecontents$endl
print Total of $totalchars characters read.$endl
```

The output of this example script on a UNIX host is:

Read: This is a test of script file access in ascii mode.
This should print 3 lines of which this is the second.
And this is the third.

Total of 130 characters read.
Example 2.

The following script is another example of accessing a file:

```
i=100
file = open temp.bin
while(i>0)
    fprintf file $i
    i=i-1
endwhile
close file
file = open temp.bin
sum=0
while(freadb file i)
    sum=sum+i
endwhile
close file
endl="\n"
print The numbers between 1 and 100 sum to $sum!$endl
```

The output of this example script is:

The numbers between 1 and 100 sum to 5050!

Example 3.

The following script calculates the CRC32 value of the memory range 0x010000 to 0x010100 and prints the result in the Target window. This script demonstrates the use of loops, conditional statements, expressions, variables, and other debug server scripting constructs in a real application.
# Change the following values to specify the memory
# range you want to calculate a CRC32 for.
# Note: locations from memstart to memend-1 are used
# to compute the CRC32 value.
memstart=0x010000
memend=0x010100
# This is the CRC32 polynomial. This is the same as
# is used in ethernet packets.
p=2+4+16+32+128+256+1024+2048+4096+65536+4194304+8388608+67108864
r=0
ptr=memstart
while(ptr<memend)
currbyte=m -d1 $$ptr
currbit=128
while(currbit)
test=r&(1<<31)
r=r<{1}
r=r|(currbyte&currbit)
if(test)
r=r^p
endif
currbit=currbit>>1
endwhile
ptr=ptr+1
endwhile
# This loop is for the 32 zeros appended to the
# original memory contents
i=0
while(i<32)
test=r&(1<<31)
r=r<{1}
if(test)
r=r^p
endif
i=i+1
endwhile
# Now the resulting 32 bit CRC is in r
# Many of the same ASCII control codes that are used
# in C are supported in debug server scripts.
endl="\n"
print CRC32 = $$r$endl
MULTI: Configuring Connections for ARM Targets
Using the gproxy Network Proxy Utility

This Appendix Contains:
- Network Proxy File Format
- Connecting to your Target
The Green Hills network proxy (gproxy) utility is run from the command line. Any messages or diagnostic output is displayed in the command prompt window. This utility is intended to be used to forward INDRT connections to embedded operating system targets that are not directly accessible from your debugging host computer but that are connected to another intermediate computer. That intermediate computer can run the gproxy utility to allow other debugging host computers to connect to the specified targets.

The gproxy command has the following syntax:


<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-port host_port</td>
<td>Specifies the UDP port on which to listen when forwarding a single UDP port. To forward TCP connections or multiple connections, you must use the -f argument. The default value is 2220.</td>
</tr>
<tr>
<td>-c</td>
<td>Causes diagnostic messages to be printed when connections are made.</td>
</tr>
<tr>
<td>-d</td>
<td>Causes diagnostic messages to be printed each time data is forwarded over a connection. With this option, gproxy will print the size of each data buffer forwarded and the source and destination.</td>
</tr>
<tr>
<td>-h</td>
<td>Prints gproxy command usage summary.</td>
</tr>
<tr>
<td>target[:port]</td>
<td>Specifies the name of the target to which to connect when forwarding a single UDP port. The port option specifies the port to connect to (the default port value is 2220). This argument is required unless -f is used to specify a connection file.</td>
</tr>
<tr>
<td>-f file</td>
<td>Reads a list of connections to forward from file. For a description of the file format, see “Network Proxy File Format” on page 157.</td>
</tr>
</tbody>
</table>
Network Proxy File Format

The connection file contains a list of network connections to forward, each on a single line. Each line is specified as follows:

\[
\text{in\textunderscore port} \ \text{TCP|UDP} \ [\text{in\textunderscore ip}] \ : \ [\text{out\textunderscore port}] \ \text{out\textunderscore ip}
\]

<table>
<thead>
<tr>
<th><strong>in_port</strong></th>
<th>The incoming port to listen on when forwarding.</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>UDP</td>
</tr>
<tr>
<td><strong>in_ip</strong></td>
<td>The IP address or hostname of the interface on which to listen. This option can be used to specify a particular interface to listen on if multiple network interfaces are available. If this option is not specified, connections to the specified port on all interfaces will be accepted for forwarding.</td>
</tr>
<tr>
<td><strong>out_port</strong></td>
<td>The target port to connect to. If this option is not specified, the incoming port number in_port will be used.</td>
</tr>
<tr>
<td><strong>out_ip</strong></td>
<td>The IP address or hostname to which to forward connections.</td>
</tr>
</tbody>
</table>

The # character is used to begin a comment. Any text that follows the comment character on the same line is ignored.

Some example connection lines are the following:

2220  UDP  10.0.0.10 : 2221  192.168.100.100
23   tcp : server2
Connecting to your Target

When `gproxy` is launched, one or more ports are specified and are forwarded to a different machine. As an example, assume `gproxy` is running on the machine `boardgw1`, and the following connection is specified in the connection description file:

```
3000 UDP : 2220 hwboard15
```

If your computer is able to access the computer `boardgw1` but not `hwboard15`, you could connect to the `hwboard15` using INDRT by specifying `boardgw1:3000` as the Target Name in the INDRT Connection Editor.

For more information about connecting with INDRT, see “Connecting to Your Target with INDRT” on page 92.

Some network ports that you may want to forward are the following:

<table>
<thead>
<tr>
<th>Port Type</th>
<th>Port Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP</td>
<td>69</td>
<td>TFTP</td>
</tr>
<tr>
<td>UDP</td>
<td>2220</td>
<td>INDRT</td>
</tr>
<tr>
<td>TCP</td>
<td>21474</td>
<td>MULTI EventAnalyzer</td>
</tr>
<tr>
<td>TCP</td>
<td>21484</td>
<td>MULTI ResourceAnalyzer</td>
</tr>
</tbody>
</table>
Index

A

address command, 138
ADP (Angel Debug Protocol), 56
-adr option, 78, 80
(ADP) Angel Debug Protocol, 78, 80
Agilent 16700 Logic Analyzer, 50
Agilent Processor Probe, 36
See also hpserv debug server
byte order, specifying target, 42, 47
connecting MULTI to, 39
Connection Editor, 40 to 42, 44
Connection Methods
creating, 39
Custom, 45 to 46
editing, 40 to 42, 44
Custom Connection Method
topics, 48
download settings, 46 to 47
download settings for, 45, 47
firmware supported by hpserv, 37
flash/ROM debugging, 43
hostname, specifying, 42, 45
installing, 37
IP address, specifying, 42, 45
list of supported ARM targets, 36
logging host-target
communications, 44 to 45
multiple-core debugging support, 49
power failure, target, 43
probe configuration options, 44, 46
processor type, specifying, 42
program sections, which to
download, 43, 46

single-stepping method, specifying, 43, 46
system calls, enabling and disabing, 43
system requirements, 36
trace support for ARM ETM targets, 36
troubleshooting, 38
amask command, 139
Angel Debug Protocol (ADP), 56, 78, 80
Angel Monitor, 56
Connection Editor, 62
avANCED settings, 65
collection settings, 63
download settings, 64
download settings, 64
Factory Connection Method, 77
eaproto, 84
installing, 59
options, 80
Standard Connection Method, 61
supported processors, 57
api command, 121
ARM (simarm) Simulator
connecting to, 128
features, 128
installing, 128
simulated processors, 128
ARM Embedded Trace Macrocell (ETM),
see Embedded Trace Macrocell (ETM)
armsd.map file, 65, 70, 75, 84
ARMulator
Connection Editor
download settings, 74
extension kit, 89 to 90

Green Hills Software, Inc.
Index

with Tracer module, 89 to 90
ARMulator Simulator, 56 to 57
Connection Editor, 72
    advanced settings, 75
    connection settings, 74
    debug settings, 76
CPU speed, 85
Custom Connection Method, 79
    examples, 85
cycle accuracy, 57
installing, 60
options, 80
Standard Connection Method, 61
    supported processors, 57
arrays, 147
assignments, 147
-attach option, 116
-attach option, 116
autobp command, 121
-autobp option, 116
auto-hardware breakpoint continuation, 111, 116, 121

B
-baud option, 80, 99
-big option, 116
-big endian option, 81
BIOS setup, 106
    EPP Mode, 105
blockfill command, 121
-bpsc option, 116
bpstep command, 52, 85, 122
-bpstep option, 81, 116
    hpsev, 46
breakdump command, 122
breakpoint continuation, 111, 121
breakpoint stepping, 66, 71, 75, 81, 85, 112, 122
    Agilent Processor Probe targets, 43, 46
breakpoint trap, 111
breakpoints
    hardware, 111, 117, 121
    software, 66, 71, 75, 81, 85, 111, 122
    step method, 112
    system calls, 124
-bss debug server option
    hpsev, 46
    .bss sections, 141 to 142
byte order, specifying target
    for Agilent Processor Probe, 42, 47

C
cable version checking, 113, 118
calls, system
    enabling and disabling
        for Agilent Processor Probe targets, 43
-cf debug server option
    hpsev, 46
checked address range, 143
clock speed
    setting, 122
close command, 139
commands
    for all debug servers, 138
    for hpsev debug server, 52
    for ocdsev debug server, 121
    for rdsev debug server, 85
    for the Simulator for ARM (simarm), 133
commands, debug server
    in .dbs scripts, 22
    in .mbs scripts, 22
compatibility mode profiling, 67, 72, 76, 83
conditionals, 148
-config option, 81
config window, 65, 70, 75, 81
configuration
    with passthrough command, 112, 118
connecting MULTI to your target
  list of supported ARM targets, 8
native targets, 8
overview, 7 to 8
using an Agilent Processor Probe, 39
connecting to your target
  through a terminal server, 99 to 100

**Connection Editor**
  Agilent Processor Probe (*hpserv*), 40 to 42, 44
Angel/JEENI (*rdiserv*), 62
  advanced settings, 65
  connection settings, 63
  debug settings, 66
  download settings, 64
ARMulator (*rdiserv*), 72
  advanced settings, 75
  connection settings, 74
  debug settings, 76
  download settings, 74
INDRT (*rtserv*), 93
  advanced settings, 96
  connection settings, 94
  debug settings, 96
Macraigor OCD (*ocdserv*), 107
  advanced settings, 110
  connection settings, 109
  debug settings, 112
Multi-ICE Probe (*rdiserv*), 67
  advanced settings, 70
  connection settings, 69
  debug settings, 71
  download settings, 69
**simarm** Simulator, 129
Simulator for ARM (**simarm**)
  connection settings, 131
  debug settings, 131

**Connection Methods**
  Custom
    Agilent Processor Probe
      (*hpserv*), 45 to 46
Angel Monitor (*rdiserv*), 77
ARMulator (*rdiserv*), 78
EPI JEENI Probe (*rdiserv*), 77
INDRT (*rtserv*), 97
Macraigor OCD devices
  (*ocdserv*), 113
Multi-ICE Probe (*rdiserv*), 78
Remote Debug Interface
  (*rdiserv*), 77
**simarm** Simulator, 132

**Standard**
  Agilent Processor Probe
    (*hpserv*), 39 to 42, 44
  Angel/JEENI (*rdiserv*), 61
  ARMulator (*rdiserv*), 61
  EPI JEENI Probe (*rdiserv*), 61
  Macraigor OCD (*ocdserv*), 107
  Multi-ICE Probe, 61
**simarm** Simulator (**simarm**), 128
connection modes
  specifying
    Agilent Processor Probe, 45
context-sensitive help, 10
conventions, typographical, 12
**core** command, 52
-**core** debug server option
  **hpserv**, 47
-**cpu** option, 79, 81, 114
CPU speed, 85
**cpuspeed** command, 85
cycle accuracy, 89
**cycles** command, 86

**D**
-**d** option, 116
-**data** option, 47, 117
  .**data** sections, 47, 117, 141 to 142
  .**dbs** setup scripts, see setup scripts, target
**debug 8** command, 106
**debug** command, 139
debug server commands, see commands, debug server
   for all debug servers, 138
   hpserv, 52, 133
   ocdserv, 121
   rdiserv, 85
debug server options, see options, debug server
debug servers
   debugging interfaces supported by, 21
   hpserv, 36
   installing, 16
   mpserv, 34
   ocdserv, 102
   rdiserv, 56 to 57, 59
   role of, 7
debugging
   flash/ROM
      Agilent Processor Probe targets, 43
   mode, selecting, 45
   multiple cores, 49
   real-time operating systems, 92
   run-mode, 92
debugging interfaces
   debug servers associated with
      various, 21
   list of supported, 8
delay command, 122
delrange command, 139
delrangeall command, 139
-device debug server option
   hpserv, 47
dlcheck command, 122
-dll option, 81
-dn option, 98 to 99
document set, 6
download checking, 117, 122
dynamic downloading, 95, 98
dynamically linked libraries (DLLs), 110, 118, 126

E
echo command, 139
Embedded Trace Macrocell (ETM)
   trace support with Agilent Processor
   Probe connection, 36
-endian debug server option
   hpserv, 47
endianness, see byte order, specifying target
EPI JEENI Probe, 56
   Connection Editor, 62
      advanced settings, 65
      connection settings, 63
      debug settings, 66
      download settings, 64
   Custom Connection Method, 77
      examples, 84
      installing, 59
      options, 80
   Standard Connection Method, 61
      supported processors, 57
EPP Mode, 105 to 106
error codes, 88
-ethernet option, 81
ETM trace support
   dual core support, 51
examples
   Custom Connection Methods
      Agilent Processor Probe
         (hpserv), 48
      Angel/JEENI (rdiserv), 84
      ARMulator (rdiserv), 85
      INDRT (rtserv), 100
      Macraigor OCD devices
         (ocdserv), 118
      Multi-ICE (rdiserv), 84
      Remote Debug Interface
         (rdiserv), 84
      running MacDemon, 120
      scripts, 150
   exception handling, 67, 72, 76, 82
exclusive serial port access, 96, 99
-exechwp option, 59, 82
expressions, 146

F
F1 key, for help, 10
flash
  debugging options
    for Agilent Processor Probe, 43
-forcefftp option, 98 to 99
fprint command, 139
fprintb command, 139
fread command, 139
freadb command, 140

G
getenv command, 140
GHSTrace option, 90
GHSTraceIgnoreFetch option, 90
gproxy utility
  command format, 156
  file format, 157
  overview, 156
Green Hills Debug Probes, 34
Green Hills Probe, 34
Green Hills Tracer module, 85
  configuring, 89
  options, 90
  with ARMulator, 89 to 90

H
halt command, 140
haltnum command, 123
hardware breakpoint support
  Remote Debug Interface (RDI), 58
  hardware breakpoints, 111, 116 to 117, 121
  hardware, target
    configuring, 16 to 18, 20, 28
    See also setup scripts, target
  installing, 16
help, see online help
help command, 140
Help menu, 10
-hostname debug server option
  hpserv, 45
hostname, specifying
  for Agilent Processor Probe, 42, 45
hp command, 53
hpserv debug server, 36
  See also Agilent Processor Probe
  -bpstep, 46
  -bss, 46
  -cf, 46
  commands, 52
  -core, 47
Custom Connection Method
  examples, 48
  -device, 47
dual ETM trace support, 51
-endian option, 47
installing, 38
mode= option, 45
role of, 36
setup= option, 45
-hostname option, 45
supported Agilent firmware, 37

I
-ice option, 79, 82
INDRT, 92
  connecting to, 92
Connection Editor, 93
  advanced settings, 96
  connection settings, 94
  debug settings, 96
Custom Connection Method, 97
  Ethernet, 97
  examples, 100
  serial, 99
Index

installation
  Agilent Processor Probe, 37
  Angel Monitor, 59
  ARM (simarm) Simulator, 128
  ARMuulator Simulator, 60
debug server, 16
EPI JEENI Probe, 59
JEENI Probe, 59
Macraigor OCD devices, 104
Macraigor OCDemon Raven, 105
Macraigor OCDemon Wiggler, 105
MULTI, 16
Multi-ICE Probe, 60
Remote Debug Interface (RDI), 59
target hardware, 16
instruction cache, 123
Integrated Development Environment (IDE), see MULTI Integrated Development Environment (IDE)
INTEGRITY real-time operating system, 9
INTEGRITY Real-Time Operating System (RTOS), 95, 98
interrupt sources, 53
disabling during target setup, 24
inticache command, 123
IP address, specifying
  for Agilent Processor Probe, 42, 45

J
JEENI Probe, 56
  installing, 59
  supported processors, 57
JTAG scan chain, 47

L
Launcher, see MULTI Launcher
libraries
  Remote Debug Interface (RDI), 66, 71, 76, 81
linker directives files, 25, 88
created by New Project Wizard, 16 to 18
creating custom, 20
editing, 25 to 26
for Macraigor OCD targets, 125
need for, 16
Linux targets, 9
listrange command, 140
listsec command, 123
listvars command, 123
-little option, 117
load command, 141
-loadall option, 117
-loaddir tftp_dir option, 98
log command, 123
-log option, 47, 117, 120
logging, 47, 97, 113, 117, 120, 123
logging host-target communications
  for Agilent Processor Probe, 44 to 45
Logic Analyzer
  collecting ETM trace, 50
loops, 149

M
m command, 88, 142, 150
-m option, 53
-macd option, 115
MacDemon, 103
  running, 119
  with Macraigor OCDemon Raven, 103
  with Macraigor OCDemon Wiggler, 102
Macraigor OCD devices, 102
  common connection problems, 125
  connecting to, 106
  Connection Editor, 107
    advanced settings, 110
    connection settings, 109
debug settings, 112
  Custom Connection Method, 113
examples, 118
dynamically linked libraries (DLLs), 126
installing, 104
  OCDemon Raven, 105
  OCDemon Wiggler, 105
linker directives files, 125
options, 116
Raven, 103
  with MacDemon, 103, 119
Standard Connection Method, 107
target system requirements, 104
troubleshooting, 105
Wiggler, 102
  supported processors, 102
  with MacDemon, 103, 119
.mbs setup scripts, see setup scripts, target
memchunk command, 123
memory size
  setting, 82
memory, target
  configuring using a setup script, 23
  memory map, defining, 25
  testing access to through your debugging interface, 28
memory.ld file, 25
  editing, 25
memread command, 124 to 125
-memsizet option, 82
memwrite command, 124 to 125
mini instruction cache, 123
minmemaccess command, 124
mode= debug server option
  hpserv, 45
modes, connection, see connection modes
mpserv debug server, 34
MULTI
  installing, 16
MULTI Integrated Development Environment (IDE)
  document set, 6
online help, 10
overview, 2
MULTI Launcher
  overview, 4
Multi-ICE Probe, 56
  Connection Editor, 67
    advanced settings, 70
    connection settings, 69
    debug settings, 71
    download settings, 69
  Custom Connection Method, 78
    examples, 84
  installing, 60
  options, 80
  Standard Connection Method, 61
  supported processors, 57
multi-ice.dll library, 60, 87
multiple core debugging
  with Agilent Processor Probe, 49
multiple cores
  with Agilent Processor Probe, 47
N
native targets
  connecting MULTI to, 8
Network proxy utility
  command format, 156
  file format, 157
  overview, 156
New Project Wizard
  configuring target hardware using, 16 to 17
  configuring target hardware with, 18
description, 16
testing target configuration with, 16, 18
-nobrk options, 82
-nodlcheck option, 117
-noexcl option, 99 to 100
nofail command, 142
-nohwbp option, 117
Index

noload command, 142
-nosyscalls option, 118
-noversion option, 118
-nozeropic option, 48

O
-ocddll option, 118
ocdserv debug server, 102
commands, 121
connecting with, 106
collection command syntax, 113
Connection Editor, 107
advanced settings, 110
connection settings, 109
debg settings, 112
Custom Connection Method, 113
examples, 118
installing, 104
MacDemon, 102 to 103
examples, 120
options, 120
running, 119
options, 116
Standard Connection Method, 107
starting, 106
-oldprofiling option, 83
on-chip debugging (OCD), 102
online help
Help menu, 10
overview, 10
UNIX systems, 11
Windows systems, 10
open command, 142
options
MacDemon, 120
ocdserv debug server, 116
rdiserv debug server, 80
simarm Simulator, 132
options, debug server
hpserv, 45 to 47
OS targets, 9
OSE targets, 9

P
-p option, 48
-parallel option, 83
passive attachment mode, 111, 116
passthrough command, 112, 118
-passthru option, 118
polling, OCD, 113
-port option, 98, 115, 120
position independent code (PIC), 48, 53
power failure, target
Agilent Processor Probe, 43
PowerPC 555 targets
Agilent Processor Probe
considerations, 42
PowerPC 8240 targets
Agilent Processor Probe
considerations, 42
print command, 142
processor type, specifying
for Agilent Processor Probe, 42
processors supported
for Macraigor OCDemon Raven
connections, 103
for Macraigor OCDemon Wiggler
connections, 102
profiling, 67, 72, 76, 83
program sections
download, specifying which to
for Agilent Processor Probe, 43, 46

R
random command, 142
rdiprop command, 86
rdiserv debug server, 56
and Tracer module, 89
Angel/JEENI Connection Editor
advanced settings, 65
connection settings, 63
download settings, 64
**ARMulator Connection Editor**
- advanced settings, 75
custom settings, 74
download settings, 76
download settings, 74
commands, 85
- exechwbp option, 59
installing, 59
**Multi-ICE Connection Editor**
- advanced settings, 70
custom settings, 69
download settings, 69
options, 80
- Standard Connection Method, 61
starting, 60
- supported interfaces, 56
- supported processors, 57
troubleshooting, 87
- real-time operating systems (RTOS)
- debugging, 92
- reg command, 53, 86, 124
- register reads and writes, testing, 27
- registers
  - reading, 53, 143
  - writing, 53, 143
- regnum command, 143
- regsdm command, 87
- Remote Debug Interface (RDI), 56
  - connecting to, 60
  - Custom Connection Method, 77
  - examples, 84
  - error codes, 88
- hardware breakpoint support, 58
- installing, 59
- libraries, 66, 71, 76, 81
- self-describing modules, 87
- supported interfaces, 56
- supported processors, 57
- troubleshooting, 87
- resetting the target, 53, 124
- rg command, 143
- ROM
  - debugging options
  - for Agilent Processor Probe, 43
- rom command, 124
- ROM mode, 132
- -rom option, 83, 118, 132, 135
- -rom_use_entry option, 132, 135
- row command, 124
- rrow command, 124
- rst command, 53, 87, 124
- rtserv debug server
  - connecting with, 92
  - connection command syntax, 97
- **Connection Editor**, 93
  - advanced settings, 96
  - connection settings, 94
  - debug settings, 96
  - Custom Connection Method, 97
  - Ethernet, 97
  - examples, 100
  - serial, 99
  - starting, 92
- run command, 143
- run-mode debugging, 92
- running scripts, 145
- S
  - scan chain, 47
  - script command, 143, 145
  - scripting language
  - arrays, 147
assignments, 147
conditionals, 148
examples, 150
expressions, 146
general debug server, 145 to 146
loops, 149
new (.mbs) vs. old (.dbs) style, 20, 28
variable expansion, 149
scripts
running, 145
writing, 146
secmap command, 125
section map, 125
self-describing modules, 87
-set option, 83
-setrange command, 143
setup command, 143, 145
setup files, 48, 84
-setup option, 48, 84, 118, 145
setup scripts
editing, 22 to 23
setup scripts, target
created by New Project Wizard, 16 to 18, 21
creating custom, 20 to 21
editing, 22
example, 25
need for, 16
new (.mbs) vs. old (.dbs) style, 20, 28
running .dbs scripts manually, 31
running .dbs scripts when connecting, 30
running .mbs scripts manually, 31
running .mbs scripts when connecting, 29
specifying and running, 28
specifying in Connection Editor, 19
testing commands for, 22
setup= debug server option
hperv, 45
simarm Simulator
connecting to, 128
Connection Editor, 129
collection settings, 131
debug settings, 131
Custom Connection Method, 132
features, 128
installing, 128
options, 132
simulated processors, 128
Standard Connection Method, 128
simulated processors
with ARM (simarm) Simulator, 128
simulation modes, 133
simulator
system calls, 135
Simulator for ARM (simarm)
commands, 133
simulation modes, 133
simulators
connecting to
ARM (simarm) Simulator, 128
features
simarm, 128
installing
ARM (simarm), 128
supported by MULTI, 8
single stepping, 52, 66, 71, 75, 81, 85, 112, 116, 122
method, specifying
for Agilent Processor Probe
targets, 43, 46
sleep command, 143
Slingshot, 34
software breakpoints, 66, 71, 75, 81, 85, 111, 116, 122
standalone_config.ld file, 25
editing, 25
standalone_ram.ld file, 25
standalone_romcopy.ld file, 25
standalone_romrun.ld file, 25
Index

status command, 144
step command, 144
stepint command, 53
SuperTrace Probe, 34
.syscall section, 118
syscalls command, 125
system calls, 111, 116, 118, 124
system requirements, see targets

target command, 52, 85, 121, 133
target connections, see connecting MULTI
to your target
ARM (simarm) Simulator, 128
INDRT, 92
Macraigor OCD devices, 106
Remote Debug Interface (RDI), 60
target system
resetting, 53
target system requirements
Macraigor OCD devices, 104
targets
Agilent Processor Probe, 36
configuring, 16 to 18, 20, 28
See also setup scripts, target
configuring memory using a setup
script, 23
debug servers for supported, 21
installing, 16
list of supported, 8
memory map, defining, 25
native, 8
OS, 9
system requirements
Agilent Processor Probe, 36 to 37
testing the configuration of, 26 to 28
terminal server connections, 99 to 100
.text sections, 141 to 142
TFTP directory, 95, 98
ThreadX targets, 9

Tornado targets, 9
trace
Agilent Processor Probe support for
(with ARM ETM), 36
dual ETM trace support, 51
Tracer module, 85
configuring, 89
options, 90
with ARMulator, 89 to 90
troubleshooting
Agilent Processor Probe setup, 38
Macraigor OCD connections, 105, 125
rdiserv debug server, 87
Remote Debug Interface (RDI), 87
typographical conventions, 12

UNIX hosts
debugging from with MacDemon, 103,
119
-usemapfile option, 84

variable expansion, 149
variables
declaring, 146
expansion, 149
type, 146

workspaces
overview, 5
writing scripts, 146
examples, 150

zeropic command, 53
-zeropic option, 48