

MULTI: Configuring Connections for V850 and RH850 Targets



**Green Hills Software
30 West Sola Street
Santa Barbara, California 93101
USA
Tel: 805-965-6044
Fax: 805-965-6343
www.ghs.com**

DISCLAIMER

GREEN HILLS SOFTWARE 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 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 to notify any person of such revision or changes.

Copyright © 1983-2013 by Green Hills Software. 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.

Green Hills, the Green Hills logo, CodeBalance, GMART, GSTART, INTEGRITY, MULTI, and Slingshot are registered trademarks of Green Hills Software. AdaMULTI, Built with INTEGRITY, EventAnalyzer, G-Cover, GHnet, GHnetLite, Green Hills Probe, Integrate, ISIM, u-velOSity, PathAnalyzer, Quick Start, ResourceAnalyzer, Safety Critical Products, SuperTrace Probe, TimeMachine, TotalDeveloper, DoubleCheck, and velOSity are trademarks of Green Hills Software.

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

For a partial listing of Green Hills Software and periodically updated patent marking information, please visit http://www.ghs.com/copyright_patent.html.

PubID: connect_v800-496213

Branch: <http://toolsvc/branches/release-branch-2013-5-bto>

Date: October 4, 2013

Contents

Preface	vii
About This Book	viii
Supported Target Connections for V850 and RH850	viii
The MULTI Document Set	x
Conventions Used in the MULTI Document Set	xi
 1. Using the Connection Editor	 1
General Information Settings	3
Target-Specific Tabs	4
The Command Line Viewer	5
Action Buttons	6
 2. Green Hills Monitor (monserv) Connections	 7
Supported Features for Green Hills Monitor (monserv) Connections	8
Connecting to Your Target with a Green Hills Monitor	8
Creating a Standard Green Hills Monitor (monserv) Connection	
Method	9
Using the Green Hills Monitor (monserv) Connection Editor	9
Using Custom Green Hills Monitor (monserv) Connection	
Methods	13
 3. Renesas V850/V850E ICE (850eserv2)	
Connections	19
Target System Requirements	20
Installing Your Renesas V850/V850E ICE	20
Additional Windows Setup and Configuration	20
Required Definition File for Network V850/V850E Interfaces	21
Using the Green Hills SuperTrace Probe to Collect Real-time Trace	
Data	21

Connecting to Your Target with an Renesas V850/V850E ICE	23
Creating a Standard Renesas V850/V850E ICE (850eserv2)	
Connection Method	23
Using the Renesas V850/V850E ICE (850eserv2) Connection	
Editor	24
Using Custom Renesas V850/V850E ICE (850eserv2) Connection	
Methods	29
Hardware Breakpoints	33
Troubleshooting	33
General Troubleshooting Checklist	33
Error Messages	34
Target Commands for Renesas V850/V850E ICE (850eserv2)	
Connections	35
Data and Register Commands	36
Execution Command	36
Event, Trigger, and Trace Setting Commands	36
Trace Display Commands	37
Emulator Configuration Commands	37
Other Commands	37
Target Command Descriptions	38
4. RTE (rteserv2) Connections	67
Supported Targets for RTE (rteserv2) Connections	68
Installing Your RTE	68
Checking Your RTE Connection	68
Using the Green Hills SuperTrace Probe to Collect Real-time Trace	
Data	69
Connecting the SuperTrace Probe to a Midas RTE Cube	70
Collecting Trace Data	70
Connecting to a RTE Target	70
Creating a Standard RTE (rteserv2) Connection Method	71
Using the RTE (rteserv2) Connection Editor	71
Commands for RTE (rteserv2) Connections	73
General rteserv2 Commands	73
Commands for Specific RTEs	74

5. Multi-core Simulator for V850 and RH850 (simrh850)	77
Connections	77
Installing the Multi-core Simulator for V850 and RH850 (simrh850) . . .	78
Connecting to the Multi-Core Simulator for V850 and RH850 (simrh850)	78
Creating a Multi-core Simulator for V850 and RH850 (simrh850)	
Connection Method	78
Using the Multi-core Simulator for V850 and RH850 (simrh850)	
Connection Editor	79
Using Custom Multi-core Simulator for V850 and RH850 (simrh850)	
Connection Methods	82
Additional Commands for Multi-core Simulator for V850 and RH850 (simrh850) Connections	84
Profiling with simrh850	85
Simulation Modes	85
OS Simulation Mode	85
ROM Mode	87
Unsupported Features	87
Connecting to the Multi-core Simulator for V850 and RH850 (simrh850) as a Stand-alone Debug Server	88
 A. Green Hills Debug Server Command and Scripting Reference	 89
Green Hills Debug Server Commands	90
Generic Debug Server Commands	90
Additional Debug Server Commands	96
The Green Hills Debug Server Scripting Language	96
General Notes	97
Scripting Syntax	97
Example Scripts	101
 B. Target-Specific Trace Options and Triggers	 105
Renesas V850E IECUBE Target-Specific Options	106

Renesas V850E2RV3 IECUBE2 Target-Specific Options	108
Renesas RH850 F1L Target-Specific Options	110
V850E/V850E2R RTE Target-Specific Options	112
V850 Target-Specific Trace Triggers	115
Index	117

Preface

Contents

About This Book	viii
The MULTI Document Set	x
Conventions Used in the MULTI Document Set	xi

This preface discusses the purpose of the manual, the MULTI documentation set, and typographical conventions used.

About This Book

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 configure connections to your target using the appropriate debug server. Specifically:

- The chapters in this book provide connecting instructions, features, commands, and options specific to each debug server. This information supplements the general connection instructions described in the documentation about connecting to your target in the *MULTI: Debugging* book.
- Appendix A, “Green Hills Debug Server Command and Scripting Reference”, documents the commands that are supported by some 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 5.0. The new scripting conventions are described in the documentation about MULTI scripts in the *MULTI: Scripting* book.



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 installation directory for release notes, **README** files, and other supplementary documentation.

Supported Target Connections for V850 and RH850

The following table lists the target connections supported for V850 and RH850. For detailed information about each type of target connection, see the appropriate chapter, as identified in the table.

Debugging interface	Green Hills debug server	Debug server chapter
Green Hills Monitor	monserv	Chapter 2, “Green Hills Monitor (monserv) Connections” on page 7
Renesas V850/V850E ICE	850eserv2	Chapter 3, “Renesas V850/V850E ICE (850eserv2) Connections” on page 19
RTE Server	rteserv2	Chapter 4, “RTE (rteserv2) Connections” on page 67
Simulator for V850 and RH850	simrh850	Chapter 5, “Multi-core Simulator for V850 and RH850 (simrh850) Connections” on page 77



Note

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*

This list may not be comprehensive. Contact Green Hills Technical Support if your particular debugging interface, monitor, simulator, or OS does not appear in the list.

The MULTI Document Set

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

- *MULTI: Getting Started* — Describes how to create, build, and debug an example project in MULTI.
- *MULTI: Licensing* — Describes how to obtain, install, and administer Green Hills licenses.
- *MULTI: Building Applications* — Describes how to use the compiler driver 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.
- *MULTI: Debugging* — Describes how to use the MULTI Debugger and its associated tools.
- *MULTI: Debugging Command Reference* — Describes how to use Debugger commands and provides a comprehensive reference of Debugger commands.
- *MULTI: Managing Projects and Configuring the IDE* — Describes how to create and manage projects and how to configure the IDE.
- *MULTI: Scripting* — Describes how to create MULTI scripts and contains information about the MULTI-Python integration.

For a comprehensive list of the books provided with your MULTI installation, see the **toc_target.pdf** file located in the **manuals** subdirectory of your installation.

Most books are available in the following formats:

- A printed book (select books are not available in print).
- Online help, accessible from most MULTI windows via the **Help** → **Manuals** menu.
- An electronic PDF, available in the **manuals** subdirectory of your installation.

Conventions Used in the MULTI Document Set

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.

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

Convention	Indication	Example
bold type	Filename or pathname	C:\MyProjects
	Command	setup command
	Option	-G option
	Window title	The Breakpoints window
	Menu name or menu choice	The File menu
	Field name	Working Directory:
	Button name	The Browse button
<i>italic type</i>	Replaceable text	-o filename
	A new term	A task may be called a <i>process</i> or a <i>thread</i>
	A book title	<i>MULTI: Debugging</i>
monospace type	Text you should enter as presented	Type <code>help command_name</code>
	A word or words used in a command or example	The wait [-global] command blocks command processing, where -global blocks command processing for all MULTI processes.
	Source code	<code>int a = 3;</code>
	Input/output	<code>> print Test</code> <code>Test</code>
	A function	<code>GHS_System()</code>
ellipsis (...) (in command line instructions)	The preceding argument or option can be repeated zero or more times.	debugbutton [name]...

Convention	Indication	Example
greater than sign (>)	Represents a prompt. Your actual prompt may be a different symbol or string. The > prompt helps to distinguish input from output in examples of screen displays.	> print Test Test
pipe () (in command line instructions)	One (and only one) of the parameters or options separated by the pipe or pipes should be specified.	call <i>proc</i> <i>expr</i>
square brackets ([]) (in command line instructions)	Optional argument, command, option, and so on. You can either include or omit the enclosed elements. The square brackets should not appear in your actual command.	.macro <i>name</i> [<i>list</i>]

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.

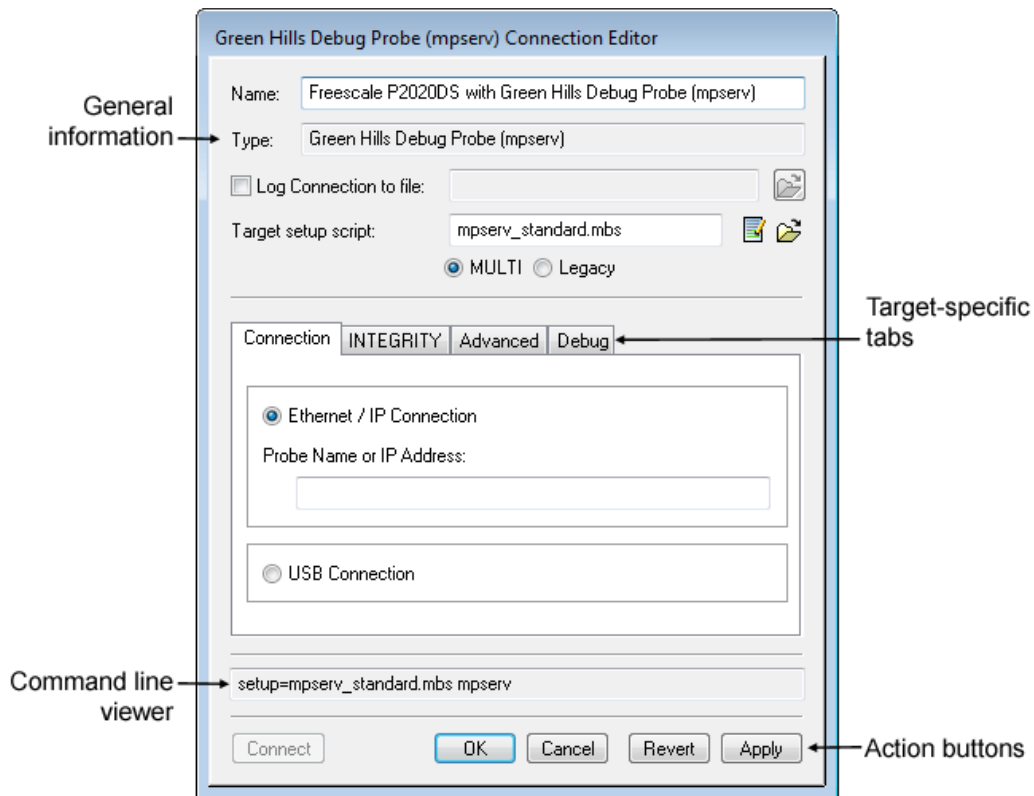
Chapter 1

Using the Connection Editor

Contents

General Information Settings	3
Target-Specific Tabs	4
The Command Line Viewer	5
Action Buttons	6

The **Connection Editor** is a target specific dialog that you can use to configure and edit Connection Methods.



The **Connection Editor** window for Standard Connection Methods has four sections, which are called out in the above graphic and documented in the following sections.

General Information Settings

The first section of the **Connection Editor** contains fields that display and/or set basic properties of a Connection Method. These fields are standard and appear on nearly all **Connection Editor** windows. The following table describes each field.

Name
Displays the name of the Standard Connection Method you define or edit. You can change the name of your method by editing this field.
Type
Displays the type of debugging interface the Connection Method is for, which determines which debug server MULTI uses. The connection type is specified when you create a Standard Connection Method. You cannot edit this read-only field.
Log Connection to file
Enables you to log transactions between MULTI and your debug server. To specify the file to write the log to, enter a pathname or click  to browse for the desired file. On Linux/Solaris, if you check the option box but do not specify a filename, MULTI writes the log to standard error output. Logging is disabled by default.
Target Setup script
Specifies your target setup script file. This is an optional field because not all targets require setup scripts. If you require a setup script to configure your board, specify the filename (and full path, if necessary) of the appropriate script or click  to browse for it. When this Connection Method is used, MULTI usually runs the specified script before each download. If you are editing a default Connection Method created by the Project Wizard for your processor-board combination, the necessary setup script file (if applicable) appears in this field automatically. To open the Editor on the target setup script, click . If no setup script is specified in this field, the Editor opens on a file with a filename derived from the debug connection's name. For more information about target setup scripts, see the documentation about configuring your target hardware in the <i>MULTI: Debugging</i> book.

Scripting language radio buttons

Specify the scripting language so that MULTI knows how to interpret the specified target setup script, where:

- **MULTI** — Indicates that the specified target setup script was written with the MULTI scripting language. This is the default setting. For more information about MULTI scripting in general, see the documentation about MULTI scripts in the *MULTI: Scripting* book.
- **Legacy** — Indicates that the specified target setup script was written with the debug server scripting language. For more information, see Appendix A, “Green Hills Debug Server Command and Scripting Reference” on page 89.

In previous versions of MULTI, target setup scripts had **.dbs** extensions and used the Green Hills debug server scripting conventions and associated debug server commands. Beginning with the v4.0 release, that scripting language was deprecated, and support for it will be discontinued in a future release. The default setup scripts created by the **Project Wizard** have **.mbs** extensions and use MULTI Debugger commands and scripting conventions instead. You must indicate which scripting language your setup script uses so that MULTI interprets it properly. When using the MULTI scripting language, you can still specify debug server commands by using the MULTI **target** command. For information about this command, see the documentation about general target connection commands in the *MULTI: Debugging Command Reference* book.

For more detailed information about editing and using setup scripts, see the documentation about configuring your target hardware in the *MULTI: Debugging* book.

Target-Specific Tabs

The second section of the **Connection Editor** contains one or more tabs that contain fields for setting options specific to your debugging interface, type of connection, and target architecture. The number of tabs and the fields that appear vary according to the type of Standard Connection Method you are editing.

When the **Connection Editor** is first displayed after you create a new Connection Method, the settings and options 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 can be used.

The following table describes in general terms the types of settings and options that appear on the most common tabs.

Tab	Description
Connection tab	The options on this tab usually require input for the Connection Method to function properly. The settings you choose for the connection options are probably different for each new target you connect to.
INTEGRITY tab	The option on this tab is useful if you are debugging an INTEGRITY system. For more information, see the documentation for set_runmode_partner in <i>MULTI: Debugging Command Reference</i> .
Download tab	The options on this tab allow you to customize the process of downloading programs to the target. For example, this tab allows you to control what program sections are downloaded to the target.
Advanced tab	The options on this tab allow a high degree of control over the details of your connection. These options can often be left in their default states. Use this tab sparingly because changing the advanced options from their default settings may cause problems with your connection.
Debug tab	The options on this tab are provided to help troubleshoot your connection. You should not change these settings unless instructed to do so by Green Hills Technical Support.

Most **Connection Editors** include at least a **Connection** tab and a **Debug** tab, but many contain additional tabs. For most connections, you need to set or edit some of the fields on these tabs for your Standard Connection Method to work properly. For more information about the **Connection Editor** settings available for your particular target, see the chapter that documents your debug server, or if you are a Green Hills Probe or SuperTrace Probe user, see the *Green Hills Debug Probes User's Guide*.

The Command Line Viewer

The third section of the **Connection Editor** displays the command line equivalents of your GUI selections. Default selections do not always have corresponding commands or options on the command line.

The command line viewer is a read-only field. You cannot make changes to the command line by typing in this field.

Changes you make in the GUI will not be reflected in the command line viewer until you click **Apply** or **OK**.

Action Buttons

The fourth section of the **Connection Editor** contains buttons that allow you to discard your edits, apply your edits, or connect using a Connection Method. The following table describes each button.

Button	Description
Connect	Records your changes, connects to your target using the Standard Connection Method you are editing, and closes the Connection Editor .
OK	Records your changes and closes the Connection Editor .
Cancel	Discards your changes and closes the Connection Editor .
Revert	Discards your changes and leaves the Connection Editor open.
Apply	Records your changes and leaves the Connection Editor open.



Note

Clicking **Connect**, **OK**, or **Apply** in the **Connection Editor** saves your Standard Connection Method. In the future, the Connection Method will appear in the list of available Connection Methods in the **Connection Chooser** and the **Connection Organizer**.

Chapter 2

Green Hills Monitor (monserv) Connections

Contents

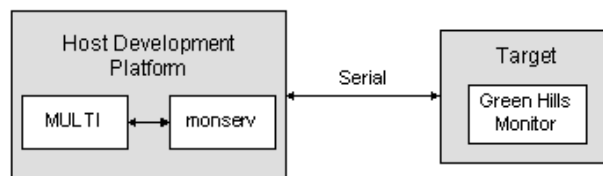
Supported Features for Green Hills Monitor (monserv) Connections	8
Connecting to Your Target with a Green Hills Monitor	8

This chapter supplements the general target connection information contained in Chapter 2, “Setting Up Your Target Hardware” of this book and the documentation about connecting to your target in the *MULTI: Debugging* book with specific information for Green Hills Monitor connections.

Supported Features for Green Hills Monitor (monserv) Connections

The Green Hills Monitor is a debug monitor that allows MULTI to download, run, and debug programs. The debug monitor allows you to read and write memory and program registers, set breakpoints, and perform other debugging functions.

MULTI uses the **monserv** debug server to connect to your Green Hills Monitor, as shown in the following diagram.



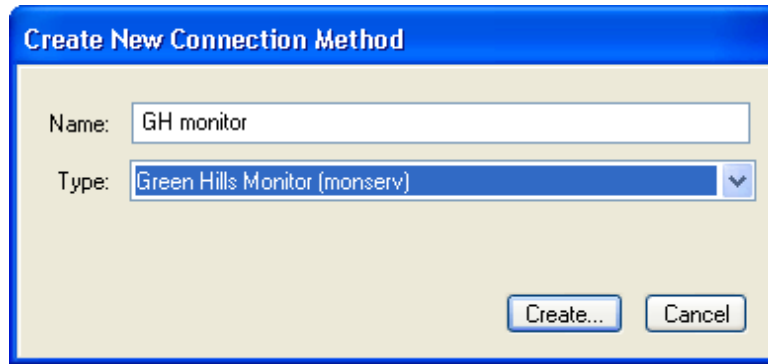
Connecting to Your Target with a Green Hills Monitor

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, the documentation about 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 Green Hills Monitor connections.

Creating a Standard Green Hills Monitor (monserv) Connection Method

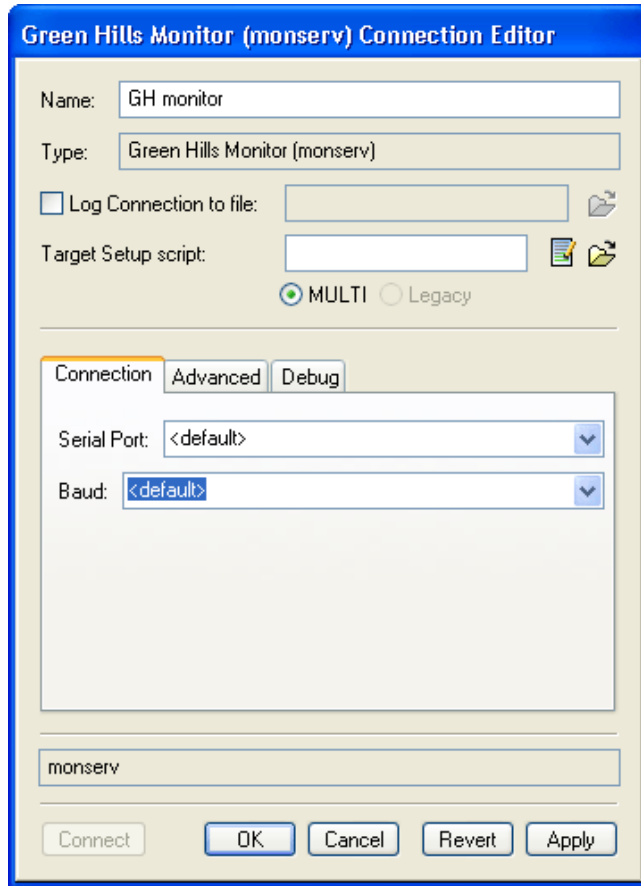
When creating a new Standard Green Hills Monitor Connection Method, select **Green Hills Monitor (monserv)** as the connection type in the **Create New Connection Method** dialog box.



For detailed information about creating new Standard Connection Methods, see the documentation about Standard Connection Methods in the *MULTI: Debugging* book.

Using the Green Hills Monitor (monserv) Connection Editor

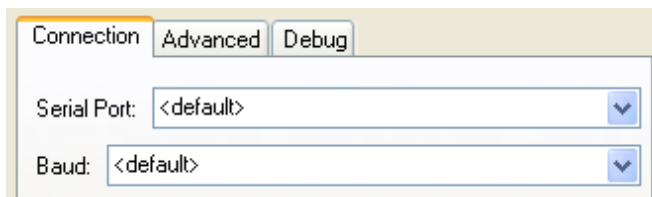
In addition to the generic fields that appear on all Connection Editors for Standard Connection Methods, the **Green Hills Monitor (monserv) Connection Editor** includes **Connection**, **Advanced**, and **Debug** tabs that provide settings and options specific to your target and host operating systems.



When the **Connection Editor** is first displayed after you create a new Connection Method, the settings and options 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 can be used.

All of the fields on the **Connection**, **Advanced**, and **Debug** tabs of the **Green Hills Monitor (monserv) Connection Editor** are described in detail in the tables below. (See the documentation about the Connection Editor in the *MULTI: Configuring Connections* book for a description of the other fields and options, which appear on all **Connection Editors**.)

Green Hills Monitor (monserv) Connection Settings



The screenshot shows the 'Connection' tab of the Green Hills Monitor (monserv) Connection Editor. It features two dropdown menus: 'Serial Port' and 'Baud', both currently set to '<default>'. The 'Advanced' and 'Debug' tabs are also visible.

Serial Port

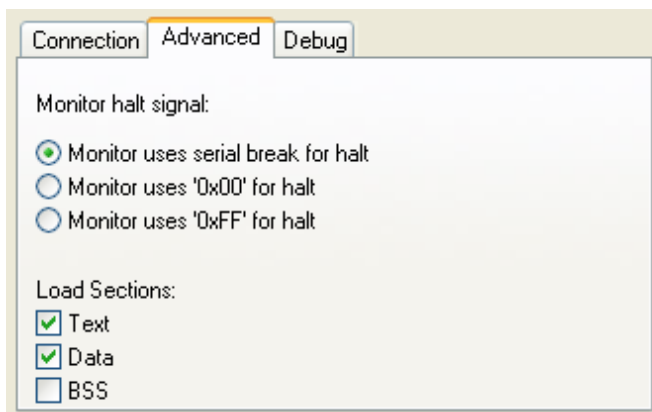
Specifies the serial port device to use when connecting to the target. The default for each supported operating system is:

- (Windows) **COM1**
- (Solaris) **/dev/ttya**
- (Linux) **/dev/ttyS0**
- (HP-UX) **/dev/tty00**

Baud

Sets the serial port communication speed. The default baud rate is 9600.

Green Hills Monitor (monserv) Advanced Settings



The screenshot shows the 'Advanced' tab of the Green Hills Monitor (monserv) Connection Editor. It contains two sections: 'Monitor halt signal' with three radio button options (the first is selected), and 'Load Sections' with three checked checkboxes.



Warning

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

Monitor halt signal

Specifies which signal will interrupt and halt target program execution and return control to the monitor. Select one of the available halt mechanisms:

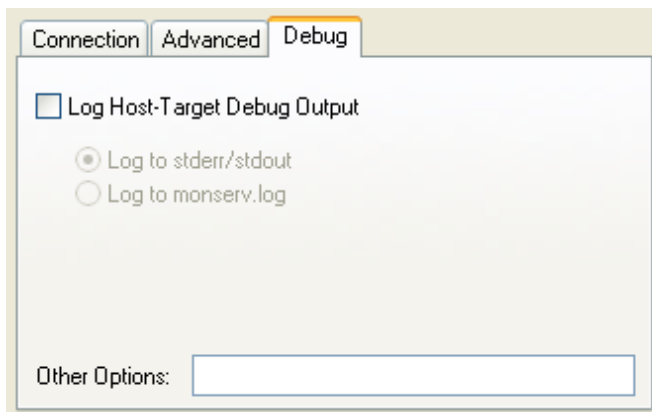
- **Monitor uses serial break for halt** (default)
- **Monitor uses '0x00' for halt**
- **Monitor uses '0xFF' for halt**

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.

Green Hills Monitor (monserv) Debug Settings



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 monserv and the Green Hills Monitor. Logging is disabled by default.
Log to stderr/stdout Log to monserv.log Allows you to specify the destination for host-target debug output. These fields will be dimmed unless Log Host-Target Debug Output is selected. Select the appropriate button to send the debug output to stderr/stdout (the default setting) or monserv.log .
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 Green Hills Monitor (monserv) Connection Methods

To connect to your Green Hills Monitor 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**.

monserv [-sp *port*] [-baud *baud_rate*] [*options*]...

where:

- **-sp *port*** is optional and specifies the serial port to use when connecting to the target. If no serial port is specified, the default is:
 - (Windows) **COM1**
 - (Solaris) **/dev/ttya**
 - (Linux) **/dev/ttyS0**
 - (HP-UX) **/dev/tty00**
- **-baud *baud_rate*** is optional and sets the serial port communication speed. The valid arguments for *baud_rate* are 9600, 19200, 38400, 57600, and 115200. If the **-baud** option is not used, **monserv** defaults to 9600 baud.
- *options* can be any non-conflicting combination of the **monserv** options listed in “Other Options for Custom Green Hills Monitor (monserv) Connection Methods” on page 14.

**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 the documentation about Custom Connection Methods in the *MULTI: Debugging* book for more information about Custom Connections.

For example connection commands, see “Example Custom Green Hills Monitor (monserv) Connection Methods” on page 17.

Other Options for Custom Green Hills Monitor (monserv) 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 Green Hills Monitor (monserv) Connection Method.

-bss
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.
-dbreak
Specifies that the character <code>0xFF</code> will interrupt and halt the target program execution. (If this option is not used, the default halt monitor signal is a serial break.)
-debug <i>n</i>
Sets the communication channel debugging level to <i>n</i> . See the note and table below for more information.
-loadall
Enables downloading of all program sections. By default, all sections are loaded except for uninitialized data (<code>.bss</code>) sections, which are cleared by the default Green Hills startup code. The -loadall option allows you to download all sections, including <code>.bss</code> .
-nobss
Disables uninitialized data (<code>.bss</code>) clearing by ocdserv . This option is enabled by default and is documented for backward compatibility only.
-nodata
Disables downloading of initialized data (<code>.data</code>) 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).

-zbreak

Specifies that the character `0x00` will interrupt and halt the target program execution. (If this option is not used, the default halt monitor signal is a serial break.)



Note

The **-debug** option should only be used when troubleshooting the communications channel between **monserv** and the debug monitor. The debugging level specified by *n* is the sum of the following bit flags.

Bit Value	Description
1	Displays communications with the debug monitor at the byte level. This shows every byte as it is transmitted or received. Every line preceded by <code>monserv:</code> shows what was sent by monserv to the monitor. Every line preceded by <code>monitor:</code> shows what was sent back by the monitor to monserv. Example: <pre>monserv: 01 0f 0f monitor: 05 monitor: 00 00 00 03 04 monitor: 07</pre>
2	Displays communications at the message level. Every line preceded by <code>request:</code> shows a message sent to the monitor and each line preceded by <code>response:</code> shows the monitor's response in a human readable format. Example: <pre>request: Monitor cpu/coproc id response: 304 request: Initialize response: Okay request: Write Register 17 = 109ec response: Okay</pre>
4	Displays timeouts, error detection, and retransmits. If MULTI seems too slow, this will tell you if the slowness is due to a noisy line. Example: <pre>*Timed out on read from Monitor* *sending retry, timeout in message body* *Checksum error in Monitor reply retry* *retransmitting previous message*</pre>
8	Randomly corrupts communications to test the retry mechanism. This option causes monserv to simulate a very noisy line by randomly changing, deleting, or inserting bytes. This option should be invoked as either -debug 13 or -debug 15 (all debugging options) so that you can see the retry messages and the modified bytes. Example: <pre>monserv: *****corrupting message***** monserv: *****randomizing byte *****</pre>

Example Custom Green Hills Monitor (monserv) Connection Methods

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

Example 2.1. Serial Connection

The following command connects **monserv** to a target through the **COM1** serial port on Windows at 115200 baud.

```
monserv -baud 115200
```

Example 2.2. Solaris Connection

The following command connects **monserv** to a target through **/dev/ttyb** on Solaris at 9600 baud and uses **0x00** as the halt signal.

```
monserv -sp /dev/ttyb -zbreak
```


Chapter 3

Renesas V850/V850E ICE (850eserv2) Connections

Contents

Target System Requirements	20
Installing Your Renesas V850/V850E ICE	20
Additional Windows Setup and Configuration	20
Connecting to Your Target with an Renesas V850/V850E ICE	23
Hardware Breakpoints	33
Troubleshooting	33
Target Commands for Renesas V850/V850E ICE (850eserv2)	
Connections	35

This chapter supplements the general target connection information contained in Chapter 2, “Setting Up Your Target Hardware” of this book and the documentation about connecting to your target in the *MULTI: Debugging* book with specific information for Renesas V850/V850E In-Circuit Emulator (ICE) connections.

Target System Requirements

The **850eserv2** debug server included with this distribution of MULTI is version 2.0 for the Microsoft Windows environment. This software runs under Microsoft Windows 7, Windows Vista, Windows XP, Windows 9x, Windows NT 4.0 and Windows 2000. This software does not run under any Windows 3.x versions (for example, Windows 3.1, Windows 3.11, or Windows NT 3.51).

Installing Your Renesas V850/V850E ICE

Before attempting to connect to your target, you must install your Renesas ICE. For instructions on hardware setup, diagnostics, and configuration, see your Renesas ICE documentation. As part of the setup process, you may need to download the latest **EXEC** library file, monitor, drivers, and/or device files from the Renesas web site.

The instructions in the remainder of this chapter assume that you have already installed MULTI, downloaded any necessary files from the Renesas Electronics web site, and set up your ICE. You should also know the I/O address of the PC interface card (for example, if your ICE connects to your PC with a PCI card) and know how to set environment variables under your operating system.

Additional Windows Setup and Configuration

In addition to following the relevant installation and configuration instructions contained in your Renesas hardware documentation, “Installing Your Renesas V850/V850E ICE” on page 20, and the documentation about configuring your target hardware in the *MULTI: Debugging* book, you must configure your Windows device drivers and your Windows environment variables.

Drivers for ICE interface cards are provided with the hardware. (You may need to download the latest drivers from the Renesas Electronics web site. See “Installing

Your Renesas V850/V850E ICE” on page 20.) For proper driver installation, see your Renesas documentation.

Required Definition File for Network V850/V850E Interfaces

For network connections to V850/V850E ICE, (for example, the IE-70000–MC-SV3 ICE), the **exws.ini** definition file must be modified to specify the hostname and port number of the V850/V850E ICE to which you want to connect. This definition file should then be copied to your Windows directory, which is typically at **c:\windows** or **c:\winnt**.

The following is an example **exws.ini** file:

```
;*****
:*  SV3 Driver environment                      *
;*                                              *
;*****
[WSIF]
HOST0=necice 0x7c6 ;HostName & Port No.

;File End
```

The semicolon (;) is a comment character; anything following a semicolon until the end of a line is ignored.

[WSIF] is the name of a section that is required in the **exws.ini** file.

The **HOST0=** option specifies the hostname and port number. The first field is for the hostname and the second field, delimited by whitespace, is for the port number. The hostname can also be written in IP address style.



Note

The port number of the IE-70000–MC-SV3 is fixed to 0x7c6 and cannot be changed.

Using the Green Hills SuperTrace Probe to Collect Real-time Trace Data

The following describes connecting the SuperTrace Probe to an Renesas IE Cube ICE, configuring the SuperTrace Probe, and collecting trace data. For more information about the SuperTrace Probe, see the *Green Hills Debug Probes User's*

Guide. For more information about connecting your debug server see “Connecting to Your Target with an Renesas V850/V850E ICE” on page 23.

Before connecting a SuperTrace Probe, you must be able to debug a program from MULTI with an Renesas IE Cube.

Connecting the SuperTrace Probe to an Renesas IE Cube

An IE Cube that can communicate with the SuperTrace Probe is identified by the Mictor connector located on the front of the IE Cube.

To connect to the IE Cube:

1. Plug the tracepod into the trace adapter (PATV8-01A).
2. Plug the trace adapter into the IE Cube.



Note

If this puts too much strain on the Mictor connector on the IE Cube, a Mictor extension cable is available from Green Hills Software (part number GH-ME-V85-mic), which can be inserted between the trace adapter and the IE Cube.

Configuring the SuperTrace Probe

Use a telnet console, a serial console, or **gpadmin** to set the following settings:

- Adapter must be set to `v850e-trace-24`.
- Target must be set to `v850e_trace`.



Note

There is a hyphen in the adapter name and an underscore in the target name.

For more information about the SuperTrace Probe, see the documentation about installing your probe in the *Getting Started* book for your probe.

Collecting Trace Data

To collect trace data from the target, you must specify the SuperTrace Probe in your **850eserv2** connection. For more information, see "**SuperTrace Probe is connected via**" in "Renesas V850/V850E ICE (850eserv2) Connection Settings" on page 26.

After the connection is made, verify that the target-specific trace options dialog is configured correctly for your target.



Note

Besides capturing trace data using a SuperTrace Probe (option only available for the IECUBE), trace can be obtained through the internal trace buffer available in both the IECUBE and IECUBE2

The IECUBE2 can also be used with the QB-V850E2-SP trace memory extension (also known as LTT or Long Term Trace) to increase the trace buffer size.

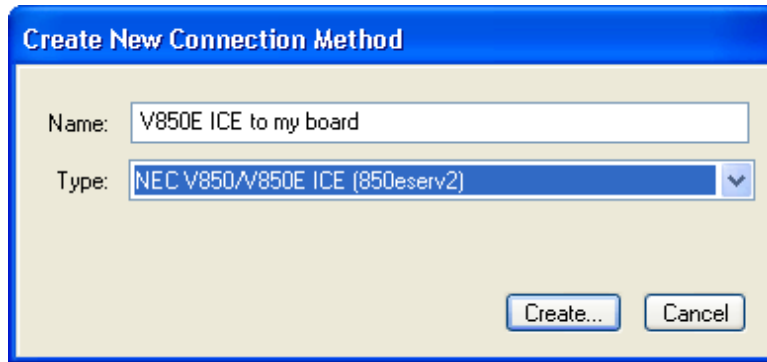
Connecting to Your Target with an Renesas V850/V850E ICE

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 the documentation about 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 Renesas V850/V850E ICE connections.

Creating a Standard Renesas V850/V850E ICE (850eserv2) Connection Method

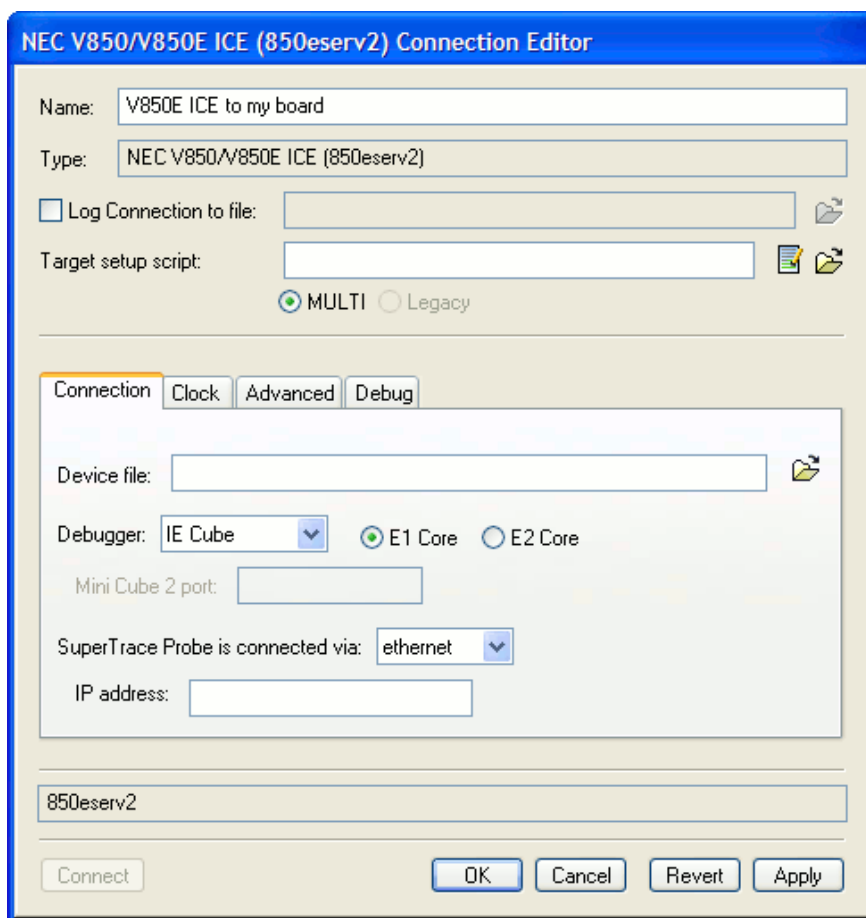
When creating a new Standard Renesas V850/V850E ICE Connection Method, select **Renesas V850/V850E (850eserv2)** as the connection type in the **Create New Connection Method** dialog box.



For detailed instructions on creating new Standard Connection Methods, see the documentation about connecting to your target in the *MULTI: Debugging* book.

Using the Renesas V850/V850E ICE (850eserv2) Connection Editor

In addition to the generic fields that appear on all Connection Editors for Standard Connection Methods, the **Renesas V850/V850E (850eserv2) Connection Editor** includes **Connection**, **Clock**, **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 **Connection**, **Clock**, **Advanced**, and **Debug** tabs are set to default values. The options on these tabs allow a high degree of control over the details of your connection, but can usually be left in their default states. Use the **Advance** tab carefully because changing the advanced options from their default settings can sometimes cause problems with your connection.

All of the fields on the **Connection**, **Clock**, **Advanced** and **Debug** tabs of the **Renesas V850/V850E (850eserv2) Connection Editor** are described in the subsequent sections.

Renesas V850/V850E ICE (850eserv2) Connection Settings

Device file

Sets the fully qualified location of the device file corresponding to the target to which you are connecting. To provide backward compatibility with previous versions of **850eserv**, instead of specifying the device file here, you can alternatively set the `IEPATH` and `DEVICE_FILE` environment variables (where `IEPATH` is the fully qualified directory containing the device file and `DEVICE_FILE` is the name of the device file without directory names). Either this field or the environment variables must be set to connect to your target.

There are several device files located in the **v850e** directory within the MULTI installation directory. Device files might have also been provided with your target. Check the documentation provided with your target to determine which device file to use.

Debugger

Indicates the type of Renesas debugger or In-Circuit Emulator (ICE) you are using to connect to your target. The default is the IE Cube. Other options include CD-NW, Mini Cube, Mini Cube 2, E1 Emulator JTAG, E20 Emulator JTAG, E1 Emulator Serial, E20 Emulator Serial, E1 Emulator LPD, E20 Emulator LPD.

E1 Core/E2 Core

Selects the V850 core type you are connecting to. When connecting to a V850E1 or V850E2/ME3 core, select **E1 Core**. When connecting to a V850E2 core, select **E2 Core**.

Serial port

Sets the port for your Mini Cube 2 or E1/E20 Emulator Serial. This is a required option if you are using an Renesas Mini Cube 2 ICE or E1/E20 Emulator Serial to connect to your target.

SuperTrace Probe is connected via

Allows you to collect trace data from your target using a Green Hills SuperTrace Probe, if your target supports collecting trace data. The SuperTrace Probe is used in tandem with the existing Renesas ICE connection (see **Debugger**).

The trace data collected with the SuperTrace Probe allows you to use TimeMachine and other trace facilities of the MULTI Debugger (see the documentation about analyzing trace data with the TimeMachine tool suite in the *MULTI: Debugging* book).

If you are using a SuperTrace Probe to collect trace data from your target, specify the location of the SuperTrace Probe here. If the SuperTrace Probe is connected via Ethernet, you must also specify the IP address or hostname of the SuperTrace Probe in the **IP Address** field.

For more information about configuring your SuperTrace Probe, see the documentation about installing your probe in the *Getting Started* book for your probe.

IP Address

If you are using a SuperTrace Probe to collect trace data from your target and the SuperTrace Probe is connected via Ethernet, you must specify the IP address or hostname of the SuperTrace Probe here.

Renesas V850/V850E ICE Clock Settings
Main clock

Specifies the main clock frequency in kHz.

Sub clock

Specifies the sub clock frequency in Hz.

Use sub clock when break is hit?

Specifies whether or not to use the sub clock when the program breaks.

Clock Rate

Specifies the CPU clock rate.

Renesas V850/V850E ICE (850eserv2) Advanced Settings

The screenshot shows a software window titled 'Renesas V850/V850E ICE (850eserv2)'. It has four tabs: 'Connection', 'Clock', 'Advanced' (which is highlighted), and 'Debug'. In the 'Advanced' tab, there are three checkboxes: 'Protect I/O memory' (checked with a green checkmark), 'Write 0's into BSS section' (unchecked), and 'Debug with RD850 or AZ850 (TIP)' (unchecked). Below the checkboxes are two text input fields. The first is labeled 'Registry ID code:' and the second is labeled 'Environment variables:'.

Protect I/O memory

Prevents **850eserv2** from reading or writing I/O memory, effectively read/write protecting the I/O memory.

Writes 0's into BSS Section

Writes zeros to the `.bss` section of the target before running the executable on the target. This option might be necessary if your setup script does not clear the `.bss` section.

Debug with RD850 or AZ850 (TIP)

Connect to an Renesas RD850 task debugger or an AZ850 event analyzer.

Registry ID code

Specifies a Windows Registry key. If you do not know what your Windows Registry key is, leave this field empty.

Environment variables

Sets the specified environment variables before launching **850eserv2**. The environment variables string is of the form:

```
env1=val1,env2=val2,...
```

where `env1`, `env2`, are the names of environment variables to set and `val1`, `val2` are the respective values to assign to those variables.

Renesas V850/V850E ICE (850eserv2) Debug Settings

Connection Clock Advanced **Debug**

Communications Log with Debugger:

☒ No log

☐ Log to stderr

☐ Log to 850eserv.log

☐ Log Trace retrieval

☐ Display result of flashing target

☐ Log TIP process in tipserv

Other Options:

Communications log with debugger

Controls logging of all communications between **850eserv2** and the Renesas ICE. Log output can be saved to a file (**850eserv2.log**), directed to `stderr`, or disabled.

Log trace retrieval

Logs trace retrieval events to **850eserv2_trace.log**.

Display result of flashing target

Displays diagnostic status information when writing to flash memory during download.

Log TIP process in tipserv

Logs TIP events to **tipserv.log**. This option is available when **Debug with RD850 or AZ850 (TIP)** is selected in the **Advanced** tab.

Other Options

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

Using Custom Renesas V850/V850E ICE (850eserv2) Connection Methods

To connect to your Renesas V850/V850E ICE 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**.

[setup=filename.mbs] 850eserv2 [options]...

where:

- **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 **850eserv2** command. See the documentation about configuring your target hardware in the *MULTI: Debugging* book for more information about setup scripts.

- *options* can be any non-conflicting combination of the **850eserv2** options listed in “Options for Custom Renesas V850/V850E ICE (850eserv2) Connection Methods” on page 30.



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 the documentation about Custom Connection Methods in the *MULTI: Debugging* book for more information about Custom Connections.

Options for Custom Renesas V850/V850E ICE (850eserv2) Connection Methods

The options listed below are available for Custom Renesas V850/V850E ICE (850eserv2) Connection Methods. For additional information, see the manual **sv-v850e2-us-*.pdf**.

-2m
Specifies that the CPU clock operates at 10 MHz. This option is only valid when used with -cdnw .
-cdnw
Indicates that 850eserv2 is connecting to a CD-NW ICE.
-D
Enables logging of communications between 850eserv2 and the ICE monitor. The log is written to 850eserv2.log .

-dck20
Specifies that the CPU clock operates at 20 MHz. This option is only valid when used with -cdnw or -minicube or E1/E20 Emulator JTAG.
-df=device_file
Specifies the fully qualified location of a device file corresponding to the target to which you are connecting. This option is the same as the Device File 850eserv2 Connection Editor option described in “Renesas V850/V850E ICE (850eserv2) Connection Settings” on page 26.
-eljtag
Indicates that 850eserv2 is connecting to E1 Emulator JTAG.
-ellpd
Indicates that 850eserv2 is connecting to E1 Emulator LPD.
-elserial
Indicates that 850eserv2 is connecting to E1 Emulator Serial.
-e20jtag
Indicates that 850eserv2 is connecting to E20 Emulator JTAG.
-e20lpd
Indicates that 850eserv2 is connecting to E20 Emulator LPD.
-e20serial
Indicates that 850eserv2 is connecting to E20 Emulator Serial.
-env=env_vars key
Sets the specified environment variables before launching 850eserv2 . <i>env_vars</i> is of the form: env1=val1, env2=val2 where <i>env1</i> , <i>env2</i> , are the names of environment variables to set, and <i>val1</i> , <i>val2</i> , are the respective values to assign to those variables.
-l
Enables logging of communications between 850eserv2 and the ICE monitor. The log is written to <code>stderr</code> .
-id key
Specifies a Windows Registry key. If you do not know what your Windows Registry key is, leave this field empty.
-iecube
Indicates that 850eserv2 is connecting to an IE Cube ICE.

-minicube
Indicates that 850eserv2 is connecting to a Mini Cube ICE. Note that the -minicube and -minicube2 options are mutually exclusive.
-minicube2
Indicates that 850eserv2 is connecting to a Mini Cube 2 ICE. For a Mini Cube 2 connection, you must also specify the -p option. Note that the -minicube and -minicube2 options are mutually exclusive.
-noiop
Allow 850eserv2 to read and write I/O memory. By default, 850eserv2 does not read or write I/O memory, effectively read/write-protecting I/O memory on the target, and gives errors like "I/O Area(0xff400000-0xff741fff) is Protected" (unless the SFR command is used). This option lifts that self-imposed restraint from 850eserv2 , which allows 850eserv2 to read and write I/O memory.
-p port
Indicates the port for the Mini Cube 2 ICE or E1/E20 Emulator Serial to which you are connecting.
-stp ip_address
Indicates the IP address or hostname of the SuperTrace Probe that you are using to collect trace data from the target. This is the same as selecting STP for the SuperTrace Probe is connected via option in the 850eserv2 Connection Editor, which is described in "Renesas V850/V850E ICE (850eserv2) Connection Settings" on page 26.
-t3v
Specify that the E1 Emulator should supply 3v power to the target. This option is only available with the E1 Emulator.
-t5v
Specify that the E1 Emulator should supply 5v power to the target. This option is only available with the E1 Emulator.
-usb
Indicates that you have connected a SuperTrace Probe via USB to collect trace data from the target. This is the same as selecting USB for the " SuperTrace Probe is connected via " option in the 850eserv2 Connection Editor, which is described in "Renesas V850/V850E ICE (850eserv2) Connection Settings" on page 26.
-X0
Writes zeros to the <code>.bss</code> section of the target before running the executable on the target. This option might be necessary if your setup script does not clear the <code>.bss</code> section.

Example Custom Renesas V850/V850E ICE (850eserv2) Connection Methods

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

Example 3.1. Connecting 850eserv2 to a QB-V850ESJG3L-TB board with E1 Emulator Serial

```
850eserv2 -df=df3738.800 -elserial -p csib0 -e1 -t3v  
-dclock=5000,0,swoff -noiop -id ffffffffffffffffffffff
```

Hardware Breakpoints

There are different types of execute hardware breakpoints supported, and the number of each type can vary depending on the target. Some (type "f") will stop at the address where the breakpoint is set (before the instruction is executed), while others (type "e") stop at the next instruction. **850eserv2** will use the type "f" breakpoints first. For example, with an IE Cube ICE, the first two execute hardware breakpoints set will be type "f", and later breakpoints will be type "e". The **BRS** command can be used to see the types of the breakpoints that are set.

Troubleshooting

The following sections describe some common problems that may occur when you connect to or use **850eserv2**.

General Troubleshooting Checklist

If you are experiencing problems with your **850eserv2** connection, working through the following checklist can help you locate the problem.

- Are you using the correct version of Windows? This version of **850eserv2** only runs under Windows 7, Windows Vista, Windows XP, Windows 9x, Windows NT 4.0, and Windows 2000. It does not run under any Windows version 3.x, such as Windows 3.1, Windows 3.11, or Windows NT 3.51.

- Is all of your V850/V850E ICE hardware set up properly? If your ICE requires a PC interface card, is that card properly installed and configured? Does the ICE have power?
- Did you reboot the PC after installation?
- Are all the Windows or UNIX environment variables properly set?
- If you are running Windows NT, did you make sure you have the appropriate privileges to change the Windows NT Registry and to set System or User variables?
- Does the device file you specified in the Connection Editor under **Device File** (or specified by using the `IEPATH` and `DEVICE_FILE` environment variables) exist at the specified location? Is that the correct device file for your target?
- If you were using COFF previously, did you rebuild all your object files for the new ELF format? Do you have COFF-specific assembler directives in your assembly files that should be ported to ELF?

Error Messages

The examples below show some common error messages and explain their likely causes.

Example 3.2. Error Message: Unable to connect

If you receive an error message that is something like:

```
Unable to connect to emulator, error cal or0x1a0: fatal err (missing monitor program)
Debug server 'c:\green\850eserv2' has aborted.  remote: no remote connection established.
```

850eserv2 cannot find the V850/V850E ICE monitor **ie850.mon** or **ie850e.mon**. Make sure the environment variable `IEPATH` is set up properly.

Example 3.3. Error Message: Unable to connect

If you receive an error message that is something like:

```
Unable to connect to emulator, error 0x3a0: status err
Debug server 'c:\green\850eserv2' has aborted.  remote: no remote connection established.
```

the V850/V850E ICE may not be receiving power. Click **OK** and then check the power supply.

Example 3.4. MULTI Dialog Box: Server Message Timed Out

Server Message Timed Out, Terminate Connection?

If you receive this message, click **OK**. The V850/V850E ICE may not be powered, or the physical connection may have problems. Check the power and connection.

Example 3.5. Memory Protection Errors (Windows 98)

Windows 98 does not have true memory protection. If you encounter any General Protection Fault Errors, Fatal Errors, or Illegal Operation Errors, reboot your PC and reset the V850/V850E ICE.

Target Commands for Renesas V850/V850E ICE (850eserv2) Connections

Unlike most Green Hills debug servers, **850eserv2** does not support all of the commands listed in “Generic Debug Server Commands” on page 90. The tables below list the target commands available to **850eserv2**. Full descriptions and examples of each command appear in “Target Command Descriptions” on page 38. The commands in the summary tables below have been grouped into the following function categories:

- Data and register commands
- Execution commands
- Event, trigger, and trace commands
- Trace display commands
- Emulator configuration commands
- Other commands

You can enter all of these commands directly into the **850eserv2 Target** pane. You can also enter these commands into the MULTI Debugger command pane using the **target** command. For additional information, see the manual **sv-v850e2-us-*.pdf**.

Data and Register Commands

The following data and register commands are accepted by **850eserv2**. See “Target Command Descriptions” on page 38 for more detailed descriptions and examples of each command and its arguments.

Command	Description
A	Assembles code one line at a time
M	Displays or changes memory
REG	Displays or changes register values
SFR	Displays SFR values
U	Disassembles object code

Execution Command

The following execution command is accepted by **850eserv2**. See “Target Command Descriptions” on page 38 for more detailed descriptions and examples of this command and its arguments.

Command	Description
RST	Resets the CPU and/or the emulation hardware

Event, Trigger, and Trace Setting Commands

The following event, trigger, and trace setting commands are accepted by **850eserv2**. See “Target Command Descriptions” on page 38 for more detailed descriptions and examples of each command and its arguments.

Command	Description
B	Causes a break
BRA	Sets bus event detectors
BRS	Sets execution event detectors
SA or SHOWALL	Displays current trace analyzer settings
T or TRACE	Specifies conditions for tracing

Trace Display Commands

The following trace display commands are accepted by **850eserv2**. See “Target Command Descriptions” on page 38 for more detailed descriptions and examples of each command and its arguments.

Command	Description
TD or TDISPLAY	Displays the trace buffer
TF or TFILTER	Specifies trace buffer filtering
TMODE	Specifies the trace mode
TRUN	Restarts the trace analyzer
TS or TSTOP	Halts the trace analyzer
TSEARCH	Searches the trace buffer

Emulator Configuration Commands

The following emulator configuration commands are accepted by **850eserv2**. See “Target Command Descriptions” on page 38 for more detailed descriptions and examples of each command and its arguments.

Command	Description
COMBO	Displays or changes the COMBO break status
CPU	Displays or changes the CPU internal ROM/RAM setting
MAP	Specifies the emulator memory configuration
PIN or PINMASK	Sets the masked processor pins

Other Commands

The following additional commands are accepted by **850eserv2**. See “Target Command Descriptions” on page 38 for more detailed descriptions and examples of each command and its arguments.

Command	Description
850win	Launches the 850win configuration window.

Command	Description
CLOCK	Specifies the clock source
HELP	Displays command summary
IEPORT0	Controls the internal port0 of the ICE
IEPORT1	Controls the internal port1 of the ICE
LOG	Opens or closes a log file
MODE	Specifies the CPU mode
NOHALT	Exits MULTI without issuing a reset to the ICE
NOLOAD	Turns download suppression on or off
POWER	Turns power on or off
PROFILE	Specifies profiling support
RMEM	Displays the contents of the real-time RAM periodically
RRAMBASE	Sets or displays the base address of the real-time RAM area
TIMEBASE	Sets or displays the frequency of the CPU clock and the scalar value of the trace time-tag counter
TIMER	Shows executed clocks
VERIFY	Turns memory verify on or off

Target Command Descriptions

The following commands are accepted by **850eserv2**. You can enter these commands directly into the **850eserv2 Target** pane or you can enter them into the MULTI Debugger command pane using the **target** command.

A [*address*]

Assembles code one line at a time, where *address* specifies the target address for the assembled code. If no address is specified, assembly begins where the last **A** command left off.

When this command is issued, the command pane enters an interactive mode in which it displays the disassembly of the current instruction at the target address and prompts you to enter a new assembly line. After assembling the input and modifying the emulator memory, the command pane updates the target address, displays the next disassembled line, and prompts for more assembly input. If you want the currently displayed instruction to be left unchanged, press **Enter**. The instruction will not be changed and the next instruction will be displayed for modification.

To terminate the interactive mode, enter **END** or a period (.) for the assembly line.

Symbolic constants can be used when specifying either the address or operands in the source line. The operators **sdaoff**, **zdaoff**, **hi**, and **lo** are supported. The name **ZERO** is treated as register **RO**.

For an example of how to use this command, see “Additional Examples of 850eserv2 Target Commands” on page 60.

B [**K** | *event_expression* ...]

Specifies or displays which event detectors cause breaks.

Once you have used the **BRA** and **BRS** commands to specify how the event detectors work, you can use the **B** command to assign some or all of them to cause a break. The *event_expression* argument specifies the event(s) and/or link(s). Multiple events and/or links can be specified if they are separated by a pipe (|) symbol. The **K** option clears all breakpoints.

If you enter the command without any parameters, the list of currently assigned events will be displayed.

Any new **B** command overrides the previous break setting.

Example:

```
brk>BRS 1 a=main
brk>BRA 2 A=data2, L 0x10 W=0x00xx
brk>B BRS1|BRA2
brk>B
break on:
BRS1: A=main <0x001000ba> P=0xXX
BRA2: A=data2 <0x001002f0>, 0x100300 W=0x000000XX RW
      P=0xXX
brk>LINK a l=brs1
brk>B BRA2|LINKA
brk>B
Break on:
BRA2: A=data2 <0x001002f0>, 0x100300 W=0x000000XX RW
      P=0xXX
LINKA: l=BRS1
brk>
```

BRA [# [A=address_or_range] [x=data/mask] [P=mask4]]

Sets or displays the hardware bus event detectors. If no arguments are specified, the **BRA** command will display the current settings.

Each of the optional arguments is described in detail below.

- The # argument specifies the identification number (1-8) of the event detector. If you specify an ID that is already being used, the previous setting will be overwritten. (If you specify a range for the address field, you will not be able to use all of the event detectors.)
- **A=address_or_range** specifies an address or range for the event as follows: **A=address** specifies a single address expression, **A=start,end** specifies an address range beginning at *start* and ending at *end*, **A=start, L length** specifies an address range beginning at *start* and extending for *length* bytes.

You cannot enter both a single address and a range. If you specify a range for the address field, you will not be able to use all of the event detectors. If you do not enter an address value, any address will be used.

- The **x=data/mask** option specifies the access size and data/mask values. The data comparator detects reads and writes of specific values at the addresses. The size of the access (*x*) can be specified as **B** (byte), **H** (halfword), **W** (word), or **D** (any size, default access).

You can use a symbolic constant to specify the data, but you cannot include a mask if you do. If you do not enter a data size/value, a halfword access and a mask of all ones will be assumed.

- The **status** option specifies the type of bus transaction to detect. Valid arguments for **status** are: **RW** (read or write; this is the default), **RO** (read only), or **WO** (write only).
- **P=mask4** specifies an external probe qualifier of 4-bit width. You can specify an additional qualifier based on the external probe tips 1-4. The **mask4** value is entered as a 4-bit hex or binary value/mask.

There are eight bus event detectors, each consisting of an address/range, a data/mask, a status, and an external probe comparator. The output of each of these comparators are ANDed together to produce an event. If a starting and ending address range is specified, then two of the bus event detectors will create the event (one for the start and one for the end).

Hex values are indicated by a 0x prefix, while binary input is indicated by an 0b prefix. To specify a value with a mask, use x characters for the nibbles or bits you want to be ignored. For example, D=0x245C will only detect a data access of 0x245C, while D=0x245X will detect any data value in the range 0x2450-0x245F.

For an example of how to use this command, see “Additional Examples of 850eserv2 Target Commands” on page 60.

BRS [# [A=address_or_range] [P=mask4]]

Sets or displays the hardware execution event detectors. If no arguments are specified, the **BRS** command will display the current settings for all detectors. To display the current settings for one detector, specify the identification number (#) without any other arguments.

Each of the optional arguments is described in detail below.

- The # argument specifies the identification number (1-14) of the execution event detector. If you specify an ID that is already being used, the previous settings will be overwritten. (If you specify a range for the address field, you will not be able to use all of the event detectors.)
- **A=address_or_range** specifies an address or range for the event as follows: **A=address** specifies a single address expression, **A=start,end** specifies an address range beginning at *start* and ending at *end*, **A=start, L length** specifies an address range beginning at *start* and extending for *length* bytes.

You cannot enter both a single address and a range. If you specify a range for the address field, you will not be able to use all of the event detectors. If you do not enter an address value, any address will be used.

- **P=mask4** specifies an external probe qualifier of 4-bit width. You can specify an additional qualifier based on the external probe tips 1-4. The *mask4* value is entered as a 4 bit hex or binary value/mask. Hex values are indicated by an 0x prefix, while binary input is indicated by an 0b prefix.

There are 14 execution event detectors in the emulator, each consisting of an address and an external probe comparator. The output of the two comparators are ANDed together to generate an event. They will cause an event when the specified address or an address in a range is executed and if the external probe pins match the **P=mask4** specification. If you specify a range of addresses, two of the available execution event detectors will be used, one for the start and one for the end.

Example:

```
brk>BRS 1 A=0x2000 P=0x1
brk>BRS 2 A=main
brk>BRS
BRS1:  A=0x2000 P=0x1
BRS2:  A=main <0x001000ba> P=0xXX
brk>
```

CLOCK [int | ext]

Specifies which clock is used. The **int** argument specifies the internal clock of the ICE. The **ext** argument specifies an external clock.

When this command is entered, the RESET signal at the CPU becomes active and internal registers and the SFR are initialized.

If no argument is specified, the **CLOCK** command displays the current clock source.

COMBO [E | D [N]]

Changes or displays the COMBO break status, as described below. If no arguments are specified, the **COMBO** command displays the current COMBO break status.

The **COMBO E** command enables COMBO mode, which allows interrupts to occur while the program is halted. **COMBO D** disables the COMBO break (this is the default).

The **N** option can only be used if the **D** option is also set. **N** specifies that NMIs can be accepted.

If the emulator is in COMBO mode and a program breaks in an interrupt handler (ISR), only the higher interrupt or NMI can be accepted and processed.

When you enable COMBO, avoid halting program execution at the ISR prologue. If you halt at the ISR prologue, saving and/or restoring EZPC/EZPSW may cause the program to crash.

Example:

```
brk>COMBO
COMBO break is disabled
brk>COMBO E
brk>COMBO
COMBO break enabled
brk>
```

CPU [R=*rom*] [A=*ram*]

Changes or displays the CPU internal ROM and RAM sizes (in KB) for emulation. The **R=*rom*** argument sets the internal ROM size and the **A=*ram*** argument sets the internal RAM size (see below for appropriate values). If no arguments are specified, the **CPU** command displays the current CPU settings.

Appropriate values for *rom* are 32, 64, 128, 256, 512, or 1024, depending on the contents of the device file corresponding to the target.

Appropriate values for *ram* for the V850 ICE are 1, 2, 3, 4, 6, 8, 10, 12, 16, 20, 24, or 28, depending on the contents of the device file corresponding to the target.

Appropriate values for *ram* for the V850E ICE are 1, 2, 3, 4, 6, 8, 10, 12, 16, 20, 24, 28, 36, 44, 52, or 60, depending on the contents of the device file corresponding to the target.

Example:

```
brk>CPU R=32 A=1
brk>CPU
ROM size=32K RAM size=1K
brk>
```

HELP

Displays command summary.

IEPORT0 [[R]][output0 | output1]]

Controls the internal ICE port **port0**, which generates a special signal in the Evachip. This port is used in certain ICEs for customized CPUs.

If **output0** is specified, the **port0** port generates the signal 0. If **output1** is specified, the port generates the signal 1.

If **R** is also specified (with one of the other options), RESET is asserted first, **port0** is set to the designated value, and then RESET is de-asserted.

IEPORT1 [[R]][output0 | output1]]

Controls the internal ICE port **port1**, which generates a special signal in the Evachip. This port is used in certain ICEs for customized CPUs.

If **output0** is specified, the **port1** port generates the signal 0. If **output1** is specified, the port generates the signal 1.

If **R** is also specified (with one of the other options), RESET is asserted first, **port1** is set to the designated value, and then RESET is de-asserted.

LINK [A | B | C] [K | *n=event_expression ...*] [C=count]

Links events with a sequencer, where **A**, **B**, or **C** specifies the sequencer, *n* specifies the level of enables or disables (see below), and **C** specifies a pass count for the output of the sequencer. The **K** argument clears (kills) all settings.

If you enter the **LINK** command with only the link number or without any options, the current settings of the link event(s) will be displayed.

You can link events together with one of three event sequencers (A, B, or C). Each sequencer has 4 levels of enables and one level of disable, specified by *n=event_expression* as follows:

- **1=event_expression** specifies a list of Enable4 events.
- **2=event_expression** specifies a list of Enable3 events.
- **3=event_expression** specifies a list of Enable2 events.
- **4=event_expression** specifies a list of Enable1 events.
- **D=event_expression** specifies a list of disable events.

You can link any number of events to each enable level or to disable. To enter a list of events for an input, put a pipe character (|) between the events.

When the events occur in the proper sequence, a link event is generated that may be tied to a break or trace function. Enable4 will be used as the trigger condition for the PASS counter.

You do not have to specify all levels. Levels that are not specified are assumed to be enabled. For example, if you specify Enable4 and Enable3 only, a link will be generated when the two conditions are satisfied. Always begin the events setting from Enable4, Enable3, Enable2, and Enable1.

If the link will be used as a sectional trace event, only the first three enables can be used as the start condition. The end condition, specified with the **TRACE** command, will use the fourth enable condition. You can clear the current settings for the link by using the **K** argument.

The **C** option specifies a pass count for the output of the sequencer and should be specified with one or more *n=event_expression* lists.

For an example of how to use this command, see “Additional Examples of 850eserv2 Target Commands” on page 60.

LOG [*filename* | E | D]

Opens or closes a log file. The *filename* argument specifies a new log filename and begins logging. **E** enables output to the log file. **D** disables output to the log file. If no arguments are specified, the **LOG** command displays the current log status.

You can use the **LOG** command to open a log file to record all output from the command line window. The *filename* can include a *drive:path* specification. If you do not include an filename extension, **.TXT** is assumed. To specify a filename without an extension, enter the name, followed by a period (that is, **FILENAME.**). Output logging begins as soon as you enter the **LOG** command with a filename.

If there is a log file open, you must disable that log file before opening a new one. If the log filename specified already exists, you will be asked whether to append to it or to delete the file before beginning to log. To append to the file, enter **A**. To delete the current file contents and start new logging, enter **D**. The **LOG** command can be aborted at this point by entering a period (.).

You can temporarily disable log file output with the **D** option. To resume log file output, use the **E** option.

M [**B** | **H** | **W**] [**C**] *address* [=*value*] [**L**=*length*]

Displays or allows changes to memory beginning at *address*.

The memory contents can be displayed and edited in bytes (**B**), halfwords (**H**), or words (**W**). The default data size is bytes. If you use **B**, **H**, or **W** to specify a size, the size setting will be saved in memory and will not need to be entered again.

If you specify **C**, you will be prompted with the current value of the memory. You can enter a new value or accept the current value. The program will then prompt with the next memory location. To terminate input, enter **END** or a period (.). (The *length* parameter is not valid when **C** is specified.)

You can also change memory contents by using the =*value* argument to provide data. Interactive mode will be disabled and only the specified memory location will be modified.

The **L**= option allows you to specify how many bytes to display. The default number of bytes to display is 4 rows of 16 bytes (irrespective of the size setting).

If an address is not specified, the last memory display address +1 is used as the start address.

Example:

```
brk>m c 0x100
0x00000100: 78
0x20
0x00000101: 56.
brk>m c 0x100=0x12345678
brk>m 0x100
0x00000100: 12345678 6C6C6568 000A216F 259F853F
brk>m b 0x100
0x00000100: 78 56 34 12 68 65 6C 6C 6F 21 0A 00 3F 85 9F 25
brk>
```

MAP [**K** | (**W**=|**R**=|**U**=|**G**=*start,end*|*start*, **L** *length*)]

Sets the emulator memory configuration, as described below. If you do not specify any arguments, the **MAP** command displays the current memory map.

To direct the emulator to read or write a range of memory, use one of the following options:

W specifies emulator read/write memory.

R specifies emulator read only memory.

U specifies user (target) memory.

G specifies a guard area.

Specify address ranges using the format **start, end** (an address range beginning at *start* and ending at *end*) or **start, L length** (an address range beginning at *start* and extending for *length* bytes). Addresses specified for this command must be aligned to a 1 MB boundary.

The **K** option cancels (kills) all memory ranges.

Target memory is provided on your development board. The emulator will drive the external bus lines to access it through the pod. For all other types of memory, the emulator will not drive the external lines.

For all other memory mapping, **R** (read only) and **W** (read/write) specify where in the address space the emulator will provide memory blocks. For blocks specified with **R**, the emulator will provide an illegal access break if an attempt is made to write to it.

For the V850 ICE, there are only two contiguous 1 MB blocks of memory that the emulator can provide. One block is fixed at 0x0 and the other is located on any 1 MB boundary. The fixed block can be used only when the ROM-less mode is specified with the **MODE** command (a total of 2 MB emulation memory is available). When the internal ROM is used, the fixed 1 MB block is not available (a total of 1 MB emulation memory is available). In this case, you cannot, for example, set a RAM segment at 0x100000 and another at 0x200000, since both need to be mapped by the second memory block.

For the V850E ICE, there are two contiguous 1 MB memory blocks that the emulator can provide. Each block has sixteen 64 KB blocks that can be specified as **W**, **R**, **U**, or **G**. Mapping is allowed for these blocks anywhere between 0x0 and 0x3ffffff.

Specifying **R**=0xffff000, 0xffffffff is not allowed because this address range maps to the I/O area, which is readable and writable.

Example:

```
brk> map w=0x000000,0x0ffffff cs0
brk> map r=0x200000,0x2ffffff cs1
brk> map tr2=0x100000,0x1ffffff
```

MODE [romless16 | romless8 | single16 | single | target]

Specifies the CPU mode using one of the following arguments:

romless16 specifies ROM-less mode with 16-bit external bus.

romless8 specifies ROM-less mode with 8-bit external bus.

single16 specifies single-chip mode with 16-bit external bus.

single specifies single-chip mode with 8-bit external bus.

target selects a mode by pins MD2 to MD0 on the target

If no argument is specified, the **MODE** command displays the current CPU mode.

NOHALT [E | D]

Controls whether or not to issue a reset when exiting MULTI.

The **D** option causes a reset to be issued to the ICE when exiting MULTI. This is the default.

The **E** option specifies that no reset be issued to the ICE when exiting MULTI.

If no arguments are specified, the **NOHALT** command displays the current settings.

Example:

```
brk> NOHALT
Disable IE to continue session
brk> NOHALT e
Enable IE to continue execution
brk>
```

NOLOAD

Toggles suppression of data downloading to the ICE. When **NOLOAD** is enabled, downloading is suppressed, even if a **LOAD** command is entered from MULTI. To disable downloading suppression, issue the **NOLOAD** command again.

PIN [K | [W] [R] [N] [H] [S] [E]]

PINMASK [K | [W] [R] [N] [H] [S] [E]]

Sets masked processor pins using the following arguments:

W disables the WAIT input pin.

R disables the RESET input pin.

N disables the NMI input pin.

H disables the HLDQR input pin.

S disables the HWSTOP output pin.

E disables the EMWAIT pin (valid only for V850E).

You can use the above arguments in combination to set multiple processor pin masks with one **PINMASK** command. Each argument should be delimited by a space.

Any new **PINMASK** settings override all previous settings.

PINMASK K unmask (kills) all masks.

If no arguments are specified, the **PINMASK** command displays the current masked pin settings.

Example:

```
brk>PINMASK w r
brk>PINMASK
Pin Cond
mask=WAIT RESET
brk>
```

POWER [on | off]

Connects or disconnects a target board from the ICE without powering down the system. After reconnecting the target board to the ICE, you must reestablish MULTI-to-ICE connections by executing your setup script (**.mbs**) file again. (For more information about setup scripts, see the documentation about configuring your target hardware in the *MULTI: Debugging* book.)

If no arguments are specified, the current **POWER** setting is displayed.

Example:

```
brk> POWER OFF
brk> POWER
Power is OFF
brk>
```

PROFILE data | done

PROFILE data gathers and accumulates information from the V850/V850 ICE trace buffer to the **850eserv2** internal profile data buffers.

PROFILE done writes the data to an output file readable by MULTI and clears the **850eserv2** internal profile data buffers. Therefore, after each **PROFILE data** command, you must reload the target program and repeat the process (that, you must, reissue the **PROFILE data** and **PROFILE done** commands) if you want to gather more profiling data.

To use the **PROFILE** command, you must open the MULTI Profile window before you download your target program. For further information, see the documentation about recording and viewing profiling data in the *MULTI: Debugging* book.

REG [regname [=value]]

Displays or changes register values. Without any parameters, this command will display the contents of all the registers along with the current program counter, program status word (PSW), and a disassembly of the next instruction to execute. If the processor is running and the program counter is in external memory, then the disassembly will not be performed to avoid memory conflicts.

The PSW is displayed as characters, either uppercase or lowercase, corresponding to the bits of the register. If the character is uppercase, the bit is set; if lowercase, the bit is clear.

If a register name is specified with the **REG** command, the current contents of the register will be displayed and you will be prompted to alter the value. Hit **Enter** if you do not want to change the value of the register. You can also use the **=value** argument to set a new value without being prompted.

For an example of how to use this command, see “Additional Examples of 850eserv2 Target Commands” on page 60.

RMEM [**K**] [**L=length**] [**T=interval**]

Displays the contents of real-time RAM periodically while a program is running, as specified by the following arguments:

L=length specifies the length of memory in bytes to be displayed. The value of *length* must be greater than or equal to 1 and less than or equal to 1024. The default is 512 bytes.

T=interval specifies an update interval in seconds. The default is three seconds.

K removes all existing settings.

If no arguments are specified, the **RMEM** command displays the current settings.

The base address of real-time RAM must be specified with the **RRAMBASE** command, before you can use the **RMEM** command.

When a program starts to execute after the **RMEM** command has been issued, the contents of the real-time RAM area are displayed in the target pane and are updated at the specified interval. When program execution breaks, updating stops. If you resume program execution, updating restarts unless **RMEM K** is entered.

If you specify a short period with the **T=** option, the display of real-time RAM contents may be updated too frequently. You can stop updating with the **K** option.

Example:

```
brk>rrambase 0x3ffe000
brk>rmem l=4 t=5
brk>
run>
0x3FFE000: 6F 80 00 04
0x3FFE000: 97 80 00 04
0x3FFE000: CB 80 00 04
brk>
```

RRAMBASE [*address*]

Sets *address* as the base address of the real-time RAM. If *address* is not specified, displays the current setting.

Because the size of the real-time RAM area is fixed to 1 KB, 1 KB of real-time RAM is allocated at this address. The address should be on a 1 KB boundary and should not be in internal ROM.

RST [stop | run]

Resets the CPU and/or emulation hardware.

If **stop** is specified (default option): After the reset, stops the CPU.

If **run** is specified: After the reset, runs the CPU freely.

SA

SHOWALL

Displays the current settings of programmed events, break settings, link settings, trace conditions, CPU settings, COMBO settings, NOHALT settings, and memory verify setting.

Example:

```
brk>showall
BRS1: A=0x00100082 P=0xXX
BRS2: A=0x001000B0 P=0xXX
BRA:
    <None>
Break on:
Link event:
    <None>
Trace start:
BRS1: A=0x00100082 P=0xXX
Trace end (delay = <none>)
BRS2: A=0x001000B0 P=0xXX
ROM size=32 RAM size=2
COMBO break is disabled
NoHalt:
    Disabled
Verify:
On
brk>
```

SFR [*name* [=*newvalue*]]

Displays or changes the SFR values.

If *name* is specified: Displays the SFR specified by *name* or, if [=*newvalue*] is also specified, changes the value of the SFR *name* to *newvalue*.

If no arguments are specified: Displays all SFRs.

The value of *name* comes from the V850/V850E assembly language specification.

If you enter only the name of an SFR, its value is displayed and you are prompted to enter a new value. Hit **Enter** if you do not want to change the SFR value.

For an example of how to use this command, see “Additional Examples of 850eserv2 Target Commands” on page 60.

T [A | *x=event_expression*. . .] [D=*delay*] [K]

TRACE [A | *x=event_expression*. . .] [D=*delay*] [K]

Specifies under what conditions tracing occurs. If no arguments are specified, the **T** command displays the current trace settings. Setting the **K** option clears (kills) all trace settings.

By default, the tracer will trace all frame transactions. You may reset the tracer to this default mode by entering **TRACE A**. You can also set the tracer to trace only at specified places in your code. You can start and stop the tracer with the **S** (start) and **E** (stop) arguments, enable or disable the tracer with the **Q** (qualify) argument, and/or halt the tracer after the specified delay time with the **T** (trigger) argument as follows:

- **S=*event_expression*** specifies the event(s) and/or link(s) that start the tracer.
- **E=*event_expression*** specifies the event(s) and/or link(s) that stop (end) the tracer.
- **Q=*event_expression*** specifies the event(s) and/or link(s) that enable or disable the tracer.
- **T=*event_expression*** specifies the trigger event(s) and/or link(s) that halt the tracer after a delay.
- **D=*delay*** specifies the number of frames to capture after the trigger. (A delay is also applied to events specified by **E**.)

If you use **S** and **E**, the **T** command automatically creates the LINK condition to perform section trace.

The value of *event_expression* can be one or more event and/or link mnemonic(s) separated by a pipe character (|). For example, **BRA1 | LINKA | BRS2**.

You can assign any or all of the events and/or the link specifications to the **T**, **S**, and **E** inputs. The start and stop inputs are edge-triggered, which means that once the event(s) occur, tracing will start or stop.

Q inputs may only be tied to events. The qualifying input is level-triggered, which means that tracing only occurs if the event(s) are true; once the event(s) are no longer true, tracing will cease. The start/stop and qualify conditions are ANDed together.

The **T** (trigger) option programs the tracer to stop after an event or link occurs. You can delay the halting of the tracer by specifying **D=*delay***. This specifies how long after a trigger event the tracer will halt, where *delay* is the number of frames you want to record after the trigger.

Once the tracer halts, the system mode switches from TRACE to RUN, at which point you can view the contents of the trace buffer, change trace settings, and issue other commands.

For an example of how to use this command, see “Additional Examples of 850eserv2 Target Commands” on page 60.

TD [I | F] [(\$|S|E|T) *rel frame*] [L=#frames]

TDISPLAY [I | F] [(\$|S|E|T) *rel frame*] [L=#frames]

Displays the trace buffer. If no arguments are specified, **TD** displays frames from the current frame. Use the optional parameters described below to set the display mode parameters, where:

I specifies Instruction mode.

F specifies Data Access (or Frame) mode.

\$ sets the starting frame of the display relative to the current frame.

S sets the starting frame of the display relative to the start of the buffer.

E sets the starting frame of the display relative to the end of the buffer.

T sets the starting frame of the display relative to the trigger point.

rel frame specifies the number of frames (+/-) relative to the starting frame

L=#frames sets the number of frames (max. 32768) to display.

There are two trace buffer display modes: Instruction mode (**I**) and Data Access (**F**) mode. In Instruction mode, a disassembly of executed instructions is displayed and no stall cycles are displayed. In Data Access mode, program data reads/writes are displayed. The default mode is Instruction mode. If no display mode is specified, the last mode specified is used.

When a new trace is made, the current frame is reset to the trigger point or to the end of the trace buffer if no trigger was specified. Each time a portion of the buffer is displayed, the current frame is updated to the end of the displayed data. If no starting frame number is specified, the current frame is used.

The default number of displayed frames is 20. You can change the default number with the **L** option. If you do not specify a number of frames, the last setting will be used. When displaying in Instruction mode, there will be fewer lines of disassembled code displayed than the number of frames, since instruction execution may span more than one frame for data access. Trace data can be displayed while a program is running, but you must first stop the tracer.

With the V850E ICE, two instructions can be displayed in one frame because the V850E can execute two instructions in one clock.

By default, the **Time** field of the trace display shows the number of CPU clocks that have elapsed from the previous frame to the current frame. If you specify **TMODE TS=T**, actual time is displayed instead of the number of clocks. If the elapsed time exceeds 24 hours, 24:00:00 is displayed. To get the proper time value, set the clock frequency and scalar value with the **TIMEBASE** command.

For an explanation of the codes that are used in the **Status** field of the trace display, see “Trace Display Status Codes” on page 65. For an example trace display, see the **TD** example in “Additional Examples of 850eserv2 Target Commands” on page 60.

TF [I= [T][O] | . | *] [F= [T][E][C][O][A][D][S][M] | . | *]

TFILTER [I= [T][O] | . | *] [F= [T][E][C][O][A][D][S][M] | . | *]

Specifies or displays what is filtered out before display in the trace buffer. This command affects only the trace display and does not affect the trace buffer. If no options are specified, **TF** displays the current filter settings.

Use the **I** or **F** options to specify which fields and bus status types are displayed, where the letters following the equal sign (=) indicate the fields and status types, as described below. Some fields are permanent and cannot be removed. Any previous setting of the **I=** or **F=** argument will be overwritten.

For Instruction mode, there are two possible fields that can be controlled with the **I=** argument. The **T** option specifies time-tag and the **O** option specifies opcode. The frame number, PC address, and mnemonic fields are permanent. To enable all of the Instruction mode fields, use **I=***. To disable all but the permanent fields, use **I=.**

For Data Access (or Frame) mode, there are eight possible fields that can be controlled with the **F=** command. The **T** option specifies time-tag, the **E** command specifies external probe data, the **C** option specifies program counter information, the **O** option specifies opcode, the **A** option specifies address, the **D** option specifies data, the **S** command specifies status, and the **M** command specifies mnemonic. The frame number field is permanent. To enable all of the Data Access mode fields, use **F=***. To disable all but the permanent fields, use **F=.**

For an example of how to use this command, see “Additional Examples of 850eserv2 Target Commands” on page 60.

TIMEBASE [C=*clock*] [R=*rate*]

Sets the frequency of the CPU clock and the scalar value of the trace time-tag counter. If no arguments are specified, the **TIMER** command displays the current settings.

The **C=** command specifies the frequency of CPU clock in MHz. For example, if the clock frequency of your CPU is 25 MHz, you can specify this with **C=25**.

The **R=** command specifies the scalar value of the trace time-tag counter, which counts the number of clocks for each trace frame. Because this counter is 16-bits long, the range of the counter is 0 to 65535.

If **R=X1** is specified, 1 CPU clock increases the counter by 1. If **R=X2** is specified, 2 CPU clocks increase the counter by 1. If **R=X1K** is specified, 1024 CPU clocks increase the counter by 1. In short, the larger the value of *rate*, the longer timer value the counter can hold. The counter holds at least 1 any time trace data is displayed, which can cause a loss of precision.

The following mnemonics can usually be specified as arguments to **R=**: X1, X2, X4, X8, X16, X32, X64, X128, X256, X512, X1K, X2K, X4K, X8K, X16K, X32K, X64K, X128K, X256K, X512K, X1M, X2M, X4M, X8M, X16M, X32M, X64M, X128M, X256M, X512M, X1G, and X2G, where K corresponds to 1024, M corresponds to 1024*1024, and G corresponds to 1024*1024*1024.

If you are using a V850 ICE, however, not all of these mnemonics are valid. Specifically, you cannot use the following values: X2K, X8K, X32K, X128K, X512K, X2M, X4M, X8M, X16M, X32M, X64M, X128M, X256M, X512M, X1G, and X2G.

By default, the CPU clock frequency is set to 25 MHz and the scalar value is set to X1.

Example:

```
brk>timebase c=33.33 r=x4
brk>timebase
CPU Clock=33.33MHz Timetag Rate=X4
brk>
```

TIMER

Displays the current timer value as a number of clocks.

This command takes no arguments. The timer is reset every time the target program execution is resumed by a call to `ex_run()`. When the target program breaks, the timer stops and holds the number of clocks elapsed during execution.

TMODE [M=F|N] [TS=C|T]

Specifies the tracing mode. The **M=** argument specifies the control mode for the trace memory and the **TS=** argument specifies the display format of the TIME field in **TDISPLAY**, as described below. If no arguments are specified, **TMODE** displays the current trace mode settings.

Specifying **M=N** sets trace memory to nonstop mode, which is the default mode. In nonstop mode, the tracer overwrites old trace data if the trace buffer becomes full and continues to trace until a **TSTOP** command is entered. Specifying **M=F**, sets trace memory to full stop mode. In full stop mode, the tracer stops tracing when the trace buffer becomes full.

If **TS=C** is specified, counts of the time-tag counter are displayed in the TIME field of **TDISPLAY**. If **TS=T** is specified, the time-tag is displayed in real-time units of microseconds, milliseconds, seconds or minutes.

Example

```
brk>tmode m=f
brk>
brk>tmode
Trace Mode:
  Trace Memory=Full stop
  Time Stamp=Count
brk>
```

TRUN

Restarts the trace analyzer if your program is running but the trace analyzer is stopped.

TS**TSTOP**

Halts the trace analyzer but keeps the CPU Evachip running. The emulator mode will switch from TRACE to RUN mode. Once in RUN mode, you can change the events, links, break, and trace settings, or display the contents of the trace buffer. Use **TRUN** to restart the analyzer.

TSEARCH [*frame*] [*A=address_or_range*] [*x=data/mask*] [*C=status*] [*P=mask4*]

Searches the trace buffer for a particular type of frame information and displays, in Frame mode, the frames that match the specified conditions. The search conditions consist of an address range, a data/mask value, a bus status type, and an external probe value, as described below. You can enter a subset of these conditions, in which case the other conditions will be ignored. If you do not specify any conditions, the last specified set of conditions will be used. The arguments for setting the conditions are described below.

- The *frame* argument specifies the starting frame for the search. It is the offset from the top (the oldest frame) of the trace buffer, not the frame number that is displayed with a **TDISPLAY** command. That is, frame number 0 specifies the oldest frame. If a frame number is not specified, the search begins from the frame next to the current trace pointer, not from the last position of the previous search. If a frame that matches the specified condition(s) is found, the current trace pointer is moved to it. The trace contents are displayed in both Instruction and Data Access mode and the search is stopped. If no frame matching the specified condition(s) is found, the trace pointer stays at the starting frame of the search.
- **A=address_or_range** specifies an address or range to search for, as follows: **A=address** specifies a single address expression, **A=start,end** specifies an address range beginning at *start* and ending at *end*, **A=start, L length** specifies an address range beginning at *start* and extending for *length* bytes.
- The *x=data/mask* option specifies the access size and data/mask values. The size of the access (*x*) can be **B** (byte), **H** (halfword), **W** (word), or **D** (any size, the default).
- The **C=status** option specifies the type of bus transaction to search for. Valid arguments for *status* are: **RW** (read or write; this is the default), **RO** (read only), or **WO** (write only).
- **P=mask4** specifies an external probe qualifier of 4-bit width. The *mask4* value is entered as a 4-bit hex or binary value/mask. Hex values are indicated by a 0x prefix, while binary input is indicated by a 0b prefix. To specify a value with a mask, use x characters for the nibbles or bits you want to be ignored.

For an example of how to use this command, see “Additional Examples of 850eserv2 Target Commands” on page 60.

U [*start* | *start, end* | *start, L lines*]

Disassembles object code beginning at the specified address.

If *start, end* are specified: Disassembles object code in the range beginning at *start* and ending at *end*.

If *start, L lines*: Disassembles object code in the range beginning at *start* and extending for *lines* lines.

If no address is specified: Disassembles code beginning where the last **U** command left off.

If no range is specified: Disassembly continues for 11 lines.

Example:

```
brk>U main
      main:
0x1000ba: 5cla add -4, sp
0x1000bc: 63ff0100 st.w lp, 0[sp]
0x1000c0: 2096e803 movea 0x3e8, zero, r18
0x1000c4: 64974180 st.w r18, -0x7fc0[gp]
0x1000c8: 40360200 movhi 0x2, zero, r6
0x1000cc: 2636a086 movea -0x7960, r6, r6
0x1000d0: bffffbcff jarl 0x10008c, lp
0x1000d4: 23ff0100 ld.w 0[sp], lp
0x1000d8: 441a add 4, sp
0x1000da: 7f00 jmp [lp]
0x1000dc: 0969 or r9, r13
brk>U main, L 5
      main:
0x1000ba: 5cla add -4, sp
0x1000bc: 63ff0100 st.w lp, 0[sp]
0x1000c0: 2096e803 movea 0x3e8, zero, r18
0x1000c4: 64974180 st.w r18, -0x7fc0[gp]
0x1000c8: 40360200 movhi 0x2, zero, r6
brk>
```

VERIFY [on | off]

Sets or clears the memory write verify flag, which turns on or off memory verify.

If **on** is specified, all subsequent memory writes are verified.

If **off** is specified, all subsequent writes are not verified.

If no parameters are set, the **VERIFY** command displays the current setting.

Example:

```
brk> VERIFY off
brk> VERIFY
VERIFY IS OFF
brk>
```

Additional Examples of 850eserv2 Target Commands

This section contains additional example of the target commands for **850eserv2**. For examples of other commands and for detailed descriptions of all the **850eserv2** commands and their parameters, see “Target Command Descriptions” on page 38. For a brief summary of all of the commands available, see “Target Commands for Renesas V850/V850E ICE (850eserv2) Connections” on page 35.

Example 3.6. Using the A Command

```
brk>a 0x100082
      main:
0x100082: 1a5c          add      -4, sp
              >add 8,sp
              481a
0x100084: ff630001      st.w     lp, 0[sp]
              >nop
              0000
0x100086: 0001          mov      r1, zero
              >nop
              0000
0x100088: 36400010      movhi   0x10, zero, r6
              >end
brk>a
0x100088: 36400010      movhi   0x10, zero, r6
              >end
brk>a main
      main:
0x100082: 1a48          add      8, sp
              >movhi hi(foo),zero,r6
              40361000
0x100086: 0000          nop
              >.
brk>a
0x100086: 0000          nop
              >st.w r18,-0x8000[zero]
              60970180
0x10008A: 0010          mov      r16, zero
              >.
brk>
```


Example 3.7. The BRA Command

```
brk>BRA
Bra:
    <None>
brk>BRA 1 H=0x00XX P=0x7
brk>BRA 2 A=data1, 0x1000de
brk>BRA
BRA1: H=0x00XX RW P=0x7
BRA2: A=data1 <0x001000ba>,0x1000de D=0XXXXXXXXX RW P=0xXX
brk>
```

Example 3.8. The LINK Command

```
brk>BRS 1 a=main
brk>BRA 2 a=data1
brk>BRS 3 a=data2
brk>BRS 4 a=data3
brk>LINK a 1=brs3|brs4 2=bra2 3=brs1
brk>LINK
LINKA: 1=BRS3, BRS4 2=BRA2 3=BRS1
PASS COUNT=1
AUTO RESET SELECTED
brk>LINK k
brk>LINK
Link:
    <None>
brk>
```

Example 3.9. The REG Command

```
brk>reg
    r0: 0x00000000    r8: 0x00000000    r16: 0x00000000    r24: 0x00000000
    r1: 0x0010A8E0    r9: 0x00000000    r17: 0x00000000    r25: 0x00000000
    r2: 0x00000000    r10: 0x00000000    r18: 0x00000009    r26: 0x00000100
    r3: 0x00120FD4    r11: 0x00000002    r19: 0xFFFFFFFF    r27: 0x00000102
    r4: 0x0010A8E0    r12: 0x00000001    r20: 0x000000FF    r28: 0x00120FF8
    r5: 0x00010000    r13: 0xFFFFC000    r21: 0x0000FFFF    r29: 0x00000000
    r6: 0x00000100    r14: 0xFFFFFFFF    r22: 0x00000000    r30: 0x00103F28
    r7: 0x00000102    r15: 0x00000000    r23: 0x00000000    r31: 0x00101F86
    pc: 0x00100088    eipc: 0x00100000    fepc: 0x00100000    ecr: 0x00000000
    psw: 0x00000028    eipsw: 0x00000061    fepsw: 0x000000FF
    psw:  neItCosZ    eipsw:  NEItcosZ    fepsw:  NEITCOSZ
0x100088: 36400010          movhi    0x10, zero, r6
brk>
```

Example 3.10. The SFR Command

```
brk>sfr
p0:RW      p1:RW      p2:RW      p3:RW      p4:RW      p5:RW      p6:RW
0x0B      0xDE      0x01      0xAC      0x18      0x20      0x10
p9:RW      p10:RW     pm0:RW     pm1:RW     pm2:RW     pm3:RW     pm4:RW
0xDD      0xFB      0xFF      0xFF      0xFF      0xFF      0xFF
pm5:RW     pm6:RW     pm9:RW     pm10:RW    pmc0:RW    pmc2:RW    pmc3:RW
0xFF      0xFF      0xFF      0xFF      0x00      0x01      0x00
mm:RW      pmc10:RW    dwc:RW      bcc:RW      psc:RW      sys:RW      unlock:RW
0xB7      0x00      0xFFFF      0xAAAA      0x00      0x00      0x00
brg0:RW     bprm0:RW    csim0:RW     csot0:RW    sio0:RW     brg1:RW     bprm1:RW
0x00      0x00      0x00      0x00      0x00      0x00      0x00
csim1:RW     csot1:RW    sio1:RW     csim2:RW    csot2:RW    sio2:RW     asim00:RW
0x00      0x00      0x44      0x00      0x00      0x40      0x80
asim01:RW    asis0:R      ove0:RW     fe0:RW      pe0:RW      sot0:RW     rxb0:R
0x00      0x00      0x00      0x00      0x00      0x00      0x01FF
rxb01:R      txs0:W      txs01:W     ovic1:RW    plic0:RW    plic1:RW    plic2:RW
0xFF      0x80CC     0xCE      0x47      0x47      0x47      0x47
plic3:RW     cmic4:RW     csic0:RW     seic0:RW    sric0:RW    stic0:RW    p0ic0:RW
0x47      0x47      0x47      0x47      0x47      0x47      0x47
p0ic1:RW     p0ic2:RW     p0ic3:RW     csic1:RW    csic2:RW    ispr:R      prcmd:W
0x47      0x47      0x47      0x47      0x47      0x00      0x00
intm0:RW     intm1:RW     intm2:RW     tovs:RW     tum1:RW     tmc1:RW     toc1:RW
0x00      0x00      0x00      0x00      0x0000     0x00      0x00
tml:R        cc10:RW      cc11:RW      cc12:RW     cc13:RW     tmc4:RW     tm4:R
0x0000     0xFFFF     0xFFFF     0xFFFF     0xFFFF     0x00      0x0000
cm4:RW
0x8000
brk>
```

Example 3.11. The T (or TRACE) Command

```
brk>trace a
brk>trace
Trace ALL
brk>trace k
Trace events are cleared.
brk>brs 1 a=main
brk>brs 2 a=exit
brk>trace s=brs1 e=brs2
brk>trace
Trace start:
BRS1: A=0x00100082 P=0xXX
Trace end (delay = <none>)
BRS2: A=0x001000B0 P=0xXX
brk>
```

Example 3.12. The TD (or TDISPLAY) Command

```
brk>tdisplay i s0 l=20
```

FRAME	TIME	ACCESS	PC	OPCODE	MNEMONIC
0	3		0x00100000	e200	mov 0, r28
1	14		0x00100002	ff800004	jarl 0x100006, lp
2	28		0x00100006	e81f	mov lp, r29
3	12		0x00100008	4808	mov r8, r9
4	0		0x0010000A	4007	mov r7, r8
5	12		0x0010000C	3806	mov r6, r7
6	13		0x0010000E	0261	cmp 1, zero
7	1		0x00100010	5200	mov 0, r10
8	0		0x00100012	36200014	movea 0x14, zero, r6
9	10		0x00100016	ff8027e4	jarl 0x1027fa, lp
10	2		0x001027FC	0000	nop
11	12		0x001027FE	007f	jmp [lp]
12	27		0x0010001A	0df9	bnc 0x100038
13	41		0x00100038	26044000	addi 0x4000, gp, gp
14	13		0x0010003C	26044000	addi 0x4000, gp, gp
15	13		0x00100040	2e054000	addi 0x4000, tp, tp
16	13		0x00100044	2e054000	addi 0x4000, tp, tp
17	13		0x00100048	0e400010	movhi 0x10, zero, r1
18	0		0x0010004C	0e013f28	addi 0x3f28, r1, r1
19	12		0x00100050	05a2	be 0x100054

```
brk>tfilter f=tcoms
```

```
brk>td f s0 l=5
```

FRAME	TIME	PC	OPCODE	MNEMONIC	EXT	STATUS
0	3	0x00100000	e200	mov 0, r28	0x0000	32b
1	14	0x00100002	ff800004	jarl 0x100006, lp	0x0000	32b
2	28	0x00100006	e81f	mov lp, r29	0x0000	32b
3	12	0x00100008	4808	mov r8, r9	0x0000	32b
4	0	0x0010000A	4007	mov r7, r8	0x0000	32b

```
brk>td i e=5
```

FRAME	TIME	ACCESS	PC	OPCODE	MNEMONIC
1160	1		0x00101F7E	381b	mov r27, r7
1161	0		0x00101F80	401d	mov r29, r8
1162	10		0x00101F82	ffbfel00	jarl 0x100082, lp
1163	41		0x00100082	1a5c	add -4, sp
1164	12	WD 0x00120FD4: 0x00101F86	0x00100084	ff630001	st.w lp, 0[sp]

```
brk>
```

For an explanation of the codes are used in the **Status** field of the trace display, see “Trace Display Status Codes” on page 65.

Example 3.13. The TF (or TFILTER) Command

```
brk>tfilter i=o
```

```
brk>tfilter
```

```
Trace Filter:
```

```
Instruction mode:
```

```
Frame: Permanent
```

```

Time:      Disabled
PC:        Permanent
Opcode:    Enabled
Mnemonic:  Permanent
Frame mode:
  Frame:    Permanent
  Time:     Enabled
  PC:       Enabled
  Opcode:   Enabled
  Address:  Enabled
  Data:     Enabled
  Mnemonic: Enabled
  Status:   Enabled
brk>td i s0 l=5
| ACCESS | PC | OPCODE | MNEMONIC |
-----|-----|
|         | 0x00100000 | e200 | mov 0, r28 |
|         | 0x00100002 | ff800004 | jarl 0x100006, lp |
|         | 0x00100006 | e81f | mov lp, r29 |
|         | 0x00100008 | 4808 | mov r8, r9 |
|         | 0x0010000A | 4007 | mov r7, r8 |
brk>tfilter f=ca
brk>td f s0 l=5
| FRAME | PC | ADDRESS | EXT |
-----|-----|
| 0 | 0x00100000 | | 0x0000 |
| 1 | 0x00100002 | | 0x0000 |
| 2 | 0x00100006 | | 0x0000 |
| 3 | 0x00100008 | | 0x0000 |
| 4 | 0x0010000A | | 0x0000 |
brk>tf f=m
brk>td f s0 l=5
| FRAME | MNEMONIC | EXT |
-----|-----|
| 0 | mov 0, r28 | 0x0000 |
| 1 | jarl 0x100006, lp | 0x0000 |
| 2 | mov lp, r29 | 0x0000 |
| 3 | mov r8, r9 | 0x0000 |
| 4 | mov r7, r8 | 0x0000 |
brk>tf f=ms
brk>td f s0 l=5
| FRAME | MNEMONIC | EXT | STATUS |
-----|-----|
| 0 | mov 0, r28 | 0x0000 | 32b |
| 1 | jarl 0x100006, lp | 0x0000 | 32b |
| 2 | mov lp, r29 | 0x0000 | 32b |
| 3 | mov r8, r9 | 0x0000 | 32b |
| 4 | mov r7, r8 | 0x0000 | 32b |
brk>

```

Example 3.14. The TSEARCH Command

```

brk>tf f=coadm
brk>tsearch 1100 a=0x120000,0x130000 c=rw
| FRAME | PC | OPCODE | ADDRESS | DATA | MNEMONIC |

```

```

-----
| 1103 | | | RD 0x00120F9C |0x00100462 |
| 1105 | 0x00100462 | e7230001 | RD 0x00120FA0 |0x00120FC8 | ld.w    0[sp], r28
| 1108 | | | RD 0x00120FA4 |0x001003AC |
| 1117 | 0x0010065C | ff630005 | WD 0x00120FA4 |0x001003BC | st.w    lp, 4[sp]
| 1118 | 0x00100660 | e7630001 | WD 0x00120FA0 |0x00120FC8 | st.w    r28, 0[sp]
| 1145 | 0x00100176 | ff630005 | WD 0x00120FA4 |0x0010042C | st.w    lp, 4[sp]
| 1146 | 0x0010017A | ef630001 | WD 0x00120FA0 |0x00120FD8 | st.w    r29, 0[sp]
brk>tsearch 1100 a=0x120000,0x130000 c=r
| FRAME | PC | OPCODE | ADDRESS | DATA | MNEMONIC
-----
| 1103 | | | RD 0x00120F9C |0x00100462 |
| 1105 | 0x00100462 | e7230001 | RD 0x00120FA0 |0x00120FC8 | ld.w    0[sp], r28
| 1108 | | | RD 0x00120FA4 |0x001003AC |
brk>

```

Trace Display Status Codes

The following codes are used in the **Status** field of the **TD** trace display:

16b	Bus access with 16-bit
32b	Bus access with 32-bit
cht	Cache hit
dly	Delay processing
int	Acknowledge interrupt
reti	Branch by <code>reti</code> instruction
sfm	Starting or ending frame of trace

Chapter 4

RTE (rteserv2) Connections

Contents

Supported Targets for RTE (rteserv2) Connections	68
Installing Your RTE	68
Connecting to a RTE Target	70
Commands for RTE (rteserv2) Connections	73

This chapter supplements the general target connection information contained in Chapter 2, “Setting Up Your Target Hardware” of this book and the documentation about connecting to your target in the *MULTI: Debugging* book with specific information for RTE connections.

Supported Targets for RTE (rteserv2) Connections

RTE stands for *Real-Time Evaluator* and refers to a group of hardware and software debugging interfaces available from Midas Labs. There are many different implementations of RTE.

MULTI supports debugging with an RTE through the Green Hills debug server **rteserv2**.

Installing Your RTE

To use MULTI with an RTE, you must first install the RTE hardware (if applicable) and software provided by Midas Labs. See your RTE documentation for detailed instructions on installing the RTE.

The Green Hills debug server **rteserv2**, which supports RTE connections, is installed automatically when you install MULTI.

Checking Your RTE Connection

After you have installed the RTE according to the documentation that accompanied it, you should check the connection between the RTE and your host. To check your connection, use the **Check RTE** program from within Windows. This RTE program allows you to:

- Set the parameters for your operating environment (for example, ports, baud, I/O address, etc.)
- Check the connection between the RTE and the host
- Test RTE functions

The **Check RTE** program will ask you to make a series of selections appropriate to your RTE. After you have made these selections, click the confirmation button. This will start a connection check and function test.

If the **Check RTE** program terminates normally and does not indicate any errors, you are ready to configure your MULTI files and then begin debugging with MULTI. For information about configuring MULTI for your specific debugging environment, see the documentation about configuring your target hardware in the *MULTI: Debugging* book.

If **Check RTE** does not terminate normally, the following checklist may help you locate the problem.

Type of connection	Things to check
Serial	• Serial cable connections to the RTE and the host PC • RTE settings (for example, baud, port, etc.)
Bus	• I/O address setting (The I/O address must not conflict with another board.) • Board connection • RTE dip switch settings (for example, baud, I/O address, etc.)

For more information about **Check RTE**, RTE settings, and troubleshooting your connection, see your RTE documentation.

Using the Green Hills SuperTrace Probe to Collect Real-time Trace Data

The following describes connecting the SuperTrace Probe to a Midas RTE Cube ICE, configuring the SuperTrace Probe, and collecting trace data. For more information about the SuperTrace Probe, see *Green Hills Debug Probes User's Guide*. For more information about connecting your debug server see “Connecting to a RTE Target” on page 70.

Before connecting a SuperTrace Probe, you must first be able to debug a program from MULTI with a Midas RTE Cube.

Connecting the SuperTrace Probe to a Midas RTE Cube

The target board must have a trace breakout board on it, or you must acquire a board separately. This trace breakout board passes the necessary debug signals through to the RTE Cube while providing an additional Mictor connector into which to plug the SuperTrace Probe.

The trace collection pod can be plugged directly into most versions of this breakout board. For breakout boards labeled ADP-NECM-STP (without any subsequent numbers), Green Hills adapter PATV8-01A must be used between the trace pod and the ADP-NECM-STP breakout board.

Use a telnet console, a serial console, or **gpadmin** to set the following settings:

- Adapter must be set to `v850e-trace-24`.
- Target must be set to `v850e_trace`.



Note

There is a hyphen in the adapter name and an underscore in the target name.

For more information about the SuperTrace Probe, see the documentation about installing your probe in the *Getting Started* book for your probe

Collecting Trace Data

To collect trace data from the target, you must specify the SuperTrace Probe in your **850eserv2** connection. For more information, see "**SuperTrace Probe is connected via**" in "Renesas V850/V850E ICE (850eserv2) Connection Settings" on page 26.

After the connection is made, verify that the target-specific trace options dialog is configured correctly for your target.

Connecting to a RTE Target

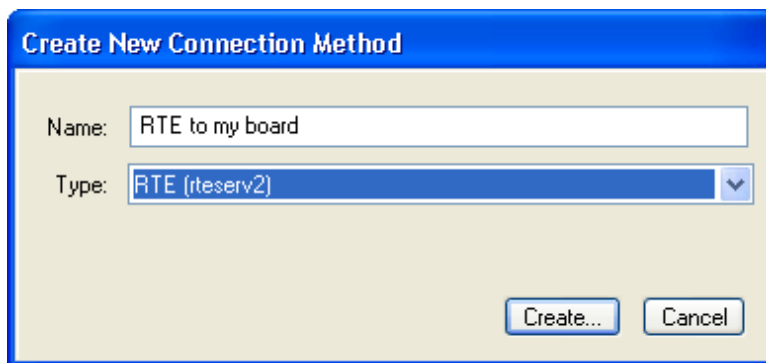
After you have installed and configured your system, you are ready to connect to your target and begin debugging. To help you connect to your target quickly and

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 the documentation about 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 RTE connections.

Creating a Standard RTE (rteserv2) Connection Method

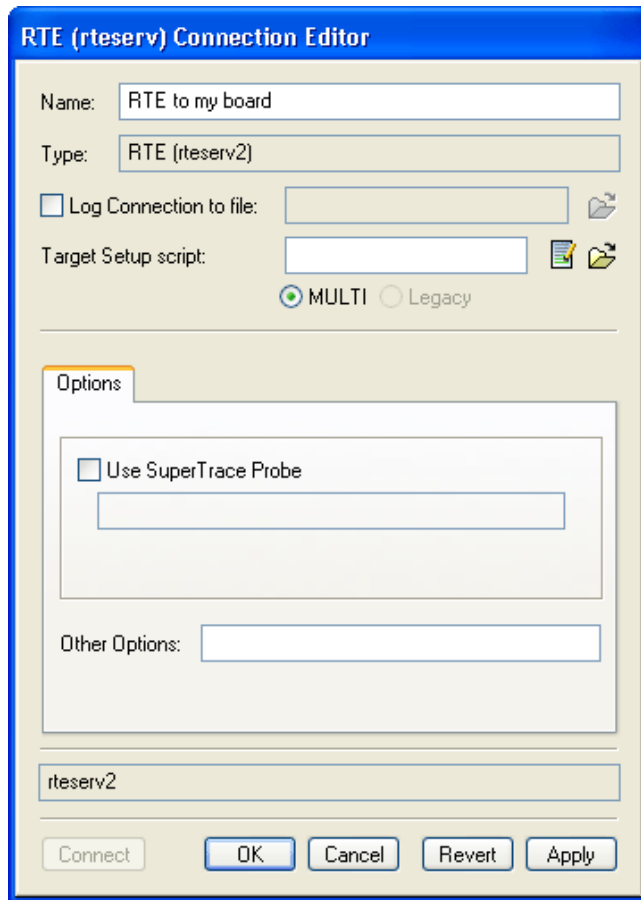
When creating a new Standard RTE Connection Method, select **RTE (rteserv2)** as the connection type in the **Create New Connection Method** dialog box.



For detailed instructions on creating new Standard Connection Methods, see the documentation about connecting to your target in the *MULTI: Debugging* book.

Using the RTE (rteserv2) Connection Editor

In addition to the generic fields that appear on all Connection Editors for Standard Connection Methods, the **RTE (rteserv2) Connection Editor** includes a **Debug** tab that allows you to set other options. However, you should not enter any other options on the **Debug** tab unless directed to do so by Green Hills Technical Support.



Log Connection to File

Controls logging of all communications for **rteserv2**. Select the radio button for your desired logging output. Logging is disabled by default.

Use SuperTrace Probe

The Debugger supports collecting trace data from hardware that supports this feature. You can collect trace data by using a Green Hills SuperTrace Probe in tandem with the **rteserv2** connection to the board.

By selecting this option, you can specify the network address of the SuperTrace Probe that **rteserv2** should connect to in order to obtain trace data. This trace data allows the user to use TimeMachine and other trace facilities of the MULTI Debugger (see the documentation about analyzing trace data with the TimeMachine tool suite in the *MULTI: Debugging* book).

For more information about configuring your SuperTrace Probe, see the documentation about configuring your probe in the *Getting Started* book for your probe.

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.

Commands for RTE (rteserv2) Connections

Unlike most Green Hills debug servers, **rteserv2** does not support all of the commands listed in “Generic Debug Server Commands” on page 90. The commands available to **rteserv2** are listed in the following sections.

General **rteserv2** Commands

The commands listed in this section are processed by **rteserv2** directly and work for all of the RTEs tested for this distribution of MULTI (see “Supported Targets for RTE (rteserv2) Connections” on page 68 for a list of tested RTEs). Each particular RTE also accepts other commands, which are listed in “Commands for Specific RTEs” on page 74.

You can enter the commands listed in this section directly into the **rteserv2 Target** pane. You can also enter these commands into the MULTI Debugger command pane using the **target** command. All of the commands for **rteserv2** are case-insensitive.

bpstat
Sends the eval command to MULTI to display the BP status.
dbgcomm [on off]
Turns on or off RTE data logging.
debug [on off]
Turns on or off the logging of debugging information.
info
Sends the eval command to MULTI to display the CPU status.
noload
Disables downloading to the target. Use this command to avoid duplicate downloading if, for example, an object already exists in memory.
tracewin
Displays the trace GUI.

Commands for Specific RTEs

In addition to the general commands described in “Commands for RTE (rteserv2) Connections” on page 73, each particular RTE also accepts other commands, which are listed below.

RTE-V821-PC

help <i>command</i> Displays help information. If a command is specified: Displays help information about the specified command. If a command is not specified: Displays a list of usable RTE commands and their formats.
inb [<i>address</i>] Reads (inputs) byte data from the I/O port and places it into <i>address</i> . If no address is specified, the previous address is used.
inh [<i>address</i>] Reads (inputs) halfword data from the I/O port and places it into <i>address</i> . If no address is specified, the previous address is used.
init Initializes the RTE-V821-PC. This command should rarely be used.
inw [<i>address</i>] Reads (inputs) word data from the I/O port and places it into <i>address</i> . If no address is specified, the previous address is used.
outb [<i>address</i>] <i>data</i> Writes (outputs) byte data at <i>address</i> to an I/O register. If <i>address</i> or <i>data</i> are not specified, the previous address and/or data are used.
outh [<i>address</i>] <i>data</i> Writes (outputs) halfword data at <i>address</i> to an I/O register. If <i>address</i> or <i>data</i> are not specified, the previous address and/or data are used.
outw [<i>address</i>] <i>data</i> Writes (outputs) word data at <i>address</i> to an I/O register. If <i>address</i> or <i>data</i> are not specified, the previous address and/or data are used.

sfr [registername [=data]]

Displays or sets the internal SFR registers.

If a register is specified, but data is not: Displays data read from the register.

If both the *registername* and *=data* arguments are specified: Writes the specified data to the specified register. The size of the data is determined automatically by the effective size of the specified register.

If no arguments are specified: lists the register names that may be used with the **sfr** command.

ver

Displays the version of the current RTE.

RTE-V851-IE

env

Sets the emulation CPU environmental values for the RTE-V851-IE.

help *command*

Displays help information.

If a command is specified: Displays help information about the specified command.

If a command is not specified: Displays a list of usable RTE commands and their formats.

init

Initializes the RTE-V851-IE. This command should rarely be used.

map

Specifies memory mapping.

nc

Specifies the excluded area for the RTE-V851-IE.

ncd

Deletes the excluded area for the RTE-V851-IE.

reset

Resets.

rrm

References memory within the real-time RAM monitor area.

rrmb

Specifies the real-time RAM monitor base address.

sfr [registername [=data]]

Displays or sets the internal SFR registers.

If a register is specified, but data is not: Displays data read from the register.

If both the *registername* and *=data* arguments are specified: Writes the specified data to the specified register. The size of the data is determined automatically by the effective size of the specified register.

If no arguments are specified: lists the register names that may be used with the **sfr** command.

timer

Measures the execution time.

ver

Displays the version of the current RTE.

Chapter 5

Multi-core Simulator for V850 and RH850 (simrh850) Connections

Contents

Installing the Multi-core Simulator for V850 and RH850 (simrh850)	78
Connecting to the Multi-Core Simulator for V850 and RH850 (simrh850)	78
Additional Commands for Multi-core Simulator for V850 and RH850 (simrh850) Connections	84
Profiling with simrh850	85
Simulation Modes	85
Unsupported Features	87
Connecting to the Multi-core Simulator for V850 and RH850 (simrh850) as a Stand-alone Debug Server	88

MULTI includes a target simulator called **simrh850** that allows you to execute and debug V850 and RH850 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 the documentation about connecting to your target in the *MULTI: Debugging* book with information about how to connect to and use the **simrh850** Simulator.

Installing the Multi-core Simulator for V850 and RH850 (simrh850)

The **simrh850** Simulator is installed automatically when you install MULTI. No other installation steps are necessary.

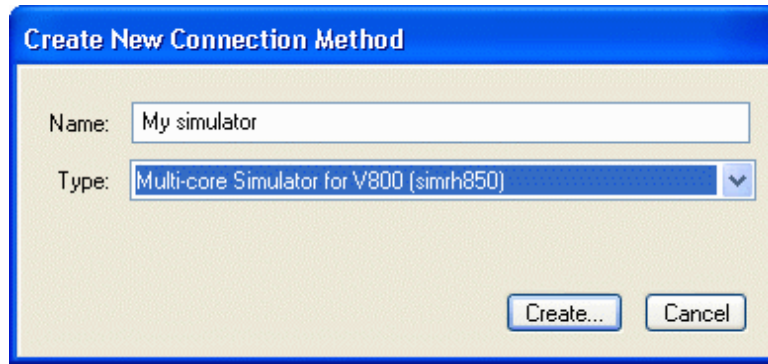
Connecting to the Multi-Core Simulator for V850 and RH850 (simrh850)

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 the documentation about 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 Multi-core Simulator for V850 and RH850 (**simrh850**) connections.

Creating a Multi-core Simulator for V850 and RH850 (simrh850) Connection Method

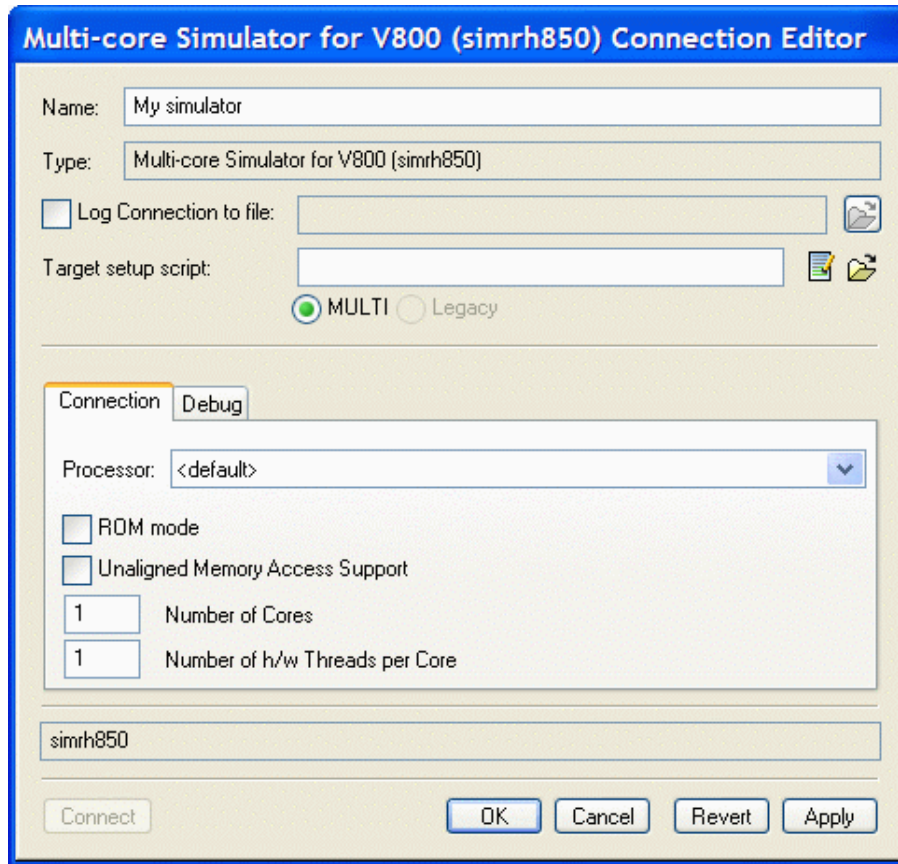
When creating a new Multi-Core Simulator for V850 and RH850 (**simrh850**) Connection Method, select **Simulator for RH850 (simrh850)** as the connection type in the **Create New Connection Method** dialog box.



For detailed information about creating new Standard Connection Methods, see the documentation about Standard Connection Methods in the *MULTI: Debugging* book.

Using the Multi-core Simulator for V850 and RH850 (simrh850) Connection Editor

In addition to the generic fields that appear on all Connection Editors for Standard Connection Methods, the **Multi-core Simulator for V850 and RH850 (simrh850) 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 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 can be used.

All of the fields on the **Connection** and **Debug** tabs of the **Multi-core Simulator for V850 and RH850 (simrh850) Connection Editor** are described in detail below. (See the documentation about the Connection Editor in the *MULTI: Configuring Connections* book for a description of the other fields and options, which appear on all **Connection Editors**.)

Multi-core Simulator for V850 and RH850 (simrh850) Connection Settings

Connection Debug

Processor: <default>

☐ ROM mode

☐ Unaligned Memory Access Support

1 Number of Cores

1 Number of h/w Threads per Core

Processor
Specifies which target processor to simulate. This field defaults to V850 .
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 85.
Unaligned Memory Access Support
Allows misaligned memory accesses on processors that support it.
Number of Cores
Selects the number of V850 and RH850 cores to simulate. Up to 32 cores may be simulated at one time. The default is 1.
Number of h/w Threads per Core
Selects the number of V850 and RH850 threads to simulate. Up to 8 threads may be simulated at one time. The default is 1. This option is only available for V850E3/RH850 cores.

Multi-core Simulator for V850 and RH850 (simrh850) Debug Settings

Connection Debug

Other Options:



Warning

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

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 Multi-core Simulator for V850 and RH850 (simrh850) Connection Methods

To connect to your **simrh850** Simulator 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**.

```
simrh850 [-cores=n] [-cpu=cpu] [-fpu=fpu] [-local_mem=start,length | start-end]  
[-no_auto_alloc] [-rh850_misalign] [-rh850_simd] [-rom[_use_entry]]  
[-shared_mem=start,length | start-end] [-stepcount=n] [-threads=n]  
[-trace_core=n] [-trace_thread=n]
```

where:

- **-cores=*n*** — Selects the number of CPU cores to simulate. Up to 32 cores may be simulated at one time. The default is 1.
- **-cpu=*cpu*** — Specifies which target processor to simulate. The default is `rh850g3m`. Other acceptable values for *cpu* are: `v850`, `v850e`, `v850e1f`, `v850e2`, `v850e2r`, `v850e2v3`, `v850e3v5`, `rh850g3k` and `rh850g3h`.
- **-fpu=*fpu*** — Specifies which FPU coprocessor to simulate. This option is only meaningful when simulating an RH850 family or V850E3 family processor. The default is `fpu30` which simulates a version 3 FPU. To simulate a version 2 FPU specify `fpu20`.
- **-local_mem=*start,length* | *start-end*** — Specifies a range of local memory to allocate for all cores and threads upon startup. This range of memory will be private for each cores and thread.
- **-no_auto_alloc** — Disables auto-allocation of simulated memory. By default, the simulator will automatically allocate memory to match the V850 and RH850 executable being simulated. When in multi-core or multi-threaded mode, all auto-allocated memory will be shared across all cores and threads. When this option is specified, only the memory configured with the **-shared_mem** and **-local_mem** options is allocated.

- **-nofastbps** — By default the simulator will simulate software breakpoints using the software breakpoint instruction. This option will cause it to switch to a method more similar to a hardware breakpoint. This will significantly slow down simulation, but is useful for debugging self-modifying software. It is intended for advanced users only.
- **-rh850_misalign** — Enables support for misaligned memory accesses in the simulator. This corresponds to the **-misalign_pack** option in the compiler.
- **-rh850_simd** — Enables support for the SIMD coprocessor available with some RH850 processors. The corresponding **-no_rh850_simd** option will force the SIMD coprocessor to be disabled.
- **-rom** — Enables ROM mode on the simulator and sets the simulator to start from the reset vector. To run the simulator in ROM mode, but start from the program's entry point, use **-rom_use_entry**. For more information, see “ROM Mode” on page 87.
- **-shared_mem=start,length | start-end** — Specifies a range of shared memory to allocate for all cores and threads upon startup. This range of memory will be shared across all cores and threads.
- **-stepcount=n** — Sets the number of instructions the simulator will simulate before polling the debugger for activity. The default is 100000 for a single-core, single-thread simulation. Otherwise the default is 1 to maximize synchronization between threads and cores. This option should only be used by advanced users to tweak performance and responsiveness trade-offs of the simulator.
- **-threads=n** — Selects the number of CPU threads to simulate. Up to 8 threads may be simulated at one time. The default is 1. This option is only available for V850E3/RH850 cores.
- **-trace_core=n** — Selects which core trace data will be collected on. Trace data may only be collected for a single core at a time. The default is core 0.
- **-trace_thread=n** — Selects which thread trace data will be collected on. Trace data may only be collected for a single thread at a time. The default is thread 0.



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 the documentation about Custom Connection Methods in the *MULTI: Debugging* book for more information about Custom Connections.

Additional Commands for Multi-core Simulator for V850 and RH850 (simrh850) Connections

The following commands, in addition to the commands listed in “Generic Debug Server Commands” on page 90, are available to **simrh850**.

You can enter all of these commands directly into the **simrh850 Target** pane. You can also enter these commands into the MULTI Debugger command pane using the **target** command. All of the commands for **simrh850** are case-insensitive.

alloc <i>[local shared] addr size</i>
Allocates <i>size</i> bytes of memory starting at address <i>addr</i> in memory.
freq <i>frequency</i>
Sets processor frequency to <i>frequency</i> MHz.
help
Displays all of the commands with their brief descriptions.
memmap
Displays a map of the currently allocated memory for all cores and threads.
snooze <i>interval</i>
Sets the V850E3/RH850 SNOOZE interval to <i>interval</i> cycles.
stepcount <i>count</i>
Sets the number of instructions the simulator will simulate before polling the debugger for activity. The default is 100000 for a single-core, single-thread simulation. Otherwise the default is 1 to maximize synchronization between threads and cores. This option should only be used by advanced users to tweak performance and responsiveness trade-offs of the simulator. Setting the value to 0 will reset to default values.

threads

Displays the V850E3/RH850 thread scheduling information. For example, the following output shows that the processor is configured for one CPU, where that CPU has six scheduling slots (from that CPU's HTSCCTL register's RND field), with the execution order (from that CPU's HTSCTBLn registers) being Thread 0, Thread 1, Thread 2, Thread 1, Thread 0 and finally Thread 2:

```
Thread schedule for core 0 (6 scheduling slots used):  
0 -> 1 -> 2 -> 1 -> 0 -> 2
```

Profiling with *simrh850*

You can use MULTI to profile with the **simrh850** simulator. Profiling with a simulator is often much more accurate than native profiling because profiling information can be obtained without periodically halting the program and recording the location of the program counter. The simulator keeps track of which instruction it is simulating and how many machine clocks have passed. The simulated clock rate is 50 MHz.

For more information about profiling, see the documentation about recording and viewing profiling data in the *MULTI: Debugging* book.

Simulation Modes

The **simrh850** 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.

The **simrh850** simulator is designed to be applicable for user level applications, but processor features such as caches, memory management unit, and any special peripherals and registers are generally not implemented.

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.

<code>access</code>	<code>alarm</code>	<code>brk</code>	<code>close</code>
<code>creat</code>	<code>exit</code>	<code>fstat</code>	<code>getpid</code>
<code>link</code>	<code>lseek</code>	<code>open</code>	<code>read</code>
<code>stat</code>	<code>time</code>	<code>unlink</code>	<code>write</code>



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.

Including a `MEMORY` directive in the link map does nothing unless a section is allocated at the address in question. For example:

```
MEMORY {  
    ...  
    memory : ORIGIN = 0x0080f000, LENGTH = 64K ...  
}  
...  
SECTIONS {
```

```
...  
.section : > memory  
...  
}*
```

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. 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 pass the **-rom_use_entry** option.

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.

Unsupported Features

The V850 does not support the following hardware features.

- Instruction pipelining
- Exact timing
- ICE mode
- Interrupts

Connecting to the Multi-core Simulator for V850 and RH850 (*simrh850*) as a Stand-alone Debug Server

To connect to the Multi-core Simulator for V850 and RH850 from the command line, independently of MULTI, enter:

simrh850 [-cpu=*cpu*] [-rom] *image_file* [*args*]...

where:

- -cpu=*cpu* — Specifies which target processor to simulate. The default is *rh850g3m*. Other acceptable values for *cpu* are: *v850*, *v850e*, *v850e1f*, *v850e2*, *v850e2r*, *v850e2v3*, *rh850g3h* and *rh850g3k*.
- -rom — Instructs the simulator to behave more like Green Hills hardware debug servers and to ignore the program's entry point. See “ROM Mode” on page 87.
- *image_file* — Is an ELF style executable file.
- *args* — Are optional command line arguments which are passed to the program in OS simulation mode. See “OS Simulation Mode” on page 85.

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

Contents

Green Hills Debug Server Commands	90
The Green Hills Debug Server Scripting Language	96

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.

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** pane. 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 0xffffffffffffffff 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:

```
address = (address & mask) | 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 <i>fd</i>
Closes the specified file descriptor, <i>fd</i> .
debug <i>n</i>
Sets the debugging bit flags.
Do not use this command unless directed to do so by Green Hills Technical Support.
delrange <i>addr</i>
Deletes a checked address range starting at <i>addr</i> . 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 <i>fd string</i>
Prints <i>string</i> to the specified file, <i>fd</i> , with script variable expansion.
fprintb <i>fd integer</i>
Prints <i>integer</i> to the specified file, <i>fd</i> , in binary mode.
fread <i>fd identifier</i>
Reads one line of text from the specified file, <i>fd</i> . The line is stored as a variable with the specified <i>identifier</i> . The number of bytes read is returned. If there is an error, -1 is returned.
freadb <i>fd identifier</i>
Reads an integer from the specified file, <i>fd</i> , in binary mode. The integer is stored as a variable with the specified <i>identifier</i> . The number of bytes read is returned. If there is an error, -1 is returned.
getenv <i>envName identifier</i>
Stores the value of the environment variable, <i>envName</i> , in the script variable with the specified <i>identifier</i> .
halt
Halts execution of the target CPU and forces it into debugging mode.

help [*command* | *group*]

(See also the device-specific **help** command listed in “Additional Debug Server Commands” on page 96.)

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>]
debug <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:

```
> 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:

```
> load all
Download Options: text data bss
```

Advanced Example:

```
> load all noload bss
Download Options: text data
```

m [-dsize] *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 MULTI register 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.
stepint [nomask mask] Turns interrupts during single stepping on (nomask) or off (mask). When set, stepint masks all interrupts during a single step. For example, on Power Architecture, interrupt masking is accomplished by masking the EE bit in the MSR. Stepping over code that alters the MSR with stepint on will result in unpredictable values in the MSR. If no argument is specified, this command displays the current value.
undef <i>variable</i> Removes <i>variable</i> and releases any memory associated with it. Example: <pre>> x=5 > undef x > print \$x Error: variable undefined!</pre>

Additional Debug Server Commands

The following table indicates where you can find information about additional commands (if any) available for your particular debugging interface.

Debugging interface	Where to find additional commands for this target
Green Hills Monitor	There are no additional commands for Green Hills Monitor connections.
Renesas V850/V850E ICE	“Target Commands for Renesas V850/V850E ICE (850eserv2) Connections” on page 35
RTE Servers	“Commands for Specific RTEs” on page 74
Simulator for V800	“Additional Commands for Multi-core Simulator for V850 and RH850 (simrh850) Connections” on page 84

The Green Hills Debug Server Scripting Language

In addition to the commands listed in “Generic Debug Server Commands” on page 90, a full scripting language is available from the **monserv** command line. This scripting language is described below.

You can run **monserv** 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 **monserv** 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 supported by **monserv**.

Expressions

Expressions in the Green Hills debug server scripting language are similar to C language expressions. Unlike C, each operator has its own precedence ($10/2*5$ evaluates to 1, not 25), and there are no unary operators (such as $\sim$ and unary $-$). 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.

Operator	Integer Function	String Function
()	Grouping of operators to ensure desired evaluation	Grouping of operators to ensure desired evaluation
*	Multiplication	Invalid
/	Division	Invalid
%	Modulus	Invalid
+	Addition	Concatenation
-	Subtraction	Invalid
<<	Bitwise left shift	Invalid
>>	Bitwise right shift	Invalid
<	Less than	Alphabetic less than
<=	Less than or equal to	Alphabetic less than or equal to
>	Greater than	Alphabetic greater than
>=	Greater than or equal to	Alphabetic greater than or equal to
==	Equality test	Equality test
!=	Inequality test	Inequality test
&	Bitwise AND	Invalid
^	Bitwise XOR	Invalid
	Bitwise OR	Invalid
&&	Logical AND	Invalid
	Logical OR	Invalid

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, or have the same name as a command.

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.

Syntax	Effect
if <i>expression</i> <i>statements</i> endif	If <i>expression</i> evaluates to zero, nothing happens. Otherwise, the block of statements between the if and endif lines are executed.
if <i>expression</i> <i>statements</i> else <i>statements</i> endif	If <i>expression</i> evaluates to zero, the debug server executes the block of statements between the else and endif lines. Otherwise, the block of statements between the if and else lines are executed.

Syntax	Effect
if <i>expression1</i> <i>statements</i>	If <i>expression1</i> does not evaluate to zero, the debug server executes the block of statements between the if and elif lines.
elif <i>expression2</i> <i>statements</i>	If <i>expression1</i> does evaluate to zero and <i>expression2</i> does not evaluate to zero, the debug server executes the block of statements between the elif and endif lines.
[elif <i>expressionX</i> <i>statements</i>]... endif	If both <i>expression1</i> and <i>expression2</i> 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. If none of the elif expressions evaluates to non-zero, the debug server will not execute anything.

Loops

The following table explains the syntax for loops.

Syntax	Effect
while <i>expression</i> <i>statements</i> endwhile	The statements between the while and endwhile lines are executed as long as <i>expression</i> does not evaluate to zero.
do <i>statements</i> dowhile <i>expression</i>	The statements between do and while are executed one time, then continue to execute as long as <i>expression</i> does not evaluate to zero.

Be careful to avoid infinite loops. If an infinite loop occurs, you must shut down and restart the debug server.

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. Integers are expanded

as two's complement signed 32-bit integers. 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. The hexadecimal string does not include a leading `0x`.

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:

```
>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.



Note

These example scripts will not work with Renesas V850/V850E ICE or RTE or connections.

Example A.1. Testing Script File Access in ASCII Mode

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**:

```
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 Linux/Solaris 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 A.2. Accessing a File

The following script is another example of accessing a file:

```
i=100
file = open temp.bin
while(i>0)
    fprintfb 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 A.3. Calculating a CRC32 Value

The following script calculates the CRC32 value of the memory range 0x010000 to 0x010100 and prints the result in the **Target** pane. 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 -dl $$ptr
    currbit=128
    while(currbit)
        test=r&(1<<31)
        r=r<<1
        r=r|(currbyte&currbit)
        if(test)
            r=r^p
        endif
        currbit=currbit>>1
    endwhile
    ptr=ptr+1
endwhile
# This loop is for the 32 zeros appended to the
# original memory contents
i=0
while(i<32)
    test=r&(1<<31)
    r=r<<1
    if(test)
        r=r^p
    endif
    i=i+1
endwhile
# Now the resulting 32 bit CRC is in r
# Many of the same ASCII control codes that are used
# in C are supported in debug server scripts.
endl="\n"
print CRC32 = $$r$endl
```


Appendix B

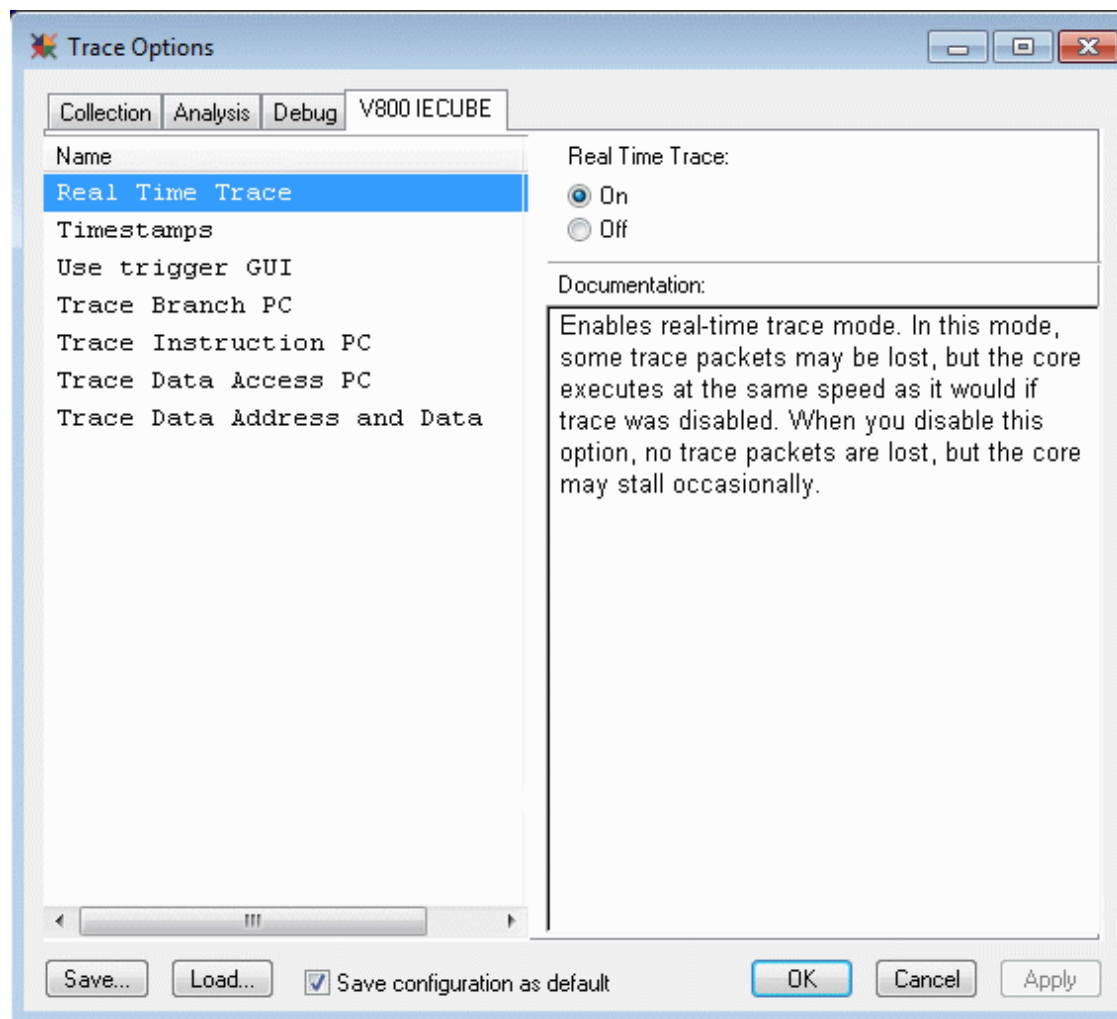
Target-Specific Trace Options and Triggers

Contents

Renesas V850E IECUBE Target-Specific Options	106
Renesas V850E2RV3 IECUBE2 Target-Specific Options	108
Renesas RH850 F1L Target-Specific Options	110
V850E/V850E2R RTE Target-Specific Options	112
V850 Target-Specific Trace Triggers	115

This section describes the target-specific trace configuration options and triggers available in the MULTI IDE. For information about generic trace options, see the *MULTI: Debugging* book.

Renesas V850E IECUBE Target-Specific Options



Each option in the **IECUBE Trace Options** window is described in the following table.

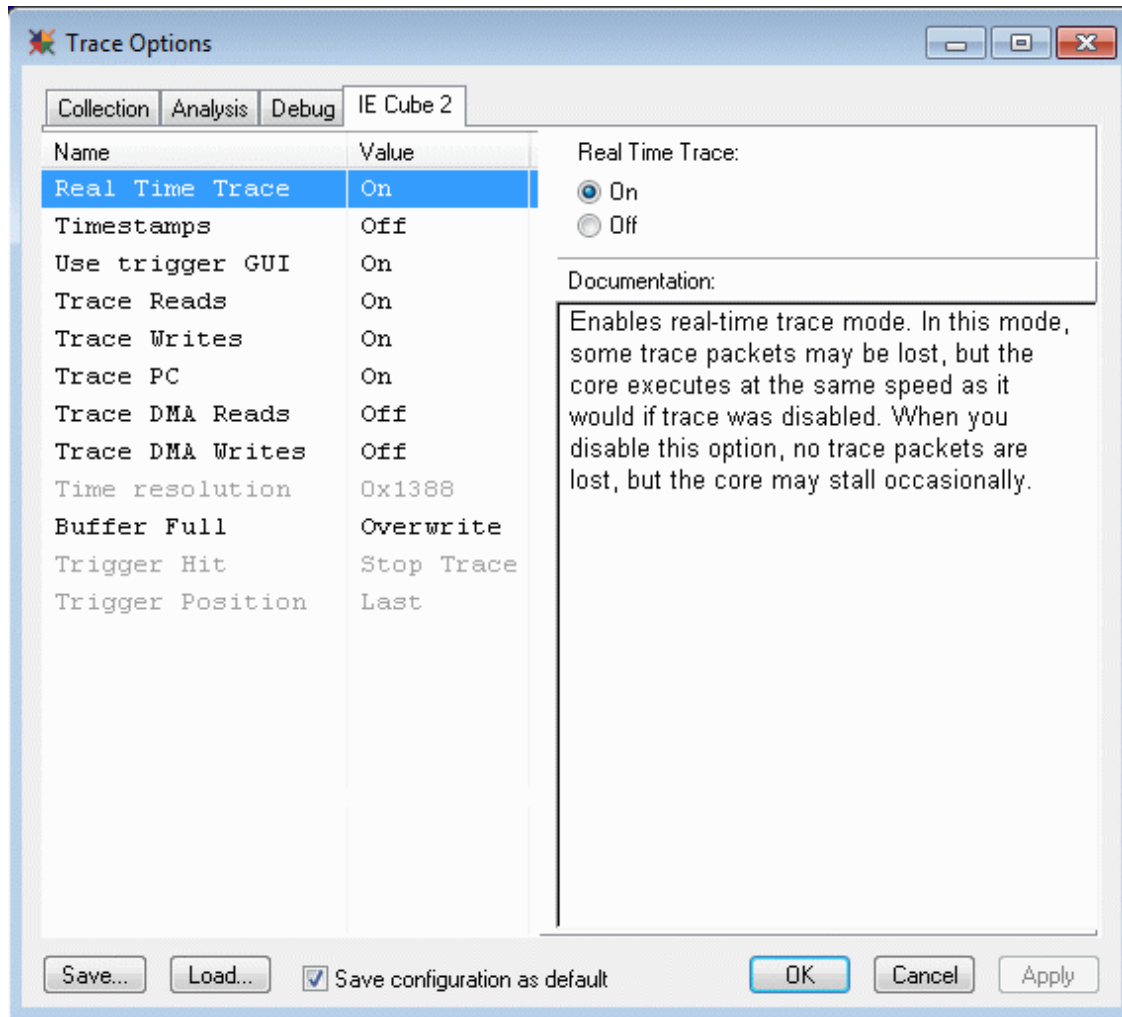
Option	Description
Real-Time Trace	Enables real-time trace mode. In this mode, some trace packets may be lost, but the core executes at the same speed as it would if trace was disabled. When you disable this option, no trace packets are lost, but the core may stall occasionally.
Timestamps	Enables timestamps. When timestamps are enabled, the trace collection device records a timestamp with each packet. Timestamps are displayed in the Trace List and are used by the MULTI Profile window, PathAnalyzer, and EventAnalyzer.
Use Trigger GUI	Enables the trigger GUI. For simple events, such as executing a function, the GUI is sufficient. However, the IECUBE also supports sophisticated event configuration that is not available via the trigger GUI. If you need access to more advanced features, you can clear this box and control all events by using debug server commands such as bra , brs , and trace .
Trace Branch PC	Enables branch program counter (PC) trace packets. This option is sufficient for fully reconstructing the instruction sequence the target executed. It is also sufficient for use with TimeMachine. For information, see the documentation about TimeMachine in the <i>MULTI: Debugging</i> book.
Trace Instruction PC	Enables instruction program counter (PC) trace packets, which results in the target outputting a trace packet for each instruction. This can yield more accurate timing information when timestamps are enabled. It also increases the probability that the trace buffer overflows in real-time trace mode.
Trace Data Access PC	Causes the target to output a program counter (PC) trace packet for every instruction that performs a memory access. This option is usually unnecessary.
Trace Data Address and Data	Enables data/address trace packets for memory accesses.



Note

Not all combinations of trace packets are valid. The user interface prevents you from selecting invalid combinations by dimming options that you cannot change given the current combination of settings.

Renesas V850E2RV3 IECUBE2 Target-Specific Options



Each option in the **IE Cube 2 Trace Options** window is described in the following table.

Option	Description
Real-Time Trace	Enables real-time trace mode. In this mode, some trace packets may be lost, but the core executes at the same speed as it would if trace was disabled. When you disable this option, no trace packets are lost, but the core may stall occasionally.
Timestamps	Enables timestamps. When timestamps are enabled, the trace collection device records a timestamp with each packet. Timestamps are displayed in the Trace List and are used by the MULTI Profile window, PathAnalyzer, and EventAnalyzer.

Option	Description
Use Trigger GUI	Enables the trigger GUI. For simple events, such as executing a function, the GUI is sufficient. However, the IECUBE2 also supports sophisticated event configuration that is not available via the trigger GUI. If you need access to more advanced features, you can clear this box and control all events by using debug server commands such as bra , brs , and trace .
Trace Reads	Enables data read trace packets for memory accesses.
Trace Writes	Enables data write trace packets for memory accesses.
Trace PC	Enables branch program counter (PC) trace packets. This option is sufficient for fully reconstructing the instruction sequence the target executed. It is also sufficient for use with TimeMachine. For information about TimeMachine, see the documentation about TimeMachine in the <i>MULTI: Debugging</i> book.
Trace DMA Reads	DMA read trace packets.
Trace DMA Writes	Enables DMA write trace packets.
Time resolution	Specifies the time resolution of trace packet timestamps. This field defaults to 5000 picoseconds when timestamps are enabled.
Buffer Full	Specifies the behavior when the trace buffer fills. You can choose to overwrite old trace data with new data, stop trace collection, or halt the process. If you stop trace collection or halt the process, note that this option can only preserve trace data acquired since the last time the target was halted, rather than all of the data acquired since the last time data was retrieved. When the Long Term Trace extension (QB-V850E2-SP trace memory extension) is used with IECUBE2, Stop Execution is not supported.
Trigger Hit	Specifies the behavior when a Delay Trace trigger is hit. It can be configured to stop trace collection or stop execution. This option is only available if a Delay Trace trigger has been set up.
Trigger Position	Specifies the desired position of the Delay Trace trigger in the trace buffer, after trace has stopped. Last means trace will stop very soon after hitting the trigger (so the trigger will end up being near to the last thing in the trace buffer). First means trace will continue to be collected until the trigger is near the beginning of the buffer. Middle means trace continues to be collected until half of the buffer has been filled. This option is only available if a Delay Trace trigger has been set up.



Note

Not all combinations of trace packets are valid. The user interface prevents you from selecting invalid combinations by dimming options that you cannot change given the current combination of settings.

Renesas RH850 F1L Target-Specific Options

Name	Value
Real Time Trace	On
Timestamps	Off
Use trigger GUI	On
Trace Reads	On
Trace Writes	On
Trace PC	On
Trace DMA Reads	Off
Trace DMA Writes	Off
Time resolution	0x1388
Buffer Full	Overwrite
Trigger Position	Last

Real Time Trace:

☒ On
☐ Off

Documentation:

Enables real-time trace mode. In this mode, some trace packets may be lost, but the core executes at the same speed as it would if trace was disabled. When you disable this option, no trace packets are lost, but the core may stall occasionally.

Save... Load... ☒ Save configuration as default OK Cancel Apply

Each option in the **RH850 Trace Options** window is described in the following table.

Option	Description
Real-Time Trace	Enables real-time trace mode. In this mode, some trace packets may be lost, but the core executes at the same speed as it would if trace was disabled. When you disable this option, no trace packets are lost, but the core may stall occasionally.
Timestamps	Enables timestamps. When timestamps are enabled, the trace collection device records a timestamp with each packet. Timestamps are displayed in the Trace List and are used by the MULTI Profile window, PathAnalyzer, and EventAnalyzer.
Use Trigger GUI	Enables the trigger GUI. For simple events, such as executing a function, the GUI is sufficient. However, the IECUBE2 also supports sophisticated event configuration that is not available via the trigger GUI. If you need access to more advanced features, you can clear this box and control all events by using debug server commands such as bra , brs , and trace .
Trace Reads	Enables data read trace packets for memory accesses.
Trace Writes	Enables data write trace packets for memory accesses.
Trace PC	Enables branch program counter (PC) trace packets. This option is sufficient for fully reconstructing the instruction sequence the target executed. It is also sufficient for use with TimeMachine. For information about TimeMachine, see the documentation about TimeMachine in the <i>MULTI: Debugging</i> book.
Trace Software	Causes the target to output a trace packet for each software trace instruction (DBCP, DBTAG, DBPUSH).
Trace Data Access PC	Causes the target to output a program counter (PC) trace packet for every instruction that performs a memory access. This option is usually unnecessary.
Trace DMA Reads	Enables DMA read trace packets.
Trace DMA Writes	Enables DMA write trace packets.
Time resolution	Specifies the time resolution of trace packet timestamps. This field defaults to 5000 picoseconds when timestamps are enabled.
Buffer Full	Specifies the behavior when the trace buffer fills. You can choose to overwrite old trace data with new data, stop trace collection, or halt the process. If you stop trace collection or halt the process, note that this option can only preserve trace data acquired since the last time the target was halted, rather than all of the data acquired since the last time data was retrieved.

Option	Description
Trigger Position	Specifies the desired position of the Delay Trace trigger in the trace buffer, after trace has stopped. Last means trace will stop very soon after hitting the trigger (so the trigger will end up being near to the last thing in the trace buffer). First means trace will continue to be collected until the trigger is near the beginning of the buffer. Middle means trace continues to be collected until half of the buffer has been filled. This option is only available if a Delay Trace trigger has been set up.



Note

Not all combinations of trace packets are valid. The user interface prevents you from selecting invalid combinations by dimming options that you cannot change given the current combination of settings.

V850E/V850E2R RTE Target-Specific Options

RTE Trace Options

Trace Stream Format

Board: Custom

Clock multiplier: ☐ 1/1 ☐ 1/2 ☒ 1/4

Data rate: ☒ SDR ☐ DDR

Number of data pins: ☒ 4 ☐ 8 ☐ 16 ☐ 24

Packet version: ☒ E1 ☐ E2

Packet Selection

Sub-switch on	Sub-switch off
☒ PC	☐ PC
☒ Data read	☐ Data read
☒ Data write	☐ Data write

Miscellaneous Options

☒ Use real-time trace mode

☒ Use timestamps

☒ Use trigger GUI

RTE Trace Options (V850E2R)

Trace Stream Format

Clock multiplier: ☐ 1/1 ☐ 1/2 ☐ 1/4 ☐ 1/6 ☐ 1/8 ☒ 1/16

Data rate: ☒ SDR ☐ DDR

Number of data pins: ☒ 4 ☐ 8 ☐ 24

Packet Selection

Sub-switch on	Sub-switch off
☒ PC	☐ PC
☒ Data read	☐ Data read
☒ Data write	☐ Data write
☒ DMA read	
☒ DMA write	

Miscellaneous Options

☐ Use real-time trace mode

☐ Use timestamps

☒ Use trigger GUI

Time resolution (ns):

Each option in the **RTE Trace Options** and **RTE Trace Options (V850E2R)** windows is described in the table below.

Option	Description
Board	Lists boards for which MULTI knows the trace stream format. Choose your board from the drop-down menu, and the relevant trace stream format options are selected for you in the RTE Trace Options window. If your board is not listed, select Custom. This option is only available in the RTE Trace Options window.
Clock multiplier	Controls the speed at which the target runs the trace clock. Options are listed as fractions of the CPU clock speed.
Data rate	Controls whether the target outputs trace data on a single clock edge (SDR) or on both clock edges (DDR).
Number of data pins	Specifies the number of trace data pins that the target uses to output trace data.
Packet version	Specifies the target's trace packet format version. This option is only available in the RTE Trace Options window.
Sub-switch on	Controls what kind of trace packets the target outputs when the sub-switch is on: • PC — Enables basic program counter (PC) trace packets for branches and jumps. • Data Read — Enables data/address trace packets for memory reads. • Data Write — Enables data/address trace packets for memory writes.
Sub-switch off	Controls what kind of trace packets the target outputs when the sub-switch is off: • PC — Enables basic program counter (PC) trace packets for branches and jumps. • Data Read — Enables data/address trace packets for memory reads. • Data Write — Enables data/address trace packets for memory writes.
DMA read	Enables DMA read trace packets. This option is only available in the RTE Trace Options (V850E2R) window.

Option	Description
DMA write	Enables DMA write trace packets. This option is only available in the RTE Trace Options (V850E2R) window.
Use real-time trace mode	Enables real-time trace mode. In this mode, some trace packets may be lost, but the core executes at the same speed as it would if trace was disabled. When you disable this option, no trace packets are lost, but the core may stall occasionally.
Use timestamps	Enables timestamps. When timestamps are enabled, the trace collection device records a timestamp with each packet. Timestamps are displayed in the Trace List and are used by the MULTI Profile window, PathAnalyzer, and EventAnalyzer.
Use trigger GUI	Enables the trigger GUI. For simple events, such as executing a function, the GUI is sufficient. However, the Midas ICE also supports some sophisticated event configuration that does not map well onto an abstract GUI. If you need access to more advanced features, you can clear this box and control all events by using debug server commands such as eve and evt .
Time resolution (ns)	Specifies the time resolution of trace packet timestamps. This option is only available in the RTE Trace Options (V850E2R) window.

The amount of data that the SuperTrace Probe can handle at one time is limited. Due to fast trace clock speeds, some V850E targets may exceed this data limit. The following two tables list the maximum trace clock speed supported for each combination of settings.

If **Use timestamps** is cleared (off):

# Data Pins	SDR Max Clock Speed	DDR Max Clock Speed
4	300 MHz	300 MHz
8	300 MHz	300 MHz
16	300 MHz	187 MHz
24	249 MHz	125 MHz

If **Use timestamps** is selected (on):

# Data Pins	SDR Max Clock Speed	DDR Max Clock Speed
4	266 MHz	266 MHz
8	249 MHz	187 MHz
16	187 MHz	125 MHz
24	249 MHz	125 MHz

V850 Target-Specific Trace Triggers

The following table lists V850-specific trace triggers:

Event	Description	Target Support
Section Sub-Switch On	Starts trace collection at the hardware level. When the conditions you set for this event are true, the target turns the section sub-switch on. By default, the target emits trace packets when the sub-switch is on.	V850E RTE
Section Sub-Switch Off	Stops trace collection at the hardware level. When the conditions you set for this event are true, the target turns the section sub-switch off. By default, the target does not emit trace packets when the sub-switch is off.	V850E RTE
Qualify Sub Switch	Filters out unwanted trace data at the hardware level. Trace data is only emitted if this event is active and if trace has been enabled by the Section Sub-Switch On event.	V850E RTE

Event	Description	Target Support
Start Trace	Starts trace collection at the hardware level. When the conditions you set for this event are true, the core starts emitting trace packets.	V850E IECUBE, V850E2RV3 IECUBE2
Stop Trace	Stops trace collection at the hardware level. When the conditions you set for this event are true, the core stops emitting trace packets.	V850E IECUBE, V850E2RV3 IECUBE2
Qualify Trace	Filters out unwanted trace data at the hardware level. Trace data is only emitted if this event is active and if trace has been enabled by the Start Trace event.	V850E IECUBE, V850E2RV3 IECUBE2
Delay Trace	Stops trace collection (and optionally stops execution) after the trigger is hit. The behavior of this trigger can be configured with the When Delay Trigger is Hit and Delay Trigger Position options.	V850E2RV3 IECUBE2

Index

Symbols

- 2m option
 - for 850eserv2, 30
- 850eserv2 debug server
 - additional setup and configuration
 - Windows, 20
 - commands (see commands, 850eserv2)
 - connecting with, 23
 - connection command syntax, 29
 - Connection Editor, 24
 - Custom Connection Method, 29
 - examples, 33
 - error messages, 34
 - installing, 20
 - options (see options, 850eserv2)
 - Standard Connection Method, 23
 - starting, 23
 - target system requirements, 20
 - troubleshooting, 33, 34
 - Windows definition files, 21
- 850win command
 - for 850eserv2, 37

A

- A command
 - for 850eserv2, 36, 39
 - example, 60
- addressof command
 - for legacy debug scripts, 90
- alloc command
 - for simrh850, 84
- amask command
 - for legacy debug scripts, 90
- arrays
 - for legacy debug scripts, 99
- assembling code, 39
- assignments
 - in legacy debug scripts, 98

B

- B command
 - for 850eserv2, 36, 39
- baud option
 - for monserv, 13
- baud rate, specifying
 - for Green Hills Monitor, 11, 13
- bpstat command
 - for dbgcomm, 73
 - for rteserv2, 73
- BRA command
 - for 850eserv2, 36, 40
 - example, 61
- breaks
 - setting, 39
- BRS command
 - for 850eserv2, 36, 41
- bss option
 - for monserv, 14
- .bss sections
 - legacy debug scripts, 93, 94
- bus event detectors
 - setting, 40

C

- cdnw option
 - for 850eserv2, 30
- checked address range
 - for legacy debug scripts, 91, 95
- CLOCK command
 - for 850eserv2, 38, 42
- close command
 - for legacy debug scripts, 91
- code assembly, 39
- COMBO breaks
 - displaying status, 42
- COMBO command
 - for 850eserv2, 37, 42
- commands
 - 850eserv2, 35, 38
 - 850win, 37
 - A, 36, 39
 - B, 36, 39
 - BRA, 36, 40
 - BRS, 36, 41
 - CLOCK, 38, 42
 - COMBO, 37, 42
 - CPU, 37, 43
 - data and register commands, 36
 - emulator configuration, 37

- event, trigger, and trace setting, 36
- execution commands, 36
- HELP, 38, 43
- IEPORT0, 38, 43
- IEPORT1, 38, 43
- LINK, 44
- LOG, 38, 45
- M, 36, 46
- MAP, 37, 47
- MODE, 38, 48
- NOHALT, 38, 48
- NOLOAD, 38, 48
- other, 37
- PIN, 37, 49
- PINMASK, 37, 49
- POWER, 38, 49
- PROFILE, 38, 50
- REG, 36, 50
- RMEM, 38, 51
- RRAMBASE, 38, 51
- RST, 36, 51
- SA, 36, 52
- SFR, 36, 52
- SHOWALL, 36, 52
- T, 36, 53
- TD, 37, 54
- TDISPLAY, 37, 54
- TF, 37, 55
- TFILTER, 37, 55
- TIMEBASE, 38, 56
- TIMER, 38, 56
- TMODE, 37, 57
- trace display, 37
- TRACE, 36, 53
- TRUN, 37, 57
- TS, 37, 57
- TSEARCH, 37, 58
- TSTOP, 37, 57
- U, 36, 59
- VERIFY, 38, 59
- for the Multi-core Simulator for V850 and RH850
 (simrh850), 84
- legacy debug scripts, 90
 - addressof, 90
 - amask, 90
 - close, 91
 - debug, 91
 - delrange, 91
 - delrangeall, 91
 - fprint, 91
 - fprintb, 91
 - fread, 91
 - freadb, 91
 - getenv, 91
 - halt, 91
 - help, 92
 - listrange, 92
 - listvars, 92
 - load, 93
 - m, 93
 - nofail, 93
 - noload, 94
 - open, 94
 - print, 94
 - random, 94
 - regnum, 94
 - rg, 94
 - run, 95
 - script, 95
 - setrange, 95
 - setup, 95, 97
 - sleep, 95
 - status, 95
 - step, 96
 - stepint, 96
 - undef, 96
- MULTI Debugger
 - target, 35, 38, 73, 84
- rteserv2, 73, 74, 75
 - bpstat, 73
 - dbgcomm, 73
 - debug, 73
 - env, 75
 - help, 74, 75
 - inb, 74
 - info, 73
 - inh, 74
 - init, 74, 75
 - inw, 74
 - map, 75
 - nc, 75
 - ncd, 75
 - nolaod, 73
 - outb, 74
 - outh, 74
 - outw, 74
 - reset, 75
 - rrm, 75
 - rrmb, 75
 - sfr, 75, 76
 - timer, 76
 - ver, 76

- simrh850
 - alloc, 84
 - freq, 84
 - help, 84
 - memmap, 84
 - snooze, 84
 - stepcount, 84
 - threads, 85
- conditionals
 - for legacy debug scripts, 99
- connecting MULTI to your target, viii
 - (see also Connection Editor)
 - (see also Connection Methods)
 - list of supported V850 and RH850 targets, viii
 - overview, viii
 - using a Green Hills Monitor, 8
- Connection Editor
 - Green Hills Monitor (monserv), 9, 10, 11, 12
 - Multi-core Simulator for V850 and RH850 (simrh850)
 - connection settings, 81
 - debug settings, 81
 - Renesas V850/V850E ICE (850eserv2), 24
 - RTE (rteserv2), 71
 - debug settings, 71
 - sections and fields of, 2
 - simrh850 Simulator, 79
- Connection Methods
 - Custom
 - Green Hills Monitor (monserv), 13, 14, 17
 - Renesas V850/V850E ICE (850eserv2), 29
 - simrh850 Simulator, 82
 - editing, 2
 - Standard, 2
 - Green Hills Monitor (monserv), 9, 10, 11, 12
 - Renesas V850/V850E ICE (850eserv2), 23
 - RTE (rteserv2), 70
 - V850 and RH850 Multi-core Simulator (simrh850), 78
- conventions
 - typographical, xi
- CPU
 - resetting, 51
- CPU command
 - for 850eserv2, 37, 43
- D**
 - D option
 - for 850eserv2, 30
 - data and register commands
 - 850eserv2, 36
 - .data sections
 - legacy debug scripts, 93, 94
 - dbreak option
 - for monserv, 14
 - .dbs setup scripts (see board setup scripts)
 - dck20 option
 - for 850eserv2, 31
 - debug command
 - for legacy debug scripts, 91
 - for rteserv2, 73
 - debug option
 - for monserv, 14, 15
 - debug server commands (see commands)
 - hpserv, 84
 - debug servers
 - logging connections, 3
 - monserv, 8
 - options (see options)
 - role of, viii
 - rteserv2, 68
 - debugging interfaces
 - list of supported, viii
 - debugging level, specifying
 - for Green Hills Monitor, 14, 15
 - definition files
 - for Renesas V850/850E ICE connections, 21
 - delrange command
 - for legacy debug scripts, 91
 - delrangeall command
 - for legacy debug scripts, 91
 - df=device_file option
 - for 850eserv2, 31
 - disassembling object code, 59
 - displaying, 43
 - displaying register values
 - changing, 50
 - displaying, 50
 - displaying settings, 52
 - document set, viii, x
 - download suppression, 48
- E**
 - eljtag option
 - for 850eserv2, 31
 - ellpd option
 - for 850eserv2, 31
 - elserial option
 - for 850eserv2, 31
 - e20jtag option
 - for 850eserv2, 31

- e20lpd option
 - for 850eserv2, 31
- e20serial option
 - for 850eserv2, 31
- echo command
 - for legacy debug scripts, 91
- emulator configuration commands, 37
- emulator memory configuration, 47
- env command
 - for rteserv2, 75
- env=env_vars option
 - for 850eserv2, 31
- error messages
 - 850eserv2 connections, 34
 - Renesas V850/V850E ICE connections, 34
- eval command, 73
- events
 - setting with 850eserv2, 36
- examples
 - Custom Connection Methods
 - Renesas V850/V850E ICE (850eserv2), 33
 - legacy debug scripts, 101
- execution commands, 36
- execution event detectors
 - setting, 41
- execution time, 76
- expressions
 - legacy debug scripts, 97

F

- fprint command
 - for legacy debug scripts, 91
- fprintb command
 - for legacy debug scripts, 91
- fread command
 - for legacy debug scripts, 91
- freadb command
 - for legacy debug scripts, 91
- freq command
 - for simrh850, 84

G

- getenv command
 - for legacy debug scripts, 91
- Green Hills Monitor, 8
 - (see also monserv debug server)
 - baud rate, specifying, 11, 13
 - connecting MULTI to, 8
 - Connection Editor, 9, 10, 11, 12
 - Connection Methods

- creating, 9
- Custom, 13, 14, 17
- editing, 9, 10, 11, 12
- debugging level, specifying, 14, 15
- halt signal, specifying, 12, 14, 15
- logging host-target communications, 13
- options (see options, monserv)
- overview, 8
- program sections, which to download, 12, 14, 15
- serial port, specifying, 11, 13

H

- halt command
 - for legacy debug scripts, 91
- halt signal, specifying
 - for Green Hills Monitor, 12, 14, 15
- hardware breakpoints, 33
- HELP command
 - for 850eserv2, 38, 43
- help command
 - for legacy debug scripts, 92
 - for rteserv2, 74, 75
 - for simrh850, 84

I

- I option
 - for 850eserv2, 31
- id key option
 - for 850eserv2, 31
- iecube option
 - for 850eserv2, 31
- IEPORT0 command
 - for 850eserv2, 38, 43
- IEPORT1 command
 - for 850eserv2, 38, 43
- inb command
 - for rteserv2, 74
- info command
 - for rteserv2, 73
- inh command
 - for rteserv2, 74
- init command
 - for rteserv2, 74, 75
- initializing the target, 74, 75
- installation
 - Renesas V850/V850E ICE, 20
 - RTE, 68
 - V850 and RH850 (simrh850) Multi-core Simulator, 78
- INTEGRITY Real-Time Operating System (RTOS), ix
- interrupts, enabling and disabling, 96

inw command
for rteserv2, 74

L

legacy debug scripts
arrays, 99
assignments, 98
.bss sections, 93, 94
conditionals, 99
.data sections, 93, 94
examples, 101
expressions, 97
loops, 100
registers, reading and writing, 94
running, 97
.text sections, 93, 94
variables
 declaring, 97
 expansion, 100
 type, 97
 writing, 97
LINK command
 for 850eserv2, 44
 example, 61
linking events with a sequencer, 44
Linux targets, ix
listrange command
 for legacy debug scripts, 92
listvars command
 for legacy debug scripts, 92
load command
 for legacy debug scripts, 93
-loadall option
 for monserv, 14
LOG command
 for 850eserv2, 38, 45
logging, 73
 debug server connections, 3
logging host-target communications
 for Green Hills Monitor, 13
loops
 for legacy debug scripts, 100

M

M command
 for 850eserv2, 36, 46
m command
 for legacy debug scripts, 93, 101
MAP command
 for 850eserv2, 37, 47

map command
 for rteserv2, 75
masked processor pins, 49
.mbs setup scripts (see board setup scripts)
memmap command
 for simrh850, 84
memory
 changing, 46
 displaying, 46
memory mapping, 75
-minicube option, 32
-minicube2 option
 for 850eserv2, 32
MODE command
 for 850eserv2, 38, 48
monitors, 8
 (see also Green Hills Monitor)
monserv debug server, 8
 (see also Green Hills Monitor)
 role of, 8
MULTI
 document set, x
 exiting without reset, 48
Multi-core Simulator for V850 and RH850 (simrh850)
 commands, 84
 simulation modes, 85

N

nc command
 for rteserv2, 75
ncd command
 for rteserv2, 75
-nobss option
 for monserv, 14
-nodata option
 for monserv, 14
nofail command
 for legacy debug scripts, 93
NOHALT command
 for 850eserv2, 38, 48
-noiop option
 for 850eserv2, 32
NOLOAD command
 for 850eserv2, 38, 48
noload command
 for legacy debug scripts, 94
 for rteserv2, 73
-noload option
 for monserv, 15
-notext option

for monserv, 15

O

open command

for legacy debug scripts, 94

options

850eserv2, 30

-2m, 30

-cdnw, 30

-D, 30

-dck20, 31

-df=device_file, 31

-eljtag, 31

-ellpd, 31

-elserial, 31

-e20jtag, 31

-e20lpd, 31

-e20serial, 31

-env=env_vars, 31

-I, 31

-id key, 31

-iecube, 31

-minicube2, 32

-noiop, 32

-p port, 32

-stp ip_address, 32

-t3v, 32

-t5v, 32

-usb, 32

-X0, 32

monserv

-baud, 13

-bss, 14

-dbreak, 14

-debug, 14, 15

-loadall, 14

-nobss, 14

-nodata, 14

-noload, 15

-notext, 15

-sp, 13

-zbreak, 15

OS targets, ix

OSE targets, ix

outb command

for rteserv2, 74

outh command

for rteserv2, 74

outw command

for rteserv2, 74

P

-p port option

for 850eserv2, 32

PIN command

for 850eserv2, 37, 49

PINMASK command

for 850eserv2, 37, 49

POWER command

for 850eserv2, 38, 49

print command

for legacy debug scripts, 94

PROFILE command

for 850eserv2, 38, 50

profiling

simulator frequency, 85

profiling support, 50

program sections

download, specifying which to

for Green Hills Monitor, 12, 14, 15

R

RAM, 43

random command

for legacy debug scripts, 94

REG command

for 850eserv2, 36, 50

example, 61

registers, reading and writing

legacy debug scripts, 94

regnum command

for legacy debug scripts, 94

Renesas RH850 FIL trace options, 110

Renesas V850/850E ICE

target system requirements, 20

Renesas V850/V850E ICE

additional setup and configuration

Windows, 20

commands, 38

data and register, 36

emulator configuration, 37

event, trigger, and trace setting, 36

execution, 36

other, 37

trace display, 37

connecting to, 23

Connection Editor, 24

Custom Connection Method, 29

examples, 33

error messages, 34

installing, 20

- options (see options, 850eserv2)
- Standard Connection Method, 23
- troubleshooting, 33, 34
- Renesas V850E IECUBE trace options, 106
- Renesas V850E2RV3 IECUBE2 trace options, 108
- reset command
 - for rteserv2, 75
- resetting the CPU, 51
- rg command
 - for legacy debug scripts, 94
- RMEM command
 - for 850eserv2, 38, 51
- ROM, 43
- rom option, 87
- rom_use_entry option, 87
- RRAMBASE command
 - for 850eserv2, 38, 51
- rrm command
 - for rteserv2, 75
- rrmb command
 - for rteserv2, 75
- RST command
 - for 850eserv2, 36, 51
- RTE
 - checking, 68
 - commands, 74, 75
 - commands for specific RTEs, 74
 - connecting to, 70
 - Connection Editor, 71
 - debug settings, 71
 - installing, 68
 - setting up, 68
 - Standard Connection Method, 70
 - troubleshooting, 69
- RTE (Real-Time Evaluator), 68
- RTE-V851-IE, 76
- rteserv2 debug server, 68
 - commands (see commands, rteserv2)
 - connecting with, 70
 - Connection Editor, 71
 - debug settings, 71
 - installing, 68
 - Standard Connection Method, 70
 - starting, 70
- run command
 - for legacy debug scripts, 95
- running legacy debug scripts, 97

S

- SA command

- for 850eserv2, 36, 52
- script command
 - for legacy debug scripts, 95, 97
- scripting language
 - monserv, 96, 97
- scripts, 4
 - (see also board setup scripts)
- serial port, specifying
 - for Green Hills Monitor, 11, 13
- setrange command
 - for legacy debug scripts, 95
- setup command
 - for legacy debug scripts, 95, 97
- setup option
 - for all debug servers, 97
- SFR command
 - for 850eserv2, 36, 52
 - example, 62
- sfr command
 - for rteserv2, 75, 76
- SFR values
 - displaying, 52
- SHOWALL command
 - for 850eserv2, 36, 52
- simrh850 Multi-core Simulator
 - connecting to, 78
 - installing, 78
 - Standard Connection Method, 78
- simrh850 Simulator
 - Connection Editor, 79
 - connection settings, 81
 - debug settings, 81
 - Custom Connection Method, 82
 - features, 78
 - simulated processors, 78
- simulated processors
 - with V850 and RH850 (simrh850) Multi-core Simulator, 78
- simulation modes, 85
- simulator
 - profiling frequency, 85
- simulators
 - connecting to
 - V850 and RH850 (simrh850) Multi-core Simulator, 78
 - features
 - simrh850, 78
 - installing
 - V850 and RH850 (simrh850), 78
 - supported by MULTI, viii
- sleep command

- for legacy debug scripts, 95
- snooze command
 - for simrh850, 84
- sp option
 - for monserv, 13
- Standard Connection Methods, 2
- status command
 - for legacy debug scripts, 95
- step command
 - for legacy debug scripts, 96
- stepcount command
 - for simrh850, 84
- stepint command
 - for legacy debug scripts, 96
- stp ip_address option
 - for 850eserv2, 32
- system requirements (see targets)

T

- T command
 - for 850eserv2, 36, 53
 - example, 62
- t3v option
 - for 850eserv2, 32
- t5v option
 - for 850eserv2, 32
- target command
 - for MULTI Debugger, 35, 38, 73, 84
- target connections
 - Renesas V850/V850E ICE, 23
 - RTE, 70
 - V850 and RH850 (simrh850) Multi-core Simulator, 78
- target system requirements
 - 850eserv2 debug server, 20
 - Renesas V850/V850E ICE, 20
- targets
 - Green Hills Monitor, 8
 - list of supported, viii
 - OS, ix
- TD command
 - for 850eserv2, 37, 54
 - example, 63
- TDISPLAY command
 - for 850eserv2, 37, 54
 - example, 63
- .text sections
 - legacy debug scripting, 93
 - legacy debug scripts, 94
- TF command
 - for 850eserv2, 37, 55
 - example, 63
- TFILTER command
 - for 850eserv2, 37, 55
 - example, 63
- threads command
 - for simrh850, 85
- ThreadX targets, ix
- TIMEBASE command
 - for 850eserv2, 38, 56
- TIMER command
 - for 850eserv2, 38, 56
- timer command
 - for rteserv2, 76
- TMODE command
 - for 850eserv2, 37, 57
- trace, 73
 - Renesas RH850 F1L trace options, 110
 - Renesas V850E IECUBE trace options, 106
 - Renesas V850E2RV3 IECUBE2 trace options, 108
 - setting with 850eserv2, 36
- targets
 - configuring, 106, 108, 110, 112
 - V850E RTE trace options, 112
 - V850E2R RTE trace options, 112
- trace analyzer, 52
 - halting, 57
 - restarting, 57
- trace buffer
 - displaying, 54
 - filtering, 55
 - tracing, 58
- TRACE command
 - for 850eserv2, 36, 53
 - example, 62
- trace commands
 - displaying, 55
- trace data
 - displaying, 54, 57
- trace display
 - commands, 37
 - status codes, 65
- tracing, 53
- troubleshooting
 - 850eserv2 connections, 33, 34
 - Renesas V850/V850E ICE connections, 33, 34
- TRUN command
 - for 850eserv2, 37, 57
- TS command
 - for 850eserv2, 37, 57
- TSEARCH command
 - for 850eserv2, 37, 58

TSERCH command

- for 850eserv2
- example, 64

TSTOP command

- for 850eserv2, 37, 57

typographical conventions, xi

U

U command

- for 850eserv2, 36, 59

undef command, 99

- for legacy debug scripts, 96

-usb option

- for 850eserv2, 32

V

V850 and RH850 (simrh850) Multi-core Simulator

- connecting to, 78

- features, 78

- installing, 78

- simulated processors, 78

V850E RTE trace options, 112

V850E2R RTE trace options, 112

ver command, 75

- for rteserv2, 76

VERIFY command, 59

- for 850eserv2, 38

version, 75, 76

W

Windows definition files, 21

writing legacy debug scripts, 97

X

-X0 option

- for 850eserv2, 32

Z

-zbreak option

- for monserv, 15

