RealView® Debugger

Version 4.1

RTOS Guide

RealView Debugger
RTOS Guide

Release Information

The following changes have been made to this document.

Change History

Date	Issue	Confidentiality	Change
March 2006	A	Non-Confidential	Release v3.0 for RealView Development Suite v3.0
March 2007	B	Non-Confidential	Release v3.1 for RealView Development Suite v3.1
September 2008	C	Non-Confidential	Release v4.0 for RealView Development Suite v4.0
27 March 2009	D	Non-Confidential	Release v4.0 SP1 for RealView Development Suite v4.0
28 May 2010	E	Non-Confidential	Release 4.1 for RealView Development Suite v4.1
30 September 2010	F	Non-Confidential	Release 4.1 SP1 for RealView Development Suite v4.1 SP1
31 May 2011	G	Non-Confidential	Release 4.1 SP2 for RealView Development Suite v4.1 SP2

Product Status

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

Web Address

`http://www.arm.com`

Contents
RealView Debugger RTOS Guide

Preface

This preface introduces the *RealView® Debugger RTOS Guide*. It contains the following sections:

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

About this book

This book describes how to debug embedded operating system (OS) applications with ARM® RealView Debugger, which requires additional software support from your OS vendor.

See the other books in the RealView Debugger documentation suite for more information about the debugger.

Intended audience

This book has been written for users who want to debug OS applications with RealView Debugger. It is assumed that users are experienced programmers, and have some experience with OS development.

Although prior experience of using RealView Debugger is not assumed, it is recommended that users first familiarize themselves with performing common debugging operations before using the OS-aware debugging features.

Using this book

This book is organized into the following chapters:

Chapter 1 ***OS Support in RealView Debugger***

Read this chapter for an introduction to the operating system support that is available in RealView Debugger.

Chapter 2 ***Configuring OS-aware Connections***

Read this chapter for details of how to use RealView Debugger OS features and configure an OS-aware connection.

Chapter 3 ***Connecting to a Target and Loading an Image***

Read this chapter for details of how to connect to your target and load an image on an OS-aware connection.

Chapter 4 ***Associating Threads with Views***

Read this chapter for details of how to work with threads in the RealView Debugger Code window.

Chapter 5 ***Working with OS-aware Connections in the Process Control view***

Read this chapter for details of how to use the Process Control view when working with OS-aware connections in RealView Debugger.

Chapter 6 ***Viewing OS Resources***

Read this chapter details of how to view the OS resources for an OS-aware connection.

Chapter 7 ***Debugging Your OS Application***

Read this chapter for a description of the features specific to debugging multithreaded images in RealView Debugger.

Typographical conventions

The typographical conventions are:

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

bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.
`monospace` (with `mono` underlined)	Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.
`monospace italic`	Denotes arguments to commands and functions where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
< and >	Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`

Further reading

This section lists publications by ARM and by third parties.

See also:

- Infocenter, `http://infocenter.arm.com` for access to ARM documentation.
- ARM web site , `http://www.arm.com` for current errata, addenda, and Frequently Asked Questions.
- ARM Glossary, `http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html`, for a list of terms and acronyms specific to ARM.

ARM publications

This book contains information that is specific to this product. See the following documents for other relevant information:

- *RealView Debugger Essentials Guide* (ARM DUI 0181)
- *RealView Debugger User Guide* (ARM DUI 0153).
- *RealView Debugger Target Configuration Guide* (ARM DUI 0182)
- *RealView Debugger Trace User Guide* (ARM DUI 0322)
- *RealView Debugger Command Line Reference Guide* (ARM DUI 0175).

For details on using the compilation tools, see the books in the ARM Compiler toolchain documentation.

For details on using RealView Instruction Set Simulator, see the following documentation for more information:

- *RealView Instruction Set Simulator User Guide* (ARM DUI 0207).

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

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

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

- *ARM DSTREAM Setting Up the Hardware* (ARM DUI 0481)
- *ARM DSTREAM System and Interface Design Reference* (ARM DUI 0499)

- *ARM DSTREAM and RVI Using the Debug Hardware Configuration Utilities* (ARM DUI 0498)
- *ARM RVI and RVT Setting Up the Hardware* (ARM DUI 0515)
- *ARM RVI and RVT System and Interface Design Reference* (ARM DUI 0517)

Other publications

For a comprehensive introduction to ARM architecture see:

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

For information on operating systems, see:

- *Operating Systems Design and Implementation* by Albert S. Woodhull, Andrew S. Tanenbaum -- (Hardcover - January 31, 2006).
- *Modern Operating Systems (International Edition)* by Andrew S. Tanenbaum -- (Hardcover - February 21, 2001).

Feedback

ARM welcomes feedback on both this product and its documentation.

Feedback on this product

If you have any problems with this product, submit a Software Problem Report:

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

Feedback on this book

If you have comments on content then send an e-mail to `errata@arm.com`. Give:

- the title
- the number, ARM DUI 0323G
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Chapter 1
OS Support in RealView Debugger

This chapter introduces the operating system support that is available in RealView® Debugger. It contains the following sections:

Note

OS support is currently only available for developers working on Windows workstations.

1.1 About OS support in RealView Debugger

RealView Debugger supports debugging of embedded operating system (OS) applications. An OS often uses separate software components to model and control the hardware with which it interacts. For example, a car engine controller might have two components to:

- model the motion of the cylinder, enabling it to control ignition and valve timing
- monitor fuel consumption and car speed and display trip distance and fuel economy on the dashboard.

Using components like this enables the OS to schedule tasks in the correct order to meet the specified deadlines. OS tasks can be:

Processes Created by the operating system, these contain information about program resources and execution state, for example program instructions, stack, and heap. Processes communicate using shared memory or tools such as queues, semaphores, or pipes.

Threads Running independently, perhaps as part of a process, these share resources but can be scheduled as tasks by the OS.

A thread can be controlled separately from a process because it maintains its own stack pointer, registers, and thread-specific data.

In single-processor systems, an OS controls one or more processes running on a single processor. Similarly, a process can have multiple threads, all sharing resources and all executing within the same address space. Because threads share resources, changes made by one thread to shared system resources are visible to all the other threads in the system.

In multiprocessor systems, specific processes and threads can be run on specific processors. For example:

- Processor 1 is dedicated to a specific task, such as car engine timing. This is a single process with no threads.
- Processor 2 has multiple tasks, such as displaying both the fuel economy and processing the average speed over a specified time. The developers implement these tasks as different processes, some of which have many threads.

——— Note ———

The OS support does not require a license from ARM® Limited, but you must obtain a plug-in for your chosen OS.

See also:

- *Debugging multithreaded application with RealView Debugger*.

1.1.1 Debugging multithreaded application with RealView Debugger

Debugging real-time systems presents a range of problems. This is especially true where the software being debugged interacts with physical hardware, because you normally cannot stop the hardware at the same time as the software. In some real-time systems, for example disk controllers, it might be impossible to stop the hardware.

Note

If you are debugging shared libraries created with the ARM linker, make sure that you have linked them with the `--dynamic_debug` option. This option generates dynamic relocations for debug sections.

See also

- *ARM® Compiler toolchain Using the Linker*
- *ARM® Compiler toolchain Linker Reference*.

1.2 Hardware for OS support

To debug an application using the OS-awareness features of RealView Debugger, the target processor must be supported by both RealView Debugger and your OS support package. See the documentation provided with your OS support package for details.

See also:

- *About OS support in RealView Debugger* on page 1-2.

1.3 OS debugging features available in RealView Debugger

The OS debugging features available in RealView Debugger enable you to:

- attach threads to Code windows, enabling you to monitor one or more threads in the system at the same time
- select individual threads to display the registers, variables, and code related to that thread
- change the register and variable values for individual threads
- display additional resource information.

——— **Note** ———

It is recommended that you read this chapter before attempting to debug an application using the OS-awareness features of RealView Debugger. This chapter includes debugging examples and assumes you have experience with using RealView Debugger for at least single-threaded programs.

———————————

See also:

- *Attaching and unattaching windows* on page 4-2
- Chapter 3 *Connecting to a Target and Loading an Image*.

1.4 Halted System Debug and Running System Debug

RealView Debugger supports different debugging modes, depending on the OS plug-in you are using:

Halted System Debug

Halted System Debug (HSD) means that you can only debug a target when it is not running. This means that you must stop your debug target before carrying out any analysis of your system.

HSD mode is not suitable for real-time systems where stopping the debug target might damage your hardware, for example disk controllers.

Running System Debug

Running System Debug (RSD) means that you can debug a target when it is running. This means that you do not have to stop your debug target before carrying out any analysis of your system. RSD gives access to the application using a *Debug Agent* (DA) that resides on the target and is typically considered to be part of the OS. The Debug Agent is scheduled along with other tasks in the system.

RSD is only available where supported by your debug target. It relies on having RealView Debugger OS extensions installed and is not provided as part of the base product.

RSD mode is intrusive because it uses resources on your debug target and makes demands on the application you are debugging. However, this debugging mode provides extra functionality not available when using HSD, for example, RSD enables you to debug threads individually or in groups, where supported by your OS and Debug Agent.

Where RSD is supported, RealView Debugger enables you to switch seamlessly between RSD and HSD mode using GUI controls or CLI commands.

See also:

- *Debug Agent* on page 1-7
- *OS-aware CLI commands* on page 1-9
- Chapter 5 *Working with OS-aware Connections in the Process Control view*
- Chapter 6 *Viewing OS Resources*.

1.5 Debug Agent

RSD requires the presence of a Debug Agent on the target to handle requests from the RealView Debugger host components. The Debug Agent is necessary so that the actions required by the host can coexist with the overall functioning of the target OS and the application environment. This relationship is shown in Figure 1-1.

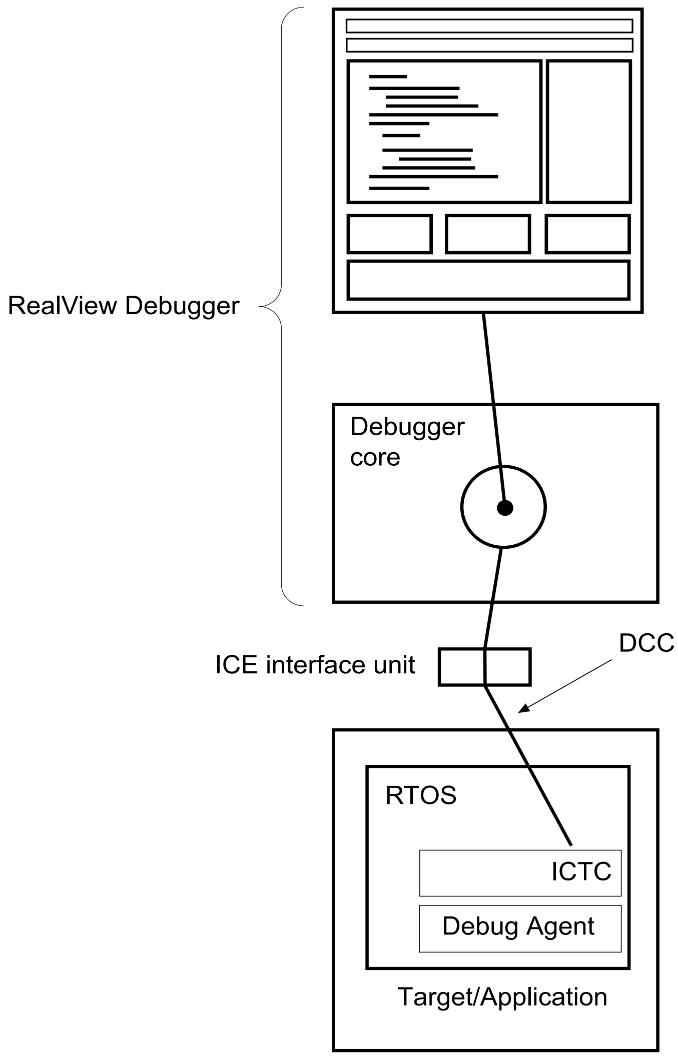


Figure 1-1 RealView Debugger and OS components

The Debug Agent and RealView Debugger communicate with each other using the *debug communications channel* (DCC). This enables data to be passed between the debugger and the target using an ARM DSTREAM™ debug and trace unit or RealView ICE debug unit, without stopping the program or entering debug state. The Debug Agent provides debug services for RealView Debugger and interacts with the OS and the application that is being debugged.

Note

A DCC device driver, the *IMP Comms Target Controller* (ICTC), is required to handle the communications between the Debug Agent and RealView Debugger. Depending on the OS, the ICTC is part of the Debug Agent or the OS.

By interacting with the OS running on the target, the Debug Agent can gather information about the system and make modifications when requested by the user, for example to suspend a specified thread.

In summary, the Debug Agent:

- provides a direct communications channel between the OS and RealView Debugger, using the ICTC
- manages the list of threads on the system
- enables thread execution control
- manages OS objects such as semaphores, timers, and queues
- accesses OS data structures during RSD mode.

1.6 OS-aware CLI commands

The following CLI commands are specific to OS-aware debugging:

- `AOS_`*`resource`*`_list` (OS action commands)
- `DOS_`*`resource`*`_list` (OS resource commands)
- `OSCTRL`
- `THREAD`.

The following CLI commands provide options that are specific to OS-aware debugging, or have a specific effect on the behavior of OS-aware connections:

- `BREAKINSTRUCTION`
- `HALT`
- `RESET`
- `STOP`.

Other commands can be used with OS-aware connections, such as those for stepping, accessing memory and registers, and setting hardware breakpoints.

See also:

- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 2 *RealView Debugger Commands*.

Chapter 2
Configuring OS-aware Connections

This chapter describes how to use RealView® Debugger operating system (OS) features and configure an OS-aware connection. It includes:

2.1 Enabling OS support

Your OS vendor supplies plug-ins to enable OS-awareness in RealView Debugger:

1. Download the plug-ins for your particular OS. To do this:
   a. Select **Help → ARM on the Web → Goto RTOS Awareness Downloads** from the Code window main menu to open the *OS Aware Debugger* page in your web browser.
   b. Locate your required OS, and download the plug-in from the OS vendor web page.
2. Install the plug-ins in your root installation. To do this, copy the `*.dll` file into the directory:

   `install_directory``\RVD\Core\...\lib`

See also:

- *Creating a new OS-aware Debug Configuration* on page 2-3.

2.2 Creating a new OS-aware Debug Configuration

It is recommended that you create a new Debug Configuration to set up OS-aware connections to your target. Although this is not required, it means that it is easy to identify the OS-aware connections, and to maintain other custom Debug Configurations. This section describes how to set up the new Debug Configuration.

This example defines a new `RealView ICE` Debug Configuration. You can do this by:

- creating a new Debug Configuration
- copying a Debug Configuration that already exists for your development platform (the method used in this example).

The example assumes that you have configured the default `RealView-ICE` Debug Configuration for your development platform.

See also:
- *Procedure*
- *Configuring a DSTREAM or RealView ICE interface unit* on page 2-4.

2.2.1 Procedure

To set up the new OS-aware Debug Configuration for your development platform:

1. Start RealView Debugger.
2. Select **Target** → **Connect to Target...** from the Code window main menu to open the Connect to Target window. Figure 2-1 shows an example:

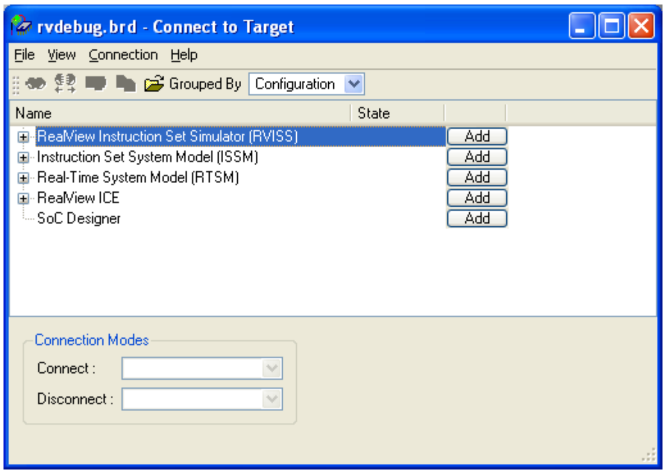

rvdebug.brd - Connect to Target
File View Connection Help
Grouped By Configuration
Name State
RealView Instruction Set Simulator (RVISS) Add
Instruction Set System Model (ISSM) Add
Real-Time System Model (RTSM) Add
RealView ICE Add
SoC Designer Add
Connection Modes
Connect :
Disconnect :

Figure 2-1 Connect to Target window

The contents of this window depend on the installed Debug Interfaces available to you.

3. Select **Configuration** from the Grouped By list.
4. Expand the required Debug Interface that you are using to access your development platform. For this example, expand `RealView ICE`.
5. Expand the required Debug Configuration that you want to use as the basis for your new configuration. For this example, expand `RealView-ICE`.
6. Right-click on the Debug Configuration to display the context menu. For this example, right-click on the `RealView-ICE` Debug Configuration.

7. Select **Copy Configuration...** from the context menu. A new Debug Configuration is created, called `Copy_of_RealView-ICE`.

8. Change the name of the new Debug Configuration to something suitable:

   a. Right-click on the Debug Configuration to display the context menu.

   b. Select **Rename Configuration...** from the context menu.

   c. Enter the required name, for example **RVI_OS_tst**.

   Your new `RealView ICE` Debug Configuration is renamed, as shown in Figure 2-2.

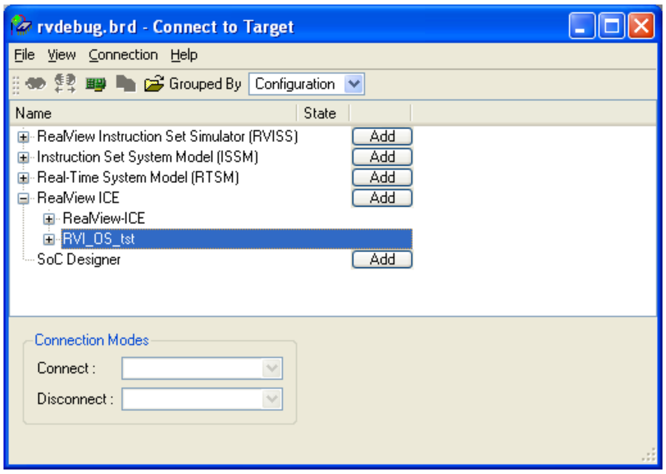

rvdebug.brd - Connect to Target
File View Connection Help
Grouped By Configuration
Name
State
RealView Instruction Set Simulator (RVISS)
Add
Instruction Set System Model (ISSM)
Add
Real-Time System Model (RTSM)
Add
RealView ICE
Add
RealView-ICE
RVI_OS_tst
SoC Designer
Add
Connection Modes
Connect :
Disconnect :

Figure 2-2 New Debug Configuration in the Connect to Target window

9. Configure OS support for the new Debug Configuration as described in your OS documentation, coupled with the information in *Customizing an OS-aware Debug Configuration* on page 2-6.

See also

- *Customizing an OS-aware Debug Configuration* on page 2-6
- the following in the *RealView Debugger User Guide*:
  — *About creating a Debug Configuration* on page 3-8
- the following in the *RealView Debugger Target Configuration Guide*:
  — *About customizing a DSTREAM or RealView ICE Debug Interface configuration* on page 2-3.

2.2.2 Configuring a DSTREAM or RealView ICE interface unit

Remember the following when specifying settings for your hardware:

- Autoconfiguring the DSTREAM or RealView ICE unit does have side-effects and might be intrusive. Where this is not acceptable, you must configure manually.
- Be aware that clicking the option **Reset on Connect** might interfere with the initialization sequence of your application or target hardware.
- The DSTREAM or RealView ICE scan chain configuration lists devices in ascending order of TAP ID.

See also

- *Customizing an OS-aware Debug Configuration* on page 2-6

- the following in the *RealView Debugger User Guide*:
  - *About creating a Debug Configuration* on page 3-8
- the following in the *RealView Debugger Target Configuration Guide*:
  - Chapter 2 *Customizing a Debug Interface configuration*.

2.3 Customizing an OS-aware Debug Configuration

For ARM architecture-based targets, OS awareness is controlled by the following settings groups in the `Advanced_Information` block:

- `RTOS_config`
- `ARM_config`.

Note

To avoid any conflicts with settings, you must configure these settings in a Debug Configuration, not in a BCD file.

See also:

- *OS configuration settings*
- *Locating the OS related settings groups* on page 2-7
- *Procedure for configuring RTOS settings* on page 2-8
- *Procedure for configuring the Basic Settings* on page 2-9
- *Procedure for configuring the connect and disconnect settings* on page 2-9.

2.3.1 OS configuration settings

You configure OS operation using the following `RTOS_config` group settings of a Debug Configuration:

Events Identifies which events to capture. Not supported in this release.

Vendor This three letter value identifies the OS plug-in, that is the `*.dll` file supplied by your vendor.

Load_when Defines when RealView Debugger loads the OS plug-in:

- Select **connect** to load the plug-in on connection.
- Select **image_load** to load the plug-in when an OS image is loaded.

The OS features of the debugger are not enabled until the plug-in is loaded and has found the OS on your target.

When the plug-in is loaded, it immediately checks for the presence of the OS. If loaded, the plug-in also checks when you load an image. This means that you might have to run the image startup code to enable OS features in the debugger.

Base_address

Defines a base address, overriding the default address used to locate the OS data structures. See your OS documentation for details.

Cpu_id Specifies a CPU identifier so that you can associate the OS-aware connection to a specific CPU. See your OS documentation for details.

Exit_Options

Defines how OS awareness is disabled. Use the context menu to specify the action to take when an image is unloaded or when you disconnect. You can also specify a prompt.

RSD Controls whether RealView Debugger enables or disables RSD. This setting is only relevant if your debug target can support RSD.

If you load an image that can be debugged using RSD, you might want to set this to **Disable** to prevent RSD starting automatically. You can then enable RSD from the Process Control view.

System_Stop

Use this setting to specify how RealView Debugger responds to a processor stop request when running in RSD mode.

In some cases, it is important that the processor does not stop. This setting enables you to specify this behavior, use:

- **Never** to disable all actions that might stop the processor. That is:
  - the `STOP` command is not actioned
  - the **Stop Target** option on the Process Control view context menu is disabled.
- **Prompt** to request confirmation before stopping the processor.
- **Don't_prompt** to stop the processor. This is the default.

Event_capture

Enables the capturing of events. Not supported in this release.

See also

- *Context menu* on page 5-6
- the following in the *RealView Debugger Command Line Interface Guide*:
  - *Alphabetical command reference* on page 2-12.

2.3.2 Locating the OS related settings groups

To locate the OS related settings groups:

1. Select **Connect to Target...** from the Code window **Target** menu to open the Connect to Target window.
2. Select **Configuration** from the Grouped By list.
3. Expand the required Debug Interface that you are using to access your development platform. For this example, expand `RealView ICE`.
4. Disconnect all target connections for the required Debug Configuration, if any.

   Note

   You cannot configure a Debug Configuration when the debugger is connected to a target in that Debug Configuration.

5. Right-click on the required Debug Configuration to display the context menu. For this example right-click on the `RVI_OS_tst` Debug Configuration.
6. Select **Properties...** from the context menu to open the Connection Properties dialog box.
7. Click **RTOS** to display the commonly used RTOS settings.
8. Configure the commonly used RTOS settings and Basic Settings as required.
9. Click the **OK** button to close the Connection Properties dialog box.

See also

- *Creating a new OS-aware Debug Configuration* on page 2-3.

2.3.3 Procedure for configuring RTOS settings

To configure RTOS settings for the new connection:

1. Select **Target** → **Connect to Target...** from the Code window main menu to open the Connect to Target window. Figure 2-2 on page 2-4 shows an example.
2. Select **Configuration** from the Grouped By list.
3. Expand the required Debug Interface in the Connect to Target window.
4. Right-click on the new Debug Configuration to display the context menu.
5. Select **Properties...** from the context menu to open the Connection Properties dialog box. Use this to customize your Debug Configuration.
6. Click the **RTOS** tab.
7. In the Vendor group, select the Vendor ID from the drop-down list.
8. In the Load When group, select either **connect** or **image_load**, depending on whether the Debug Agent is built into the OS or the image.

   This is important when you are connecting to a running target. When RealView Debugger connects, the Debug Agent might be found but symbols are not yet loaded and so the OS marker shows `RSD (PENDING SYMBOLS)`. The Debug Agent might communicate, to the debugger, all the information necessary to start RSD. In this case, RealView Debugger switches to RSD mode immediately, and you can read memory while the target is running.

   Otherwise, contact is made with the Debug Agent but RSD is not fully operational.
9. If you want to configure other RTOS settings, such as `Exit_Options` or `System_Stop`:
   a. Click **Advanced** to open the Connection Properties window with the `RTOS_config` group selected.

      Note

      If more than one `Advanced_Information` block exists, then the `RTOS_config` group in the first block is selected.

   b. Make the required changes.
   c. Select **File** → **Save and Close** from the menu to save your changes and close the Connection Properties window.
10. Click the **OK** button to close the Connection Properties dialog box.

Considerations when configuring RTOS settings

Consider the following when configuring RTOS:

- In some cases, settings in the `RTOS_config` group might conflict and so are ignored by RealView Debugger:

  Exit_Options

  RealView Debugger might ignore any of the ***_on_Unload** settings, if the image that is being unloaded has no relevance for the underlying OS, or if OS support has not been initialized.

System_Stop

These settings might conflict with the connect or disconnect mode configuration settings for the current target. This means that your target might stop on connect or disconnect even where you have specified **Never** or **Prompt** for this OS setting.

Configure the connect or disconnect settings to avoid this problem.

See also

- *OS configuration settings* on page 2-6
- *Procedure for configuring the connect and disconnect settings*
- *OS marker in the Process tab* on page 5-3.

2.3.4 Procedure for configuring the Basic Settings

For ARM architecture-based targets, you must also configure the Basic Settings on the Connection Properties dialog box.

To configure the Basic Settings:

1. Click the **Basic Settings** tab.
2. In the Vector catch group, deselect the `Enable` checkbox.
3. In the Semihosting group, deselect the `Enable` checkbox.
4. Click **OK** to save your changes to the board file and close the Connection Properties dialog box.

2.3.5 Procedure for configuring the connect and disconnect settings

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

- If a prompt is specified, it takes priority over any other user-defined setting. This prompt-first rule holds true regardless of where the setting is in the configuration hierarchy.
- If a (non-prompt) user-defined setting is specified, the setting takes priority.

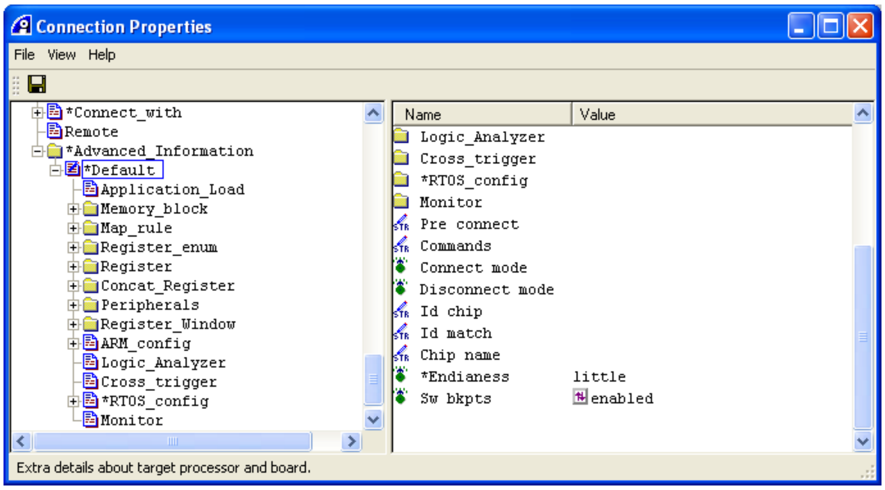
To configure connect and disconnect behavior for the new connection:

1. Select **Target** → **Connect to Target...** from the Code window main menu to open the Connect to Target window. Figure 2-2 on page 2-4 shows an example.
2. Select **Configuration** from the Grouped By list.
3. Expand the required Debug Interface in the Connect to Target window.
4. Right-click on the new Debug Configuration to display the context menu.
5. Select **Properties...** from the context menu to open the Connection Properties dialog box.
6. Locate the connect and disconnect mode settings:
   a. Click the **Commands** tab.
   b. Click **Advanced** to open the Connection Properties window with the appropriate `Advanced_Information` block selected.

Note

If more than one `Advanced_Information` block exists, then the first block is selected.

Figure 2-3 shows an example:



```
*Connect_with
Remote
*Advanced_Information
  *Default
    Application_Load
    Memory_block
    Map_rule
    Register_enum
    Register
    Concat_Register
    Peripherals
    Register_Window
    ARM_config
    Logic_Analyzer
    Cross_trigger
    *RTOS_config
    Monitor
```

Name	Value
Logic_Analyzer	
Cross_trigger	
*RTOS_config	
Monitor	
Pre connect	
Commands	
Connect mode	
Disconnect mode	
Id chip	
Id match	
Chip name	
*Endianess	little
Sw bkpts	enabled

Extra details about target processor and board.

Figure 2-3 Default group in the Connection Properties window

Note

The connect and disconnect mode settings are in the same group as the settings for defining commands. Therefore, clicking **Advanced** when the **Commands** tab is displayed causes the group containing the connect and disconnect mode settings to be selected in the Connection Properties window.

7. Right-click on the `Connect_mode` and `Disconnect_mode` setting as required to see the options available to you. These options are fixed and so might include options that are not supported by your Debug Interface. If you specify such an option, the debugger prompts you to select an appropriate mode when you try to connect or disconnect.

   For example, if you do not want to stop the target but still want to enable RSD mode, set `Connect_mode` to **no_reset_and_no_stop** from the context menu.

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

9. Click **OK** to close the Connection Properties dialog box.

Considerations for DSTREAM or RealView ICE

For connections through a DSTREAM or RealView ICE unit, what happens when a processor comes out of reset depends on the combination of:

- the RealView Debugger connection mode
- the `Default Post Reset State` setting in the RVConfig utility.

Table 2-1 summarizes the actions.

Table 2-1 Action performed after reset

Connection mode	Action performed after reset
Reset and Stop	Overridden by the RVConfig `Default Post Reset State` setting.
Reset and No Stop	Overridden by the RVConfig `Default Post Reset State` setting.
No Reset and Stop	Specified by the RealView Debugger `connect_mode` setting or **Connect Mode** on the Connect to Target window.
No Reset and No Stop	Specified by the RealView Debugger `connect_mode` setting or **Connect Mode** on the Connect to Target window.

See also

- *OS configuration settings* on page 2-6
- the following in the *RealView Debugger User Guide*:
  - Chapter 3 *Target Connection*
- the following in the *RealView Debugger Target Configuration Guide*:
  - *About customizing a DSTREAM or RealView ICE Debug Interface configuration* on page 2-3 for details of the `Default Post Reset State` setting
  - Chapter 3 *Customizing a Debug Configuration*.
- *ARM DSTREAM and RVI Using the Debug Hardware Configuration Utilities*.

2.4 Managing configuration settings

RealView Debugger provides great flexibility in how to customize Debug Configurations so that you can control your debug target and any custom hardware that you are using. Some settings can be defined in the Debug Configuration board file (for example `CONNECTION=RVI_OS_tst`) or on a per-board basis using groups in one or more BCD files linked to a Debug Configuration (for example `AP.bcd`).

See also:

- *Avoiding conflicts between settings*.

2.4.1 Avoiding conflicts between settings

To avoid conflicts between settings when you link multiple BCD groups to a Debug Configuration:

- configure memory map related settings only in BCD files
- configure all other settings only in the Debug Configuration.

See also

- the following in the *RealView Debugger Target Configuration Guide*:
  - Chapter 3 *Customizing a Debug Configuration*
  - Chapter 4 *Configuring Custom Memory Maps, Registers and Peripherals*.

Chapter 3
Connecting to a Target and Loading an Image

You establish OS-aware connections and load images to a target using the same methods as any other target. However, what happens after you connect or load an image depends on how you have configured OS-awareness for the associated Debug Configuration.

This chapter describes how to establish OS-aware connections and load an image. It contains the following sections:

3.1 Before connecting

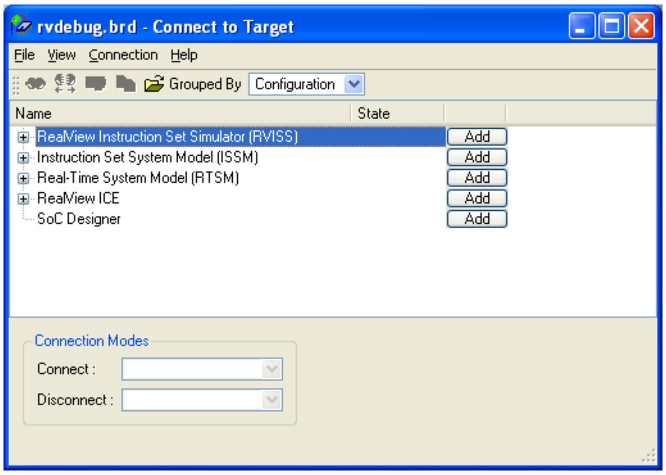
Make sure that you:

1. Compile your OS image with debug symbols enabled so that the debugger can find the data structures it requires for *Halted System Debug* (HSD). If the target is in *Running System Debug* (RSD) mode, this might not be required.
2. Install your OS plug-ins as described in *Enabling OS support* on page 2-2.
3. Create a new OS connection as described in *Creating a new OS-aware Debug Configuration* on page 2-3.
4. Configure your OS-enabled connection as described in *Customizing an OS-aware Debug Configuration* on page 2-6.

3.2 Connecting from the Code window

To connect to a target on an OS-aware connection:

1. Start RealView Debugger.
2. Select **Target** → **Connect to Target...** from the Code window main menu to open the Connect to Target window. Figure 3-1 shows an example:


rvdebug.brd - Connect to Target

File View Connection Help

Grouped By: Configuration

Name	State	
RealView Instruction Set Simulator (RVISS)		Add
Instruction Set System Model (ISSM)		Add
Real-Time System Model (RTSM)		Add
RealView ICE		Add
SoC Designer		Add

Connection Modes

Connect :

Disconnect :

Figure 3-1 Connect to Target window

3. Select **Configuration** from the Grouped By list.
4. Expand the required Debug Interface that you are using to access your development platform.

   For this example, expand `RealView ICE`.
5. Expand the Debug Configuration that you want to use.

   For this example, expand the `RVI_OS_tst` Debug Configuration that was created in *Creating a new OS-aware Debug Configuration* on page 2-3.

   ——— **Note** ———

   If you are connecting to a running target, see *Connecting to a running target* on page 3-4 before you connect.

   ———————————
6. Double-click on the target to connect.

3.3 Connecting to a running target

Depending on your target, connecting to a running target results in different startup conditions, either:

- RSD is enabled and contact with the Debug Agent is established. You can start working with threads because the Debug Agent has communicated all the necessary information to the debugger to start RSD.

  ——— **Note** ———

  In this case, you must also load symbols to start debugging. You do not have to load the OS symbols, because the application symbols are sufficient.

  ————————

- RSD is enabled and contact with the Debug Agent is established. However, the information required to start RSD is part of the OS symbols. In this case, you must load symbols before you can debug your image.

See also:

- *Before you start*
- *Procedure for connecting to a running target*.

3.3.1 Before you start

If you want to connect to a running target, or disconnect from a target without stopping the application, use the `Connect_mode` and `Disconnect_mode` configuration settings as described in *Procedure for configuring the connect and disconnect settings* on page 2-9.

——— **Note** ———

Before connecting to a running target, make sure that the endianness in the connection properties matches the endianness of the target. If it does not, you might see the following error messages:

```
Error V2001B (Vehicle): Cannot set break on non-instruction boundary
Error: Unable to detect endianness of target.
```

————————

See also

- *Procedure for configuring the connect and disconnect settings* on page 2-9
- the following in the *RealView Debugger User Guide*:
  — *Connecting to a target using different modes* on page 3-44
  — *Disconnecting from a target using different modes* on page 3-55.
- the following in the *RealView Debugger Target Configuration Guide*:
  — *Specifying connect and disconnect mode* on page 3-19.

3.3.2 Procedure for connecting to a running target

To connect to a running target:

1. Start RealView Debugger.
2. Open the Connect to Target window to view the OS-enabled connection that is running your application.
3. Select **Configuration** from the Grouped By list.

4. Expand the required Debug Interface that you are using to access your development platform. For this example, expand `RealView ICE`.
5. Expand the Debug Configuration that you want to use. For this example, expand the `RVI_OS_tst` Debug Configuration that was created in *Creating a new OS-aware Debug Configuration* on page 2-3.
6. Select the required target for which the connect mode is to be set. The Connect field in the Connection Modes group at the bottom of the Connect to Target window is enabled.
7. In the Connection Modes group, set Connect to **No Reset / No Stop**.
8. Double-click on the target to connect.

See also

- *Creating a new OS-aware Debug Configuration* on page 2-3
- *Loading the symbols for an image on a running target* on page 3-6

3.4 Loading the symbols for an image on a running target

If your target is running, then you must load the symbols for the image.

To load the symbols for an image on a running target:

1. Connect to the running target as described in *Procedure for connecting to a running target* on page 3-4.
2. When the OS marker in the Process Control view shows `RSD (PENDING SYMBOLS)`, select **Target → Load Image...** from the Code window main menu to open the Select Local File to Load dialog box.
3. To load only the symbols:
   a. Locate the required image.
   b. Select **Symbols Only**.

   Note

   Make sure that both **Auto-Set PC** and **Set PC to Entry Point** are not selected.

4. Click **Open** to load the image symbols.

3.5 Loading a multi-image application on an OS-aware connection

You load a single image to a target on an OS-aware connection in the same way as any other image. By default, loading an image replaces any other image that is loaded on the target. However, if your application has multiple images, then the method used to load the additional images is different from the method used to load the first image.

To load a multi-image application:

1. Connect to the target on the OS-aware connection.
2. For the first image you want to load:
   a. Select **Target → Load Image...** from the Code window main menu to open the Select Local File to Load dialog box.
   b. Locate the file for the first image that you want to load.
   c. Optionally, in the Sections field, enter any specific image sections that you want to load. All sections are loaded by default.
   d. If your image accepts arguments, you might want to specify initial values for the arguments. Enter any arguments in the Arguments field.
   e. If required, in the Offset field specify an address offset to be added to all sections when computing the load addresses.

   Do not change any other settings.
3. Click **Open** to load the image.
4. For each additional image you want to load:
   a. Select **Target → Load Image...** from the Code window main menu to open the Select Local File to Load dialog box.
   b. Locate the file for the first image that you want to load.
   c. Deselect the **Replace Existing Files** check box.
      This ensures that this executable file does not replace any previously loaded images.
      You can also use the `ADDFILE` command.
   d. Optionally, in the Sections field, enter any specific image sections that you want to load. All sections are loaded by default.
   e. If your image accepts arguments, enter the required arguments in the Arguments field.
   f. If required, in the Offset field specify an address offset to be added to all sections when computing the load addresses.
5. Click **Open** to load the image.

——— **Note** ———

If RealView Debugger cannot find a source file for a loaded image, then it automatically displays the Source File Location dialog box, where you can specify the new path for that source file.

———————

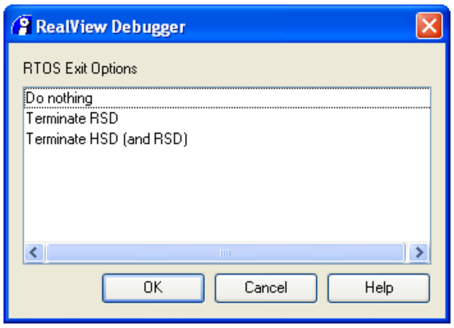
See also:
- *Creating a new OS-aware Debug Configuration* on page 2-3
- *Connecting from the Code window* on page 3-3
- the following in the *RealView Debugger User Guide*:
  — *Loading multiple images to the same target* on page 4-14
  — *Autoconfiguring search rules for locating source files* on page 4-34.

- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 2 *RealView Debugger Commands*.

3.6 OS Exit Options

The `RTOS_config` group configuration settings define how OS-awareness is disabled. You can specify the action to take when an image is unloaded or when you disconnect. Make sure that these do not conflict with the connect or disconnect mode specified for your target.

When you unload (or reload) an image, the `Exit_Options` setting decides how to disable OS-awareness. If this is set to **Prompt_on_Unload** (the default setting), then RealView Debugger displays a List Selection dialog box to enable you to specify the exit conditions, shown in Figure 3-2.


RealView Debugger

RTOS Exit Options

Do nothing
Terminate RSD
Terminate HSD (and RSD)

Figure 3-2 RTOS Exit Options List Selection dialog box

1. Select:

   Do nothing

   To maintain the current OS state.

   If the dialog box is opened because you chose to unload an image, then the image is unloaded. To reload and run the image again:

   1. Enter the `STOP` command in the **Cmd** tab of the Output view to stop the target processor.
   2. Disconnect the target processor.
   3. Establish a connection to the target processor.
   4. Load the image.
   5. Run the image.

   ——— **Note** ———

   Clicking **Cancel** has the same effect as selecting **Do nothing**.

   ————————

   Terminate RSD

   To disable RSD and end communication with the Debug Agent. The application drops out of RSD mode into HSD mode.

   Terminate HSD (and RSD)

   To end communication with the Debug Agent and unload the OS plug-in, that is the `*.dll` file.

2. Click **OK** to close the List Selection dialog box.

See also:

- *Procedure for configuring the connect and disconnect settings* on page 2-9.
- the following in the *RealView Debugger Command Line Reference Guide*:
  - *STOP* on page 2-267.

3.7 Interrupts when loading an image

When you load an image to your target, make sure that all interrupts are reset. If interrupts are not reset when the image executes (either by a `STEP` or `GO` command), an IRQ might occur that is associated with the previous image. If this happens, then it can cause the current image to run incorrectly.

See also:

- the following in the *RealView Debugger Command Line Reference Guide*:
  - *GO* on page 2-159
  - *STEPINSTR* on page 2-259
  - *STEPLINE* on page 2-261
  - *STEPOINSTR* on page 2-263
  - *STEPO* on page 2-265.

3.8 Resetting OS state

The OS plug-in samples the OS state to determine the current state of the OS kernel, for example uninitialized, pending symbol load, or running.

HSD is enabled and any resources shown are those of the previous image execution when you perform the following steps:

1. Load and run an image to a target on an OS-aware connection.
2. Either:
   - stop execution
   - disconnect from the target, then reconnect.
3. Reload without resetting the OS state to its initial value.

To make sure that this does not happen, you must always reset the OS state to its uninitialized value each time the OS is reloaded. For example, you might have to perform a power-cycle.

See also:

- the following in the *RealView Debugger User Guide*:
  - *Connecting to a target* on page 3-28
  - *Disconnecting from a target* on page 3-53
  - *Loading an executable image* on page 4-4
  - *Stopping execution* on page 8-4.

Chapter 4
Associating Threads with Views

When *Halted System Debug* (HSD) or *Running System Debug* (RSD) is fully operational, you can start to work with threads in the RealView® Debugger Code window. This is described in the following sections:

- *Attaching and unattaching windows* on page 4-2
- *The current thread* on page 4-3
- *Using the Cycle Threads button* on page 4-4
- *Working with the thread list* on page 4-5.

You can also work with threads in the Process Control view, see Chapter 5 *Working with OS-aware Connections in the Process Control view* for details.

4.1 Attaching and unattaching windows

If you are licensed to use multiprocessor debugging mode, RealView Debugger Code windows can be attached to specific connections. Similarly, if you are working with an OS-enabled connection, you can attach Code windows to threads.

——— **Note** ———

Be aware that attaching windows to threads does not work in the same way as attaching windows to connections in multiprocessor debugging mode.

————————

The window attachment status is displayed in the Code window title bar:

<blank> Specifies that the Code window is not attached to a thread or a connection.

If an OS-aware connection is the current connection, then by default, all unattached Code windows display details of the *current thread* for that connection. That is, the details of the thread that was most recently running on the target when the target stopped.

`[Target]` Specifies that the Code window is attached to the OS-aware connection, that is a debug target board or a specified processor on a multiprocessor board. This window displays details of the current thread on that target, if available.

——— **Note** ———

Be aware that, if you later attach the Code window to a thread, and unattach the window from the thread, then the connection is also unattached from that Code window. If the OS-aware connection is not the current connection, then the details of the OS-aware connection are replaced by those of the current connection.

————————

thread_id When the Code window is attached to a specific thread, the thread ID replaces the connection name in the title bar (for example, `T0x25520_thread_01.ARM`). If the Code window is attached to the connection, then the connection status is also replaced.

——— **Note** ———

If you open a new Code window, the attachment of that Code window is inherited from the calling window.

————————

When working with threads, you can change the attachment of your Code window using the *thread list*.

See also:

- *Working with the thread list* on page 4-5
- the following in the *RealView Debugger User Guide*:
  - — *Attaching a Code window to a connection* on page 7-10.

4.2 The current thread

When working with a multithreaded application, RealView Debugger designates a thread as the *current thread*:

- In HSD, the current thread is initially set to the thread that was running on the processor when it stopped. If you are working with an unattached Code window, this shows details about the current thread.

  When the current thread changes, for example when you stop the target with a different thread active, the Output view displays details of the new current thread. This includes the thread number in decimal and the thread name, if it is available.

- Initially in RSD, the current thread is undefined and so RealView Debugger designates a thread at random to be the current thread.

You can change the current thread using the **Cycle Threads** button.

See also:

- *Using CLI commands for examining thread information*
- *Using the Cycle Threads button* on page 4-4.

4.2.1 Using CLI commands for examining thread information

If you are working in an unattached Code window, the current thread defines the scope of many CLI commands. If you are working in an attached window, the scope of CLI commands is defined by the attached thread.

You can use CLI commands to work with threads, for example:

`print @r1`	Prints the value of the register R1 for the thread that was current when the processor stopped.
`thread,next`	Changes the current thread.

See also

- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 2 *RealView Debugger Commands* for details of the `THREAD` command.

4.3 Using the Cycle Threads button

When HSD or RSD is fully operational, this enables the **Cycle Threads** button and drop-down arrow on the Connect toolbar in the Code window:

- Click **Cycle Threads** to change the current thread.
- Click the **Cycle Threads** drop-down arrow to access the thread list where you can change your thread view and windows attachment.

——— **Note** ———

For HSD, the thread list is only available when the target is stopped.

———————————

See also:

- *Viewing thread details*.

4.3.1 Viewing thread details

Click the **Cycle Threads** button to cycle through the threads to view the details for another thread. This changes the current thread and updates your code view. The new current thread appears in the Code window title bar and the color box changes.

You can only cycle through the threads in this way in a Code window that is not attached to a thread. If your Code window is attached to a thread and you try to cycle threads in this way, a dialog box appears:

```
Window attached. Do you want to detach first?
```

Click **Yes** to unattach the window and change the thread view. Click **No** to abort the action and leave the thread view unchanged.

——— **Note** ———

You can use the **Cycle Threads** button to cycle through the thread list in a Code window that is attached to a *connection* without changing the windows attachment.

———————————

When you change the current thread, the Output view displays details of the new current thread.

You can also change the current thread using the **Thread** tab in the Process Control view.

See also

- *Working with the thread list* on page 4-5
- *Examining thread details in the Thread tab* on page 5-8.

4.4 Working with the thread list

This section describes how to work with the thread list.

See also:

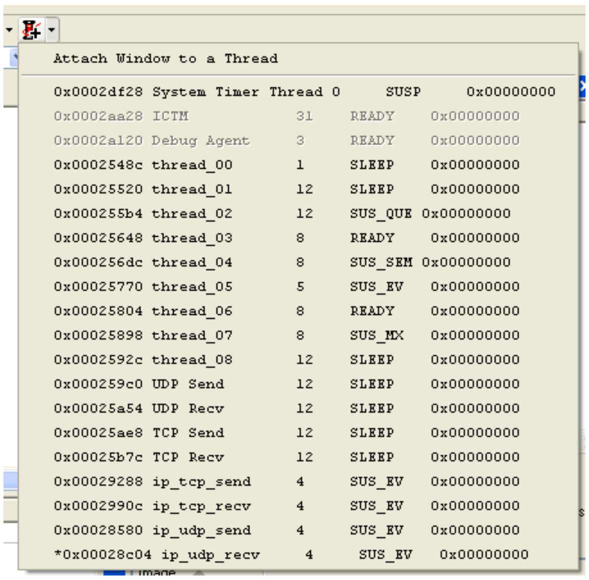
- *The thread list*
- *Identifying the current thread* on page 4-6
- *Captive threads* on page 4-6
- *Attaching windows to threads* on page 4-6.

4.4.1 The thread list

The thread list on the **Cycle Threads** button is available:

- for a processor running in HSD mode, when that processor is stopped
- for a processor running in RSD mode.

Click the **Cycle Threads** drop-down arrow to cause RealView Debugger to fetch the list of threads from the target and display a summary. Figure 4-1 shows an example:



```
Attach Window to a Thread

0x0002df28 System Timer Thread 0     SUSP     0x00000000
0x0002aa28 ICTM             31   READY    0x00000000
0x0002a120 Debug Agent      3    READY    0x00000000
0x0002548c thread_00        1    SLEEP    0x00000000
0x00025520 thread_01        12   SLEEP    0x00000000
0x000255b4 thread_02        12   SUS_QUE 0x00000000
0x00025648 thread_03        8    READY    0x00000000
0x000256dc thread_04        8    SUS_SEM 0x00000000
0x00025770 thread_05        5    SUS_EV   0x00000000
0x00025804 thread_06        8    READY    0x00000000
0x00025898 thread_07        8    SUS_MX   0x00000000
0x0002592c thread_08        12   SLEEP    0x00000000
0x000259c0 UDP Send         12   SLEEP    0x00000000
0x00025a54 UDP Recv         12   SLEEP    0x00000000
0x00025ae8 TCP Send         12   SLEEP    0x00000000
0x00025b7c TCP Recv         12   SLEEP    0x00000000
0x00029288 ip_tcp_send      4    SUS_EV   0x00000000
0x0002990c ip_tcp_recv      4    SUS_EV   0x00000000
0x00028580 ip_udp_send      4    SUS_EV   0x00000000
*0x00028c04 ip_udp_recv      4    SUS_EV   0x00000000
```

Figure 4-1 Example thread list

——— **Note** ———

The number of threads a Debug Agent can handle is defined by your OS vendor. When the thread buffer is full, no threads can be displayed.

———————————

The first option on this menu is **Attach Window to a Thread**. Use this to control windows attachment.

Below the menu spacer is a snapshot of the threads running on the target when the request was made. The thread information shown depends on the OS plug-in. In the example in Figure 4-1, the fields shown (from left to right) for each thread are the:

- address of the thread control block
- name of the thread
- status of the thread.

Click on a new thread, in the thread list, to change the code view so that it displays the registers, variables, and code for that thread:

- If you click on a new thread in an unattached Code window, it becomes attached to the thread automatically. The thread details appear in the Code window title bar and the color box changes color.

  If you change the thread view in this way, other unattached windows are not affected, that is they remain unattached and continue to show the current thread.

- If you click on a new thread in a Code window that is attached to a connection, it becomes attached to the thread automatically.

- If you click on a new thread in a Code window that is attached to a thread, it becomes attached to the specified thread.

In this release, the Debug Agent handles up to 64 threads. Where the thread list does not show the full details, use the thread selection box to see all the threads detected on the system.

See also

- *Attaching windows to threads*.

4.4.2 Identifying the current thread

In Figure 4-1 on page 4-5, the asterisk (*) shows the current thread. Because the Code window is unattached, any thread-specific CLI commands you submit operate on this thread.

——— **Note** ———

RealView Debugger designates a thread to be the current thread when you are in RSD.

————————

See also

- *The current thread* on page 4-3.

4.4.3 Captive threads

The thread list shows:

- All threads on the system that can be captured by RealView Debugger, that is they can be brought under debugger control. These are called *captive threads*.

- Special threads, scheduled along with other tasks in the system, that cannot be captured, that is they are not under the control of RealView Debugger. These *non-captive threads* are grayed out.

Threads that are essential to the operation of RSD (such as Debug Agent) are grayed out to show that they are not available to RealView Debugger. The threads that are grayed out depends on your target.

See also

- *Special threads in the Thread tab* on page 5-9.

4.4.4 Attaching windows to threads

You can attach the Code window to the current thread, or to any other thread as required.

Note

Attaching windows to threads does not work in the same way as attaching windows to connections in multiprocessing mode.

Attaching the Code window to the current thread

To attach the Code window to the current thread:

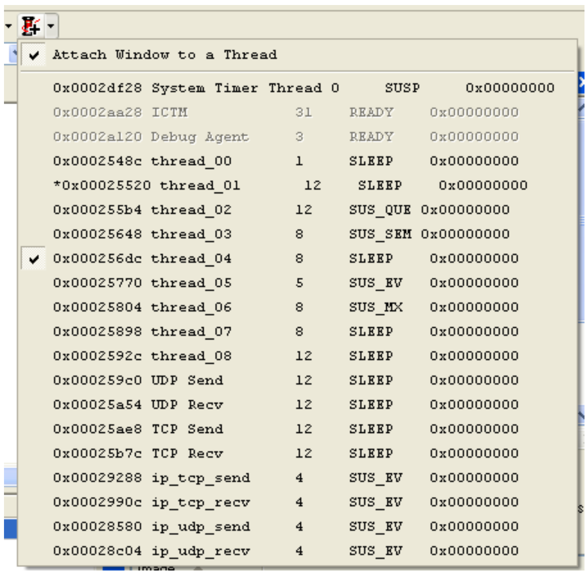
1. Click the **Cycle Threads** drop-down arrow to display the thread list, shown in Figure 4-1 on page 4-5.
2. Select **Attach Window to a Thread** to attach the Code window to the current thread.

   Note

   Select the option again to unattach the thread.

If you display the thread list from an unattached Code window, click on a thread to change the thread view and attach the window automatically. The thread details appear in the Code window title bar and the color box changes color.

If you display the thread list from a Code window that is attached to a thread, the first menu item, **Attach Window to a Thread**, is ticked. The attached thread is also marked by a check mark, shown in Figure 4-2.



```
✓ Attach Window to a Thread

  0x0002df28 System Timer Thread 0     SUSP     0x00000000
  0x0002aa28 ICTM           31    READY    0x00000000
  0x0002a120 Debug Agent    3     READY    0x00000000
  0x0002548c thread_00      1     SLEEP    0x00000000
  *0x00025520 thread_01      12    SLEEP    0x00000000
  0x000255b4 thread_02      12    SUS_QUE 0x00000000
  0x00025648 thread_03      8     SUS_SEM 0x00000000
✓ 0x000256dc thread_04      8     SLEEP    0x00000000
  0x00025770 thread_05      5     SUS_EV   0x00000000
  0x00025804 thread_06      8     SUS_MX   0x00000000
  0x00025898 thread_07      8     SLEEP    0x00000000
  0x0002592c thread_08      12    SLEEP    0x00000000
  0x000259c0 UDP Send       12    SLEEP    0x00000000
  0x00025a54 UDP Recv       12    SLEEP    0x00000000
  0x00025ae8 TCP Send       12    SLEEP    0x00000000
  0x00025b7c TCP Recv       12    SLEEP    0x00000000
  0x00029288 ip_tcp_send    4     SUS_EV   0x00000000
  0x0002990c ip_tcp_recv    4     SUS_EV   0x00000000
  0x00028580 ip_udp_send    4     SUS_EV   0x00000000
  0x00028c04 ip_udp_recv    4     SUS_EV   0x00000000
```

Figure 4-2 Example thread list in an attached window

Note

If you select a thread in the thread list, it does not become the current thread. This changes the thread view and attaches the Code window. However, if you click the **Cycle Threads** button to change the thread, the next thread in the thread list becomes the current thread.

Attaching the Code window to any thread

To attach the Code window to any thread:

1. Select **View** → **Process Control** from the Code window main menu to open the Process Control view.
2. Click the **Threads** tab.
3. Right-click on the name of the thread that you want to attach to the Code window. The context menu is displayed.
4. Select **Attach Window to** from the context menu.

   The Code window is attached to the selected thread.

See also

- Chapter 5 *Working with OS-aware Connections in the Process Control view*
- the following in the *RealView Debugger User Guide*:
  - *Attaching a Code window to a connection* on page 7-10.

Chapter 5
Working with OS-aware Connections in the Process Control view

The Process Control view shows details about each connection known to RealView® Debugger. This chapter includes:

5.1 The Process Control view and OS-aware connections

When the OS has been detected, the view contains the following tabs:

Process Use the **Process** tab to see the processor details and information about any image(s) loaded onto the debug target. That is:

- image name
- image resources, including DLLs
- how the image was loaded
- load parameters
- associated files
- execution state.

For OS-aware connections, additional details are displayed showing the OS execution state.

Memory Map

If you are working with a suitable target you can enable memory mapping and then configure the memory using the **Memory Map** tab.

Thread This displays OS-specific information about the threads that are configured on the target.

See also:

- *OS marker in the Process tab* on page 5-3
- *Examining thread details in the Thread tab* on page 5-8
- the following in the *RealView Debugger User Guide*:
  - — Chapter 4 *Loading Images and binaries*
  - — Chapter 9 *Mapping Target Memory*.

5.2 OS marker in the Process tab

OS markers in the Process Control view show the current state of the OS process. To open the Process Control view if it is not visible in your Code window, select **View → Process Control** from the Code window main menu.

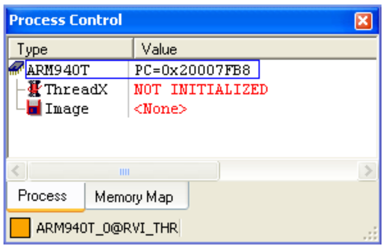
See also:

- *OS marker initial state*
- *OS marker with an OS-aware image loaded* on page 5-4
- *OS marker for a running target with HSD fully operational* on page 5-4
- *OS marker for a running target with RSD fully operational* on page 5-5
- *OS marker status* on page 5-5
- *Context menu* on page 5-6.

5.2.1 OS marker initial state

You can choose to load the OS plug-in when you establish an OS-aware connection to the target or when you load an image. If *Halted System Debug* (HSD) or *Running System Debug* (RSD) is enabled, the **Process** tab contains the OS marker:

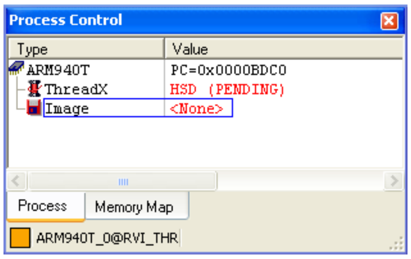
- Figure 5-1 shows the initial state after connection, and you have chosen to load the plug-in on image load. The default setting is to load the plug-in on image load.



Type	Value
ARM940T	PC=0x20007FB8
ThreadX	NOT INITIALIZED
Image	<None>

Figure 5-1 OS marker initial state (OS plug-in not loaded)

- Figure 5-2 shows the initial state after connection, and you have chosen to load the plug-in on connection.



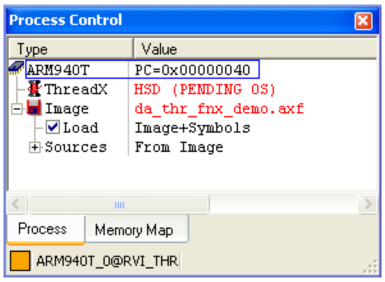
Type	Value
ARM940T	PC=0x0000BDC0
ThreadX	HSD (PENDING)
Image	<None>

Figure 5-2 OS marker initial state (OS plug-in loaded)

In this state, the **Thread** tab and the **Cycle Threads** button are not available.

5.2.2 OS marker with an OS-aware image loaded

With an OS-aware image loaded but not running (or with symbols only loaded), the **Process** tab contains the OS marker shown in Figure 5-3.



Type	Value
ARM940T	PC=0x00000040
ThreadX	HSD (PENDING OS)
Image	da_thr_fnx_demo.axf
✓ Load	Image+Symbols
Sources	From Image

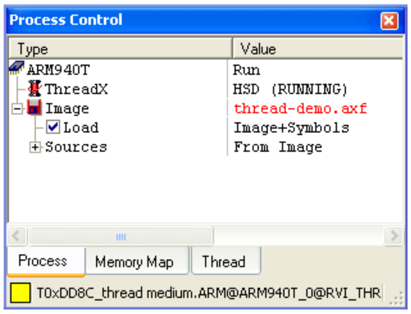
Figure 5-3 OS marker with loaded image (OS plug-in loaded)

Figure 5-3 shows a debug target where RealView Debugger has successfully loaded the required OS debug symbols, and the OS has not started.

In this state, the **Thread** tab and the **Cycle Threads** button are not available.

5.2.3 OS marker for a running target with HSD fully operational

Figure 5-4 shows a target running in HSD mode. RealView Debugger has detected that a suitable OS-aware image has been loaded to the target and threads are being rescheduled.



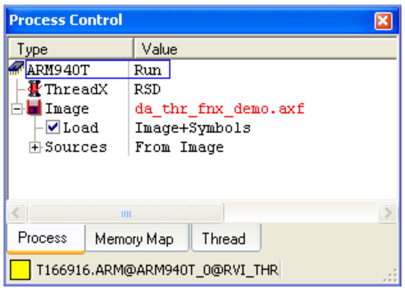
Type	Value
ARM940T	Run
ThreadX	HSD (RUNNING)
Image	thread-demo.axf
✓ Load	Image+Symbols
Sources	From Image

Figure 5-4 OS marker in the Process Control view (HSD)

In this state, the **Thread** tab is available and the **Cycle Threads** button is enabled on the Connect toolbar.

5.2.4 OS marker for a running target with RSD fully operational

Figure 5-5 shows a target running in RSD mode. RealView Debugger has detected that a suitable OS-aware image has been loaded to the target and threads are being rescheduled.



Type	Value
ARM940T	Run
ThreadX	RSD
Image	da_thr_fnx_demo.axf
Load	Image+Symbols
Sources	From Image

T166916.ARM@ARM940T_0@RVI_THR

Figure 5-5 OS marker in the Process Control view (RSD)

In this state, the **Thread** tab is available and the **Cycle Threads** button is enabled on the Connect toolbar.

5.2.5 OS marker status

Table 5-1 describes OS marker status in the Process Control view.

Table 5-1 OS marker status in the Process Control view

Status	Meaning
NOT INITIALIZED	HSD or RSD is enabled but the triggering event that loads the plug-in, the `*.dll` file, has not occurred. This is the state immediately after connection if the plug-in is configured to load after image load.
HSD (PENDING)	The plug-in has been loaded and RealView Debugger is waiting for the OS symbols to be loaded. This is the state immediately after connection if the plug-in is configured to load on connection.
HSD (PENDING OS)	Symbols are loaded but the OS-aware plug-in is not yet active. This is the state immediately after image load.
HSD	HSD is fully operational.
HSD (RUNNING)	The system is running when RSD is disabled. This means that no up-to-date information about the OS can be shown. This value is never shown when RSD is enabled.
RSD (PENDING DA)	RealView Debugger is waiting for notification that a Debug Agent has been found.
RSD	RSD is fully operational.
HSD (RSD STOPPED)	RealView Debugger has changed from RSD to HSD debugging mode. This occurs when: • an HSD breakpoint is activated • you stop the processor using the `STOP` command • you select the **Stop Target** option on the Process Control view context menu. HSD is available.

Table 5-1 OS marker status in the Process Control view (continued)

Status	Meaning
HSD (RSD INACTIVE)	RSD was operational or pending but is now disabled. This occurs only by a user request. HSD is available.
HSD (RSD DEAD)	RSD was operational but is now disabled. This is usually because the Debug Agent is not responding to commands or the OS has crashed. HSD is available.
RSD DEAD	RSD was operational but is now disabled. The debug target is still running. HSD is not available.
RSD (PENDING SYMBOLS)	RSD is operational, contact to the Debug Agent has been established but no symbols have been loaded. This means that RSD only works partially until the symbol table is loaded. This usually occurs after connecting to a debug target that is already running the Debug Agent.

The OS marker is usually shown in black text in the **Process** tab. Where the marker is shown in red, either OS support is not ready, as shown in Figure 5-3 on page 5-4, or there has been an error.

See also

- *Context menu*
- *Setting breakpoints* on page 7-4.

5.2.6 Context menu

Right-click on the OS marker to see the context menu where you can control the target on the OS-aware connection:

Disable RSD Depending on the current mode, this option disables or enables RSD.

The initial state depends on the `RSD` setting in the `RTOS_config` group of the connection properties:

- If RSD is enabled, select **Disable RSD** to shutdown the Debug Agent cleanly. This disables all previously set RSD breakpoints. However, all HSD breakpoints are maintained. You can re-enable RSD again if required.
- If you select **Enable RSD**, system breakpoints are enabled but you have to enable thread breakpoints yourself.

Stop Target Stops the target processor.

If the processor is running in RSD mode, then RSD is suspended. The status in the Process Control view changes to (see Table 5-1 on page 5-5):

```
HSD (RSD STOPPED)
```

Click **Run** to start execution and restart RSD.

Note

This option is disabled if you have set the `System_Stop` setting to **Never** in the connection properties, or the processor is running in HSD mode.

Properties Select this option to see details about the Debug Agent. This includes:

- the status of the RSD module
- settings as specified in the connection properties

- RSD breakpoints.

See also

- *Customizing an OS-aware Debug Configuration* on page 2-6
- *More on breakpoints* on page 7-3
- *Setting breakpoints* on page 7-4.

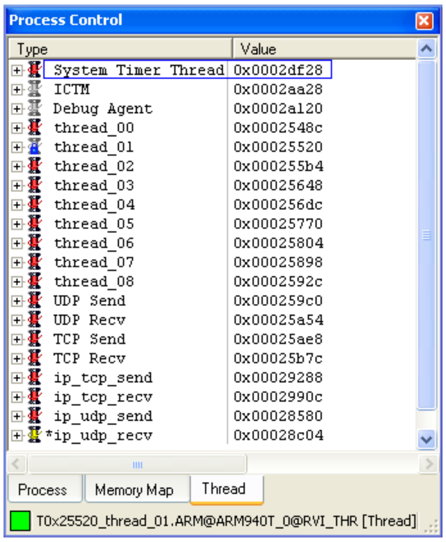
5.3 Examining thread details in the Thread tab

The **Thread** tab in the Process Control view displays OS-specific information about the threads that are configured on the target:

- In HSD mode, the **Thread** tab shows the thread state when the processor is stopped. When you start a processor that runs in HSD mode, the thread details are cleared from the tab.
- In RSD mode, the **Thread** tab shows a snapshot of the last known state of the system.

Select **View** → **Threads Tab** from the menu to display the **Thread** tab if the Process Control view is not currently visible. The Process Control view is opened as a floating view.

Expand the tree to see each configured thread and the associated summary information, shown in Figure 5-6.



Type	Value
System Timer Thread	0x0002df28
ICTM	0x0002aa28
Debug Agent	0x0002a120
thread_00	0x0002548c
thread_01	0x00025520
thread_02	0x000255b4
thread_03	0x00025648
thread_04	0x000256dc
thread_05	0x00025770
thread_06	0x00025804
thread_07	0x00025898
thread_08	0x0002592c
UDP Send	0x000259c0
UDP Recv	0x00025a54
TCP Send	0x00025ae8
TCP Recv	0x00025b7c
ip_tcp_send	0x00029288
ip_tcp_recv	0x0002990c
ip_udp_send	0x00028580
*ip_udp_recv	0x00028c04

🟩 T0x25520_thread_01.ARM@ARM940T_0@RVI_THR [Thread]

Figure 5-6 Thread tab in the Process Control view

In this example, the Process Control view is floating and so the color box on the view reflects the color box of the calling Code window.

See also:

- *Thread icons* on page 5-9
- *Special threads in the Thread tab* on page 5-9.

5.3.1 Thread icons

Each thread is identified by an icon:

- A red icon indicates a thread that is currently not captive.
- A blue padlock indicates that the Code window is attached to this thread.
- A gray icon indicates a special thread that is not under the control of RealView Debugger and so cannot become captive.
- When you are working in RSD mode, a yellow icon indicates that a thread is captive, for example after hitting a breakpoint.

  The padlock icon is shown if the Code window is attached to the thread, regardless of its captive state.

  In HSD mode, a yellow icon indicates the thread that was running when the target stopped.

The asterisk (*) shows the current thread.

5.3.2 Special threads in the Thread tab

For RSD, the **Thread** tab includes special threads that are visible but are not available to you. Threads of this type depend on your OS. For example, such threads might include:

ICTM Part of the Debug Agent, the *IMP Comms Target Manager* handles communications between the Debug Agent and the target.

Debug Agent

The main part of the Debug Agent runs as a thread under the target OS. This passes thread-level commands to RealView Debugger.

See also

- *Debug Agent* on page 1-7.

5.4 Changing the current thread and attachment

You can change the current thread and attachment status from the Process Control view:

1. Select **View** → **Process Control** from the Code window main menu to open the Process Control view.
2. Click the **Threads** tab.
3. Right-click on the name of the thread that you want to make the current thread. The context menu is displayed.
4. Select **Make Current** from the context menu.

See also:
- *The current thread* on page 4-3.

5.5 Updating the system snapshot

When running in RSD mode, you can update the system snapshot:

1. Select **View** → **Process Control** from the Code window main menu to open the Process Control view.
2. Click the **Threads** tab.
3. Right-click on any thread name. The context menu is displayed.
4. Select **Update All** from the context menu. A snapshot of the current state of all threads is taken, and the **Threads** tab is updated accordingly.

   ——— **Note** ———

   Selecting this option when the target is running in HSD mode has no effect.

   ————————

Chapter 6
Viewing OS Resources

This chapter describes how to view the OS resources. It includes:

6.1 About viewing OS resources

You view OS resources using the Resource Viewer (see Figure 6-1 on page 6-3). The Resource Viewer contains the Resources list, which displays all the resources available to view in RealView® Debugger.

With an OS application loaded, OS-specific tabs are added to the Resource Viewer to display the processes or threads that are configured.

In this release, the Debug Agent handles up to 64 threads and the Resources list shows all the threads on the system. This display differs from the thread list where threads that are not under the control of RealView Debugger are grayed out.

Other tabs might be included to support the display of other OS objects, for example **process_list**.

Where no OS has been detected, the Resources list contains only the **Connection** tab showing the connection.

For more information about the meaning of the tabs and information displayed in the Resource Viewer, see the user manual for your OS.

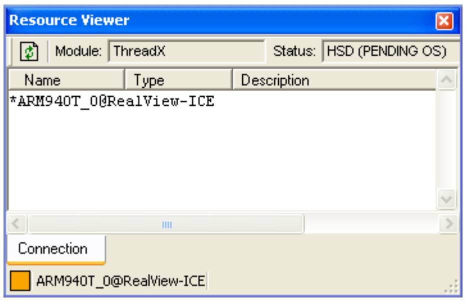
——— **Note** ———

Different OS plug-ins might display information in different ways in the Resource Viewer, for example by adding new menus. Similarly, other RealView Debugger extensions might add other tabs to the Resources list.

6.2 Viewing details in the Output view

To see the details of an entry in the Output view:

1. Select **View** → **Resource Viewer** from the Code window main menu to open the Resource Viewer. Figure 6-1 shows an example:



```
*ARM940T_0@RealView-ICE
```

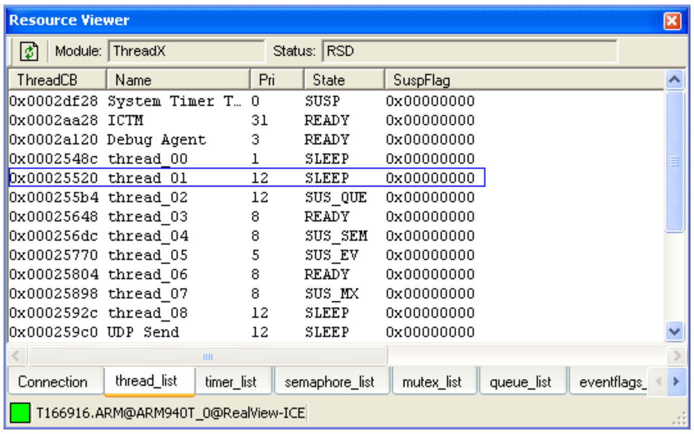
Figure 6-1 Resource Viewer

Note

If the target is running in *Halted System Debug* (HSD) mode, you must stop execution to view the OS resources.

2. Select the tab containing the entities that you want to view.

   For example, to see the threads, select the **task_list** tab. Figure 6-2 shows an example:



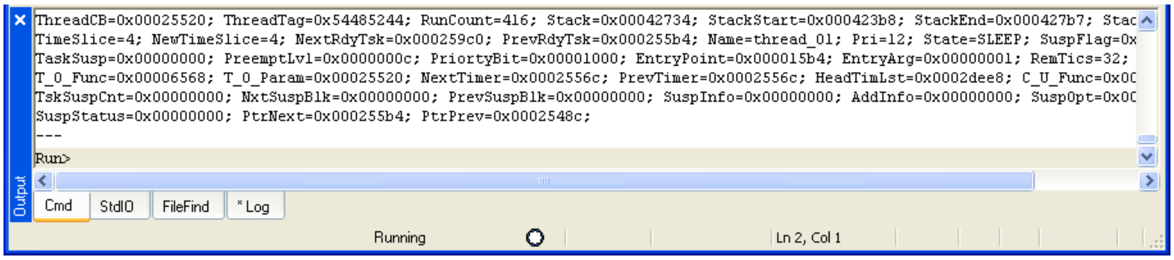
ThreadCB	Name	Pri	State	SuspFlag
0x0002df28	System Timer T…	0	SUSP	0x00000000
0x0002aa28	ICTM	31	READY	0x00000000
0x0002a120	Debug Agent	3	READY	0x00000000
0x0002548c	thread_00	1	SLEEP	0x00000000
0x00025520	thread_01	12	SLEEP	0x00000000
0x000255b4	thread_02	12	SUS_QUE	0x00000000
0x00025648	thread_03	8	READY	0x00000000
0x000256dc	thread_04	8	SUS_SEM	0x00000000
0x00025770	thread_05	5	SUS_EV	0x00000000
0x00025804	thread_06	8	READY	0x00000000
0x00025898	thread_07	8	SUS_MX	0x00000000
0x0002592c	thread_08	12	SLEEP	0x00000000
0x000259c0	UDP Send	12	SLEEP	0x00000000

Figure 6-2 Resource Viewer showing the thread list

3. Right-click on the required entry to display the context menu.

   For example, right-click on the `System Timer Thread` entry.

4. Select **Display Details** from the context menu to display the details of the chosen entry in the **Cmd** tab of the Output view. Figure 6-3 shows an example:



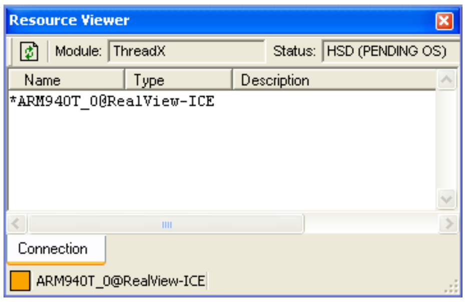
```
ThreadCB=0x00025520; ThreadTag=0x54485244; RunCount=416; Stack=0x00042734; StackStart=0x000423b8; StackEnd=0x000427b7; Stac
TimeSlice=4; NewTimeSlice=4; NextRdyTsk=0x000259c0; PrevRdyTsk=0x000255b4; Name=thread_01; Pri=12; State=SLEEP; SuspFlag=0x
TaskSusp=0x00000000; PreemptLvl=0x0000000c; PriortyBit=0x00001000; EntryPoint=0x000015b4; EntryArg=0x00000001; RemTics=32;
T_0_Func=0x00006568; T_0_Param=0x00025520; NextTimer=0x0002556c; PrevTimer=0x0002556c; HeadTimLst=0x0002dee8; C_U_Func=0x0C
TskSuspCnt=0x00000000; NxtSuspBlk=0x00000000; PrevSuspBlk=0x00000000; SuspInfo=0x00000000; AddInfo=0x00000000; SuspOpt=0x0C
SuspStatus=0x00000000; PtrNext=0x000255b4; PtrPrev=0x0002548c;
---
Run>
```

Figure 6-3 Thread details in the Output view

6.3 Viewing details in a Properties dialog box

To see the details of an entry in a Properties dialog box:

1. Select **View** → **Resource Viewer** from the Code window main menu to open the Resource Viewer. Figure 6-4 shows an example:



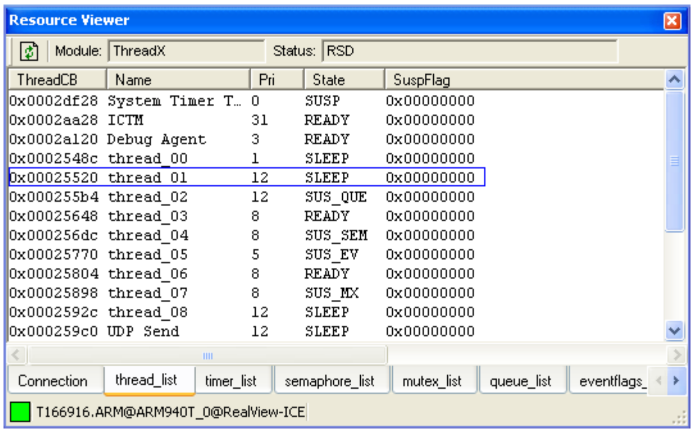
Name	Type	Description
*ARM940T_0@RealView-ICE		

ARM940T_0@RealView-ICE

Figure 6-4 Resource Viewer

2. Select the tab containing the entities that you want to view.

   For example, to see the threads, select the **task_list** tab. Figure 6-5 shows an example:



ThreadCB	Name	Pri	State	SuspFlag
0x0002df28	System Timer T…	0	SUSP	0x00000000
0x0002aa28	ICTM	31	READY	0x00000000
0x0002a120	Debug Agent	3	READY	0x00000000
0x0002548c	thread_00	1	SLEEP	0x00000000
0x00025520	thread_01	12	SLEEP	0x00000000
0x000255b4	thread_02	12	SUS_QUE	0x00000000
0x00025648	thread_03	8	READY	0x00000000
0x000256dc	thread_04	8	SUS_SEM	0x00000000
0x00025770	thread_05	5	SUS_EV	0x00000000
0x00025804	thread_06	8	READY	0x00000000
0x00025898	thread_07	8	SUS_MX	0x00000000
0x0002592c	thread_08	12	SLEEP	0x00000000
0x000259c0	UDP Send	12	SLEEP	0x00000000

T166916.ARM@ARM940T_0@RealView-ICE

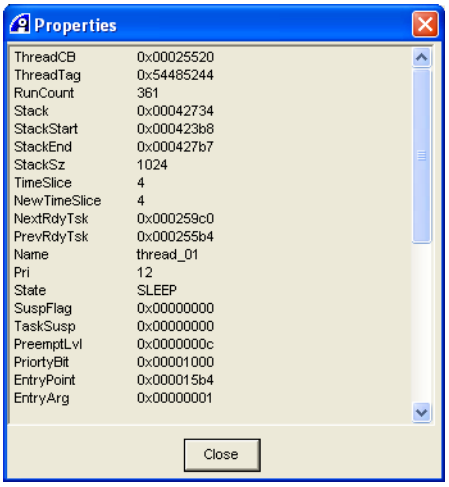
Figure 6-5 Resource Viewer showing the thread list

3. Right-click on an entry in the appropriate tab to display the context menu.

4. If you want to display the details in a Properties dialog box, then select **Display Details as Property** from the context menu.

5. Right-click on the entry in the appropriate tab to display the context menu.

6. Select **Display Details** from the context menu to display the details of the chosen entry. The details are displayed in a Properties dialog box. Figure 6-6 on page 6-5 shows an example:

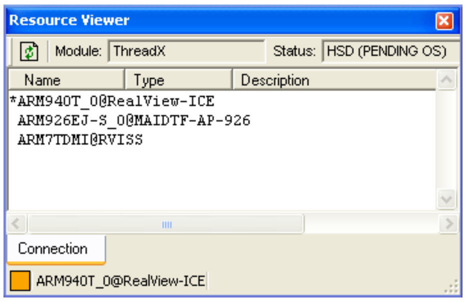


ThreadCB	0x00025520
ThreadTag	0x54485244
RunCount	361
Stack	0x00042734
StackStart	0x000423b8
StackEnd	0x000427b7
StackSz	1024
TimeSlice	4
NewTimeSlice	4
NextRdyTsk	0x000259c0
PrevRdyTsk	0x000255b4
Name	thread_01
Pri	12
State	SLEEP
SuspFlag	0x00000000
TaskSusp	0x00000000
PreemptLvl	0x0000000c
PriortyBit	0x00001000
EntryPoint	0x000015b4
EntryArg	0x00000001

Figure 6-6 Thread details in the Properties dialog box

6.4 Viewing the active connections

In multiprocessor debugging mode, the **Connection** tab of the Resource Viewer lists all active connections. Figure 6-7 shows an example:



```
*ARM940T_0@RealView-ICE
 ARM926EJ-S_0@MAIDTF-AP-926
 ARM7TDMI@RVISS
```

Figure 6-7 Active connection in the Resource Viewer

The asterisk indicates the current connection.

See also:

- the following in the *RealView Debugger User Guide*:
  - Chapter 7 *Debugging Multiprocessor Applications*.

6.5 Updating the Resource Viewer

You can update the Resource Viewer manually or have RealView Debugger update it automatically when the target stops.

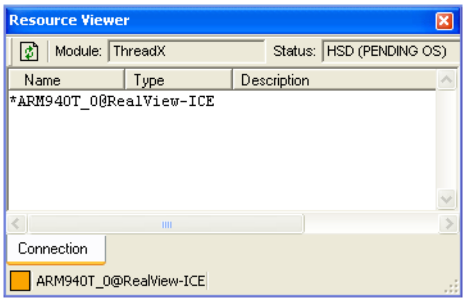
See also:

- *Toggling automatic updates of the Resource Viewer*
- *Manually updating the selected Resource list in the Resource Viewer* on page 6-8
- *Automatically updating the displayed details for a selected resource* on page 6-8.

6.5.1 Toggling automatic updates of the Resource Viewer

By default, the Resource Viewer updates automatically. To toggle the automatic update of the Resource Viewer:

1. Select **View → Resource Viewer** from the Code window main menu to open the Resource Viewer. Figure 6-8 shows an example:



*ARM940T_0@RealView-ICE

ARM940T_0@RealView-ICE

Figure 6-8 Resource Viewer

2. Right-click in the Resource Viewer to display the context menu.
3. Select **Auto Update** from the context menu.

   Note

   If automatic update disabled, then for a processor running in HSD, the OS resource tabs show the state before you started that processor.

Automatic updating refreshes the Resource Viewer as soon as execution stops at a point in your image, such as at a breakpoint.

Freezing the contents of the Resource Viewer

To freeze the contents of the Resource Viewer, disable automatic updating. You can manually update the view contents if required.

Freezing the contents of the Resource Viewer enables you to compare the contents with those in a second Resource Viewer.

See also

- *Manually updating the selected Resource list in the Resource Viewer* on page 6-8
- *Updating windows and views when a breakpoint activates* on page 12-4.

6.5.2 Manually updating the selected Resource list in the Resource Viewer

To manually update the selected Resource list in the Resource Viewer:

1. Select **View → Resource Viewer** from the Code window main menu to open the Resource Viewer. Figure 6-8 on page 6-7 shows an example.
2. Click the tab for the required Resource list.
3. Disable automatic updates.
4. Right-click in the Resource Viewer to display the context menu.
5. Select **Update View** from the context menu. The Resource Viewer is updated to reflect the current state.

You can also update the Resource list using the following methods:

- Click the tab for the appropriate resource.
- Click **Update View** on the toolbar.

If you click on the **Connection** tab, choose this option to update the status of the connections. This might be required if you change the current connection when the related Code window is attached to a connection.

See also

- *Toggling automatic updates of the Resource Viewer* on page 6-7.

6.5.3 Automatically updating the displayed details for a selected resource

To automatically update the displayed details for a selected resource when the related processor stops:

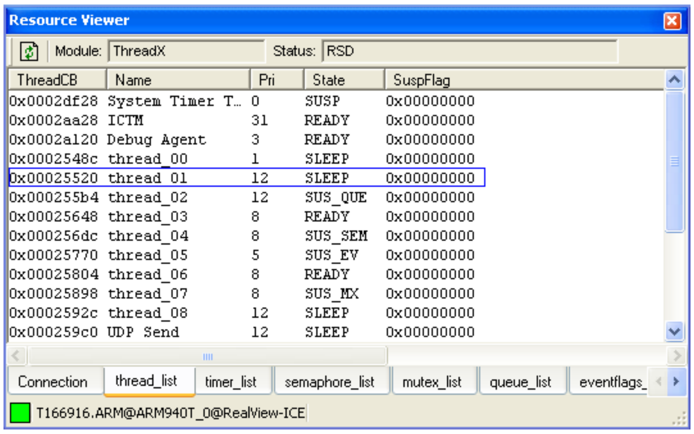
1. Select **View → Resource Viewer** from the Code window main menu to open the Resource Viewer. Figure 6-8 on page 6-7 shows an example.
2. Click the tab for the required Resource list.
3. Right-click on the required entry in the Resources list to display the context menu.
4. Select **Auto Update Details on Stop** from the context menu.

   ——— **Note** ———

   In multiprocessor debugging mode, this applies across all connections.

   ————————

5. Select the resource entry that you want to monitor. The selected resource is highlighted with a blue box. Figure 6-9 on page 6-9 shows an example:



```
ThreadCB   Name           Pri  State    SuspFlag
0x0002df28 System Timer T… 0   SUSP     0x00000000
0x0002aa28 ICTM            31  READY    0x00000000
0x0002a120 Debug Agent     3   READY    0x00000000
0x0002548c thread_00       1   SLEEP    0x00000000
0x00025520 thread_01       12  SLEEP    0x00000000
0x000255b4 thread_02       12  SUS_QUE  0x00000000
0x00025648 thread_03       8   READY    0x00000000
0x000256dc thread_04       8   SUS_SEM  0x00000000
0x00025770 thread_05       5   SUS_EV   0x00000000
0x00025804 thread_06       8   READY    0x00000000
0x00025898 thread_07       8   SUS_MX   0x00000000
0x0002592c thread_08       12  SLEEP    0x00000000
0x000259c0 UDP Send        12  SLEEP    0x00000000
```

🟩 T166916.ARM@ARM940T_0@RealView-ICE

Figure 6-9 Select resource in the Resource Viewer

When the target stops executing, the details of the selected resource are updated.

——— Note ———

How the details are displayed depends on how you have chosen to view them.

———————————

See also

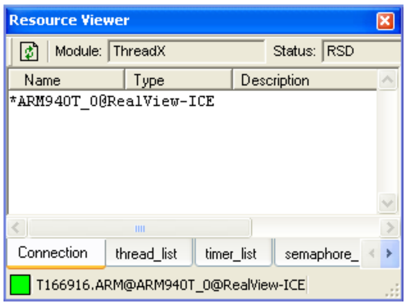
- *Viewing details in the Output view* on page 6-3
- *Viewing details in a Properties dialog box* on page 6-4.

6.6 Performing OS-specific actions on an OS-aware connection

Where supported by your Debug Agent, you can perform actions on a specified OS resource from the Resource Viewer.

To perform an action on a specific OS resource:

1. Select **View** → **Resource Viewer** from the Code window main menu to open the Resource Viewer. Figure 6-10 shows an example:



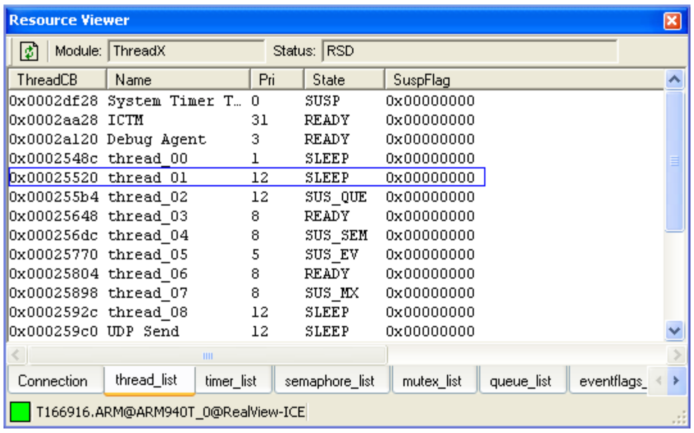
Module: ThreadX Status: RSD

Name	Type	Description
*ARM940T_0@RealView-ICE		

T166916.ARM@ARM940T_0@RealView-ICE

Figure 6-10 Resource Viewer (RSD)

2. Select the tab containing the Resources list that you want to view, for example the **thread_list** tab. Figure 6-11 shows an example:



Module: ThreadX Status: RSD

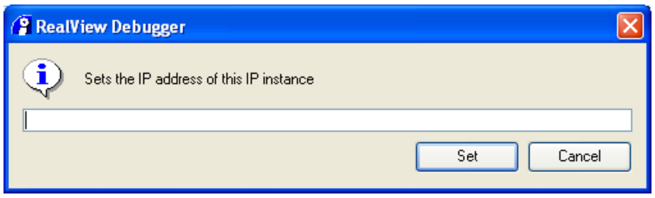
ThreadCB	Name	Pri	State	SuspFlag
0x0002df28	System Timer T…	0	SUSP	0x00000000
0x0002aa28	ICTM	31	READY	0x00000000
0x0002a120	Debug Agent	3	READY	0x00000000
0x0002548c	thread_00	1	SLEEP	0x00000000
0x00025520	thread_01	12	SLEEP	0x00000000
0x000255b4	thread_02	12	SUS_QUE	0x00000000
0x00025648	thread_03	8	READY	0x00000000
0x000256dc	thread_04	8	SUS_SEM	0x00000000
0x00025770	thread_05	5	SUS_EV	0x00000000
0x00025804	thread_06	8	READY	0x00000000
0x00025898	thread_07	8	SUS_MX	0x00000000
0x0002592c	thread_08	12	SLEEP	0x00000000
0x000259c0	UDP Send	12	SLEEP	0x00000000

T166916.ARM@ARM940T_0@RealView-ICE

Figure 6-11 Resources list in the Resource Viewer

3. Right-click on a Resources list entry to see the associated actions at the bottom of the context menu. The available actions and associated parameters depend on the Debug Agent and the Resources list you are viewing. For example, right-click on a specific thread to see the associated actions on the context menu. In this example, the Debug Agent offers the possible actions:
   - delete
   - suspend
   - resume.

   If you select an action from the context menu that requires parameters, a prompt is displayed. Figure 6-12 on page 6-11 shows an example:



Sets the IP address of this IP instance

Figure 6-12 Action argument prompt

——— **Note** ———

For information about the meaning of actions in the context menu, see the user manual for your OS.

———

4. Enter the required value.
5. Click **Set** to set the value.

In this example, you have to update the Resource Viewer to see the effect of this action.

The Resources list does not differentiate between captive threads and non-captive threads. If you try to perform an action on a non-captive thread, RealView Debugger displays an error message to inform you that the action is not available.

See also:

- *Captive threads* on page 4-6
- *Interaction of OS resources and RealView Debugger* on page 6-12
- the following in the *RealView Debugger Command Line Reference Guide*:
  - Chapter 2 *RealView Debugger Commands*.

6.7 Interaction of OS resources and RealView Debugger

Be aware of the following interactions between OS resources and RealView Debugger:

- The thread status, as shown in the Resource Viewer, is fully integrated with the thread list as shown in the Code window.
- OS resource actions and target execution control are independent. Using actions to suspend or resume threads is independent of thread control in RealView Debugger:
  - If you suspend a specific thread, that thread is not captive in the debugger. The Resource Viewer shows the thread as suspended, but the Code window does not show that the thread has stopped, that is, it is shown as running in the Registers view or in the Call Stack view.
  - If you resume a captive thread, the Resource Viewer shows the thread as running, but the debugger still believes that it has the thread captive. To update the debugger state, you must disconnect and then reconnect.
  - If you stop a thread from the Code window, using an action to resume does not show the thread as running, that is it still appears to be stopped.

Chapter 7
Debugging Your OS Application

This chapter describes features specific to debugging multithreaded images in RealView® Debugger. It contains the following sections:

Note

For full details on how to debug your applications using RealView Debugger, see the *RealView Debugger User Guide*.

7.1 About breakpoints

If you are in *Running System Debug* (RSD) mode, two types of breakpoint are available:

HSD breakpoint

This is the default breakpoint type offered by RealView Debugger. When it is hit, the breakpoint triggers and stops the processor. If you are in RSD mode when the *Halted System Debug* (HSD) breakpoint is hit, RealView Debugger changes into HSD mode.

RSD breakpoint

Any thread that hits this breakpoint stops immediately. There are different types of RSD breakpoint:

System breakpoint

This breakpoint is set by the Debug Agent and so requires RSD to be enabled. Any thread might trigger this breakpoint.

When it is triggered, a system breakpoint stops the thread that hit it, but all other threads continue.

Thread breakpoint

This breakpoint is set by the Debug Agent and so requires RSD to be enabled. A thread breakpoint is associated with a thread ID or a set of IDs, called a *break trigger group*. If any thread that is part of the break trigger group hits this breakpoint, it triggers and the thread stops. All other threads continue.

If the thread breakpoint is hit by a thread that is not part of the break trigger group, the breakpoint is not triggered and execution continues.

Process breakpoint

Not available in this release.

See also:
- *Using the break trigger group*
- *More on breakpoints* on page 7-3.

7.1.1 Using the break trigger group

The break trigger group consists of a thread ID, or a set of thread IDs, associated with a specific thread breakpoint. If any thread in the break trigger group hits the thread breakpoint, it triggers and the thread stops. All other threads, including the other threads in the break trigger group, continue.

The break trigger group is *empty* when all the threads in the group have ceased to exist. In this case, the group *disappears*. Even where the Debug Agent has the ability to communicate this information, the thread breakpoint associated with this empty break trigger group is not disabled. However, it never triggers.

If you try to reinstate a thread breakpoint where the break trigger group has disappeared, you cannot be sure that the threads specified by the group still exist or that the IDs are the same. In this release of RealView Debugger it is not possible to reinstate a thread breakpoint whose break trigger group has disappeared.

Note

A break trigger group can consist of a process, or set of processes, associated with a process breakpoint. This feature is not available in this release.

7.1.2 More on breakpoints

With OS support enabled, any breakpoint can also be a conditional breakpoint. RSD breakpoints can take the same qualifiers as HSD breakpoints and it is possible to link a counter or an expression to an RSD breakpoint.

You can also set hardware breakpoints in RSD mode but the availability of such breakpoints is determined by the debug target, that is the target processor and the Debug Agent. Hardware breakpoints are not integrated with the Debug Agent and so behave the same way in both HSD and RSD mode, their effect being to stop the target. When this occurs the communications with the Debug Agent is unavailable, and so RSD becomes inactive.

Viewing breakpoint support

To see your support for breakpoints:

- Select **Debug → Breakpoints → Hardware → Show Break Capabilities of HW...** from the Code window main menu to open an information dialog box describing the support available for your target processor.
- Select **Properties** from the **Process** tab context menu to open an information dialog box describing the Debug Agent support for breakpoints.

Memory maps and breakpoints

Where the memory map is disabled, RealView Debugger always sets a software breakpoint where possible. However, if the target is running in RSD mode, RealView Debugger sets a system breakpoint.

Where the memory map is enabled, RealView Debugger sets a breakpoint based on the access rule for the memory at the chosen location:

- a hardware breakpoint is set for areas of no memory (`NOM`), `Auto`, read-only (`ROM`), or Flash.
- if the memory is write-only (`WOM`), or where an error is detected, RealView Debugger gives a warning and displays the Create Breakpoint dialog box for you to specify the breakpoint details.

See also

- *Captive threads* on page 4-6
- *OS marker in the Process tab* on page 5-3
- the following in the *RealView Debugger User Guide*:
  - *Viewing the memory map* on page 9-8
  - Chapter 11 *Setting Breakpoints*
  - Chapter 12 *Controlling the Behavior of Breakpoints*.

7.2 Setting breakpoints

When you are debugging a multithreaded image, set breakpoints in the usual way, such as:

- by right-clicking inside the **Disassembly** tab or a source file tab
- using the Create Breakpoint dialog box
- using context menus from the Break/Tracepoints view
- submitting CLI commands.

See also:

- *Procedure for setting a breakpoint in the code view*
- *Breakpoint markers in the code views* on page 7-5
- *Changing the break trigger group* on page 7-6.

7.2.1 Procedure for setting a breakpoint in the code view

To set a breakpoint in your code view:

1. Start RealView Debugger.
2. Configure OS support and load the multithreaded image.
3. Start the image running in RSD mode until you reach a point where threads are being rescheduled.
4. Make sure that you are working in an unattached Code window so that the current thread is visible.
5. In the required source file tab, right-click in the gray area to the left of a source line to see the context menu.

   ——— **Note** ———

   The options available on the context menu depend on your debug target and Debug Agent. Where RSD or HSD is disabled, some options are grayed out.

   ————————
6. Select the required breakpoint from the list of options. That is, select:
   - **Set HSD Break** to set an HSD breakpoint
   - **Set System Break** to set a system breakpoint
   - **Set Thread Break** to set a thread breakpoint.

The **Cmd** tab in the Output view shows the breakpoint command, for example:

```
bi,rtos:thread \DEMO\#373:1 = 0x13BAC
```

——— **Note** ———

Process breakpoints are not available in this release.

————————

7.2.2 Breakpoint markers in the code views

Breakpoints are marked in the source-level and disassembly-level view at the left side of the window using color-coded icons:

- Red means that a breakpoint is active, and that it is in scope.
- Green means that a breakpoint is active, but not for the thread, or process, that is shown in the Code window. The thread shown in the Code window depends on the window attachment.
- Yellow shows a conditional breakpoint.

  Note

  Conditional thread breakpoints are not colored yellow. They are colored according to the current context (that is, red or green).

- White shows that a breakpoint is disabled.

Table 7-1 shows the icons used for the breakpoint types.

Table 7-1 Breakpoint icons for OS-aware applications

Icon	Breakpoint/Tracepoint type
●	*Halted System Debug* (HSD) breakpoint.
⧗	Thread breakpoint in *Running System Debug* (RSD).
■	RSD System breakpoint.

If you set a thread breakpoint in this way in an unattached window, the break trigger group is the current thread. If your Code window is attached to a thread, the break trigger group consists of the thread the window is attached to.

See also

- *Attaching windows to threads* on page 4-6.

7.2.3 Changing the break trigger group

If you have RSD enabled and you set a thread breakpoint, right-click on this breakpoint, marked by a thread icon, to see a context menu.

Note

The options available on the context menu depend on your debug target and Debug Agent. Where RSD or HSD is disabled, some options are grayed out.

This context menu contains options related to the break trigger group:

- **Add This Thread**
- **Remove This Thread**
- **Break Trigger Group...**

Note

These options are not available in this release. This means that you cannot change the threads that make up the group after the breakpoint is set from the Code window. Instead use the appropriate CLI command to modify the breakpoint.

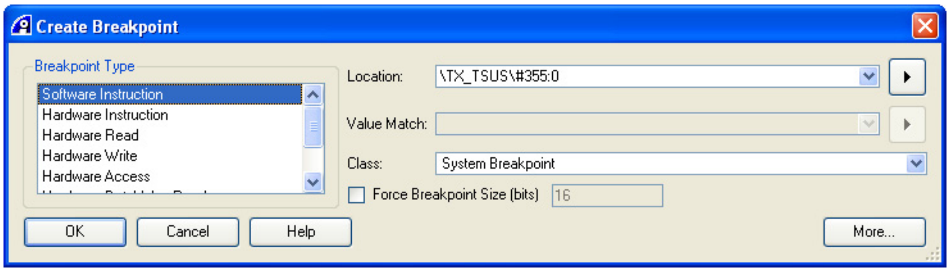
See also

- *OS-aware CLI commands* on page 1-9.

7.3 Using the Create Breakpoint dialog box

Use the Create Breakpoint dialog box to set breakpoints:

1. Right-click in the gray area to the left of a line to see the context menu.
2. Select **Create Breakpoint...** to open the Create Breakpoint dialog box. Figure 7-1 shows an example:



Location: \TX_TSUS\#355:0

Value Match:

Class: System Breakpoint

Force Breakpoint Size (bits) 16

Figure 7-1 OS Breakpoint Class selector

3. Use the Class field to specify the type of breakpoint to set. A system breakpoint is the default choice.

   Breakpoint classes are not listed if they are not supported by the Debug Agent.

The breakpoint Class options available depend on:

- hardware capability
- Debug Agent
- HSD/RSD status, for example if RSD is not available, the field shows `Standard Breakpoint`
- the `System_Stop` setting specified in your board file
- whether you are using tracepoints.

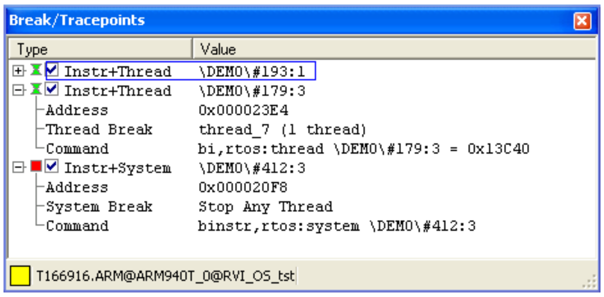
Depending on the type of breakpoint, you cannot edit an existing breakpoint where the choice is restricted by the Debug Agent. In this case, you must clear the breakpoint before you can set a new breakpoint at the same location.

See also:

- the following in the *RealView Debugger User Guide*:
  - Chapter 11 *Setting Breakpoints*
  - Chapter 12 *Controlling the Behavior of Breakpoints*.

7.4 Using the Break/Tracepoints view

Select **View** → **Break/Tracepoints** from the Code window main menu to open the Break/Tracepoints view. Figure 7-2 shows an example:

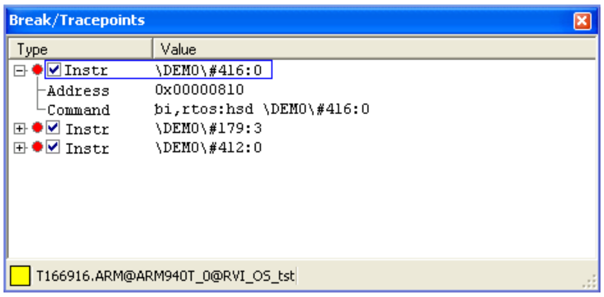


```
Break/Tracepoints
Type                Value
⊞ ✓ Instr+Thread    \DEMO\#193:1
⊟ ✓ Instr+Thread    \DEMO\#179:3
   Address          0x000023E4
   Thread Break     thread_7 (1 thread)
   Command          bi,rtos:thread \DEMO\#179:3 = 0x13C40
⊟ ✓ Instr+System    \DEMO\#412:3
   Address          0x000020F8
   System Break     Stop Any Thread
   Command          binstr,rtos:system \DEMO\#412:3
T166916.ARM@ARM940T_0@RVI_OS_tst
```

Figure 7-2 OS (RSD) breakpoints in the Break/Tracepoints view

Use this view to see the current breakpoints, or to enable, or disable, a chosen breakpoint. Breakpoints are marked using color-coded icons in the same way as in the **Disassembly** tab and a source file tab. You can also use this view to edit breakpoints using the Edit Breakpoint dialog box.

The Break/Tracepoints view shows details about each breakpoint you set. What is shown depends on whether you are working in RSD or HSD mode. Figure 7-2 shows that three breakpoints are set in RSD mode. Here, two thread breakpoints are active on background threads, and a system breakpoint is active on the current thread. Figure 7-3 shows three breakpoints are set in HSD mode. Here, the extra breakpoint details (available in RSD) are not included in the Break/Tracepoints view.



```
Break/Tracepoints
Type            Value
⊟ ● ✓ Instr     \DEMO\#416:0
   Address      0x00000810
   Command      bi,rtos:hsd \DEMO\#416:0
⊞ ● ✓ Instr     \DEMO\#179:3
⊞ ● ✓ Instr     \DEMO\#412:0
T166916.ARM@ARM940T_0@RVI_OS_tst
```

Figure 7-3 OS (HSD) breakpoints in the Break/Tracepoints view

In these examples, the Break/Tracepoints view is floating and so the view title bar reflects the title bar of the calling Code window. Figure 7-2 and Figure 7-3 show that the Code window is unattached. In this case, the Code window displays the current thread.

See also:

- *Setting breakpoints* on page 7-4
- the following in the *RealView Debugger User Guide*:
  - *Viewing breakpoint information* on page 11-21
  - *Editing a breakpoint* on page 11-26.

7.5 Stepping threads

Use the Execution group from the Debug toolbar to control program execution, shown in Figure 7-4.

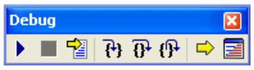


Figure 7-4 Debug toolbar

For example, to start and stop execution, and to step through a multithreaded application. These options are also available from the main **Debug** menu.

When you are debugging a multithreaded image, stepping behavior depends on the:

- current thread
- thread you are stepping through
- windows attachment.

See also:

- *Stepping in RSD mode*
- *Stepping in HSD mode* on page 7-10.

7.5.1 Stepping in RSD mode

When the processor is in RSD mode, you can step any thread independently without having to stop the target. However, you must stop the thread that you want to step, for example using a system, thread, or process breakpoint.

Note

Process breakpoints are not available in this release.

If you are using an unattached Code window then you can step the current thread in the usual way. The code view changes, however, if breakpoints are hit on other background threads while you are stepping. This is because a stopped thread becomes the current thread and is visible in the unattached window.

The example shown in Figure 7-5 represents three threads at different stages of execution.

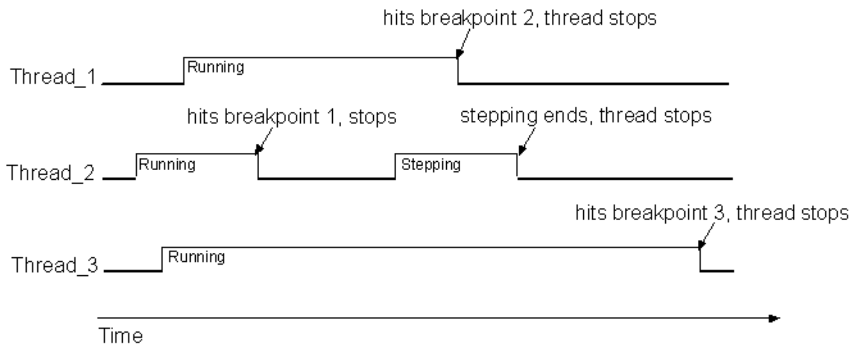

hits breakpoint 2, thread stops
Thread_1
Running
hits breakpoint 1, stops
stepping ends, thread stops
Thread_2
Running
Stepping
hits breakpoint 3, thread stops
Thread_3
Running
Time

Figure 7-5 Stepping and stopping threads

In an unattached window, where `Thread_2` is the current thread, the code view shows:

1. `Thread_2`, as stepping starts and code is examined.

2. `Thread_1`, when breakpoint `2` hits.
3. `Thread_2`, as stepping ends and the thread stops.
4. `Thread_3`, when breakpoint `3` hits.

In a window attached to `Thread_2`, the code view shows `Thread_2` as stepping completes and the thread stops. In this case:

- Any stop events on `Thread_1` or `Thread_3` are not visible in the Code window.
- The current thread changes as breakpoints are hit. It is:
  1. `Thread_2`, when breakpoint `1` hits.
  2. `Thread_1`, when breakpoint `2` hits.
  3. `Thread_2`, as the debugger internal breakpoint hits when stepping ends.
  4. `Thread_3`, when breakpoint `3` hits.

If you want to examine a background thread you must do one of the following:
- make the background thread the current thread
- attach your Code window to the background thread
- open a new Code window and attach the window to the background thread.

7.5.2 Stepping in HSD mode

When the processor is in HSD mode, you must set a breakpoint and stop your image so that you can step through the code. When you are working on a multithreaded image, any step instruction acts on the thread that was current when the processor stopped.

7.6 Adding object search paths and mappings

If RealView Debugger is unable to locate an object file for your image, you can add the location of the objects to the object file search path.

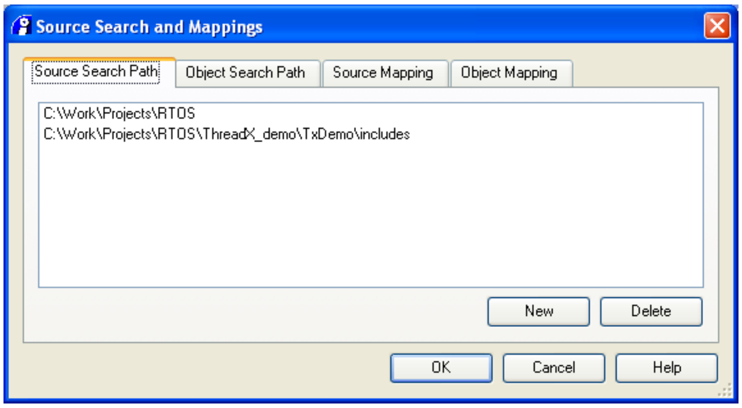
See also:

- *Adding an object search path*
- *Adding an object file mapping* on page 7-12.

7.6.1 Adding an object search path

To add an object search path:

1. Select **Debug → Set Source Search Path...** from the Code window main menu to open the Source Search and Mappings dialog box. Figure 7-6 shows an example:



C:\Work\Projects\RTOS
C:\Work\Projects\RTOS\ThreadX_demo\TxDemo\includes

Figure 7-6 Source Search and Mappings dialog box

2. Select the **Object Search Path** tab. Figure 7-7 shows an example:

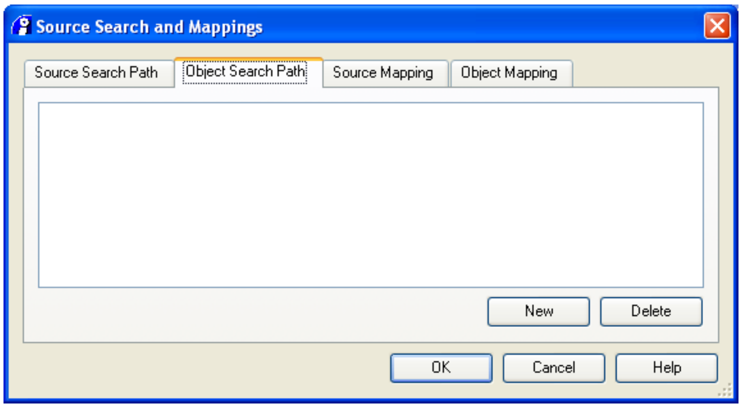


Figure 7-7 Object Search Path tab

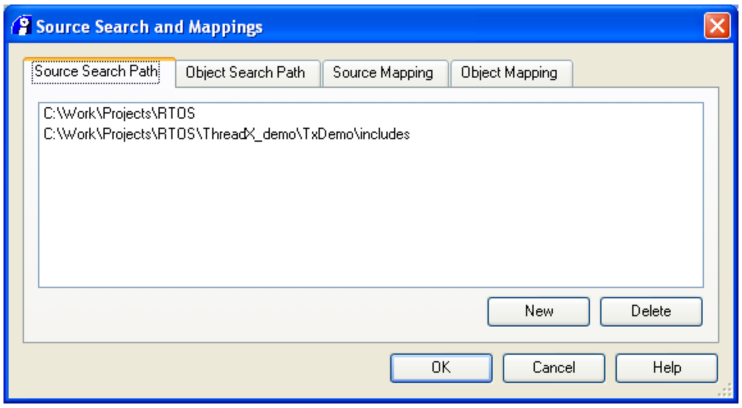
3. Click **New** to open the Browse For Folder dialog box.
4. Locate the directory containing the source files.
5. Click **OK**. The object search path is added to the list of paths.
6. If your object files are distributed between a number of directories, then repeat the previous steps for each directory.
7. When you have located all the search paths for the image, click **OK** to close the Source Search and Mappings dialog box.

The search paths that you define are saved in a settings file that is stored in the same location as the image. This file has the same name as your image, and has the `.apr` file extension. For example, if your image file is called `image.axf` image, the settings file is called `image.axf.apr`. The settings in this file are re-applied when you next load the image.

7.6.2 Adding an object file mapping

To add an object file mappings:

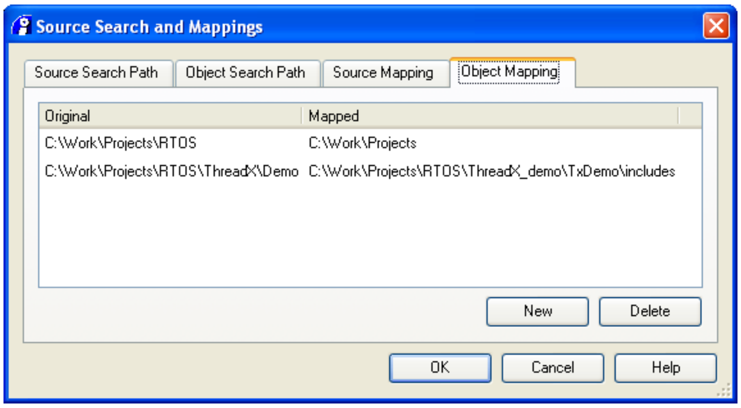
1. Select **Debug → Set Source Search Path...** from the Code window main menu to open the Source Search and Mappings dialog box. Figure 7-8 shows an example:



C:\Work\Projects\RTOS
C:\Work\Projects\RTOS\ThreadX_demo\TxDemo\includes

Figure 7-8 Object Search Path tab

2. Select the **Object Mapping** tab. Figure 7-9 shows an example:



Original	Mapped
C:\Work\Projects\RTOS	C:\Work\Projects
C:\Work\Projects\RTOS\ThreadX\Demo	C:\Work\Projects\RTOS\ThreadX_demo\TxDemo\includes

Figure 7-9 Object Mapping tab

3. Click **New** to open the New Mapping dialog box. Figure 7-10 shows an example:

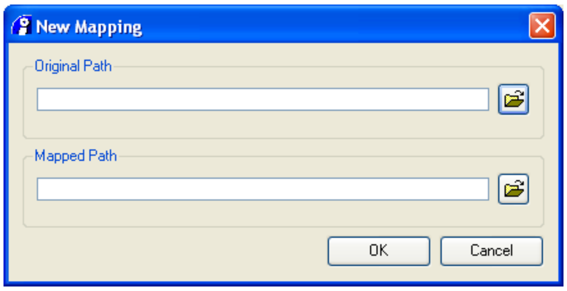


Figure 7-10 New Mapping dialog box

4. Specify the Original Path (that is, the path to the original object tree for the image). To do this, either:
   - enter the path name in the field
   - click the open file button to locate the required file.
5. Specify the Mapped Path (that is, the path to the new object tree for the image). To do this, either:
   - enter the path name in the field
   - click the open file button to locate the required file.
6. Click **OK**. The object mapping is added to the list of mappings.
7. If your object files are distributed between a number of directories, then repeat the previous steps for each directory.
8. When you have specified all the object mappings for the image, click **OK** to close the Source Search and Mappings dialog box.

The object mappings that you define are saved in a settings file that is stored in the same location as the image. This file has the same name as your image, and has the `.apr` file extension. For example, if your image file is called `image.axf` image, the settings file is called `image.axf.apr`. The settings in this file are re-applied when you next load the image.

7.7 Manipulating registers and variables

Select **View** → **Registers** from the Code window main menu to open the Registers view where you can view registers for threads in the system. If the Code window is unattached, the Registers view shows processor registers for the current thread.

If you attach a Code window to a specified thread, the Registers view displays the registers associated with the thread. These might not have the same values as the current processor registers.

You can use in-place editing to change a register value in the usual way. However, you can only see register values, and change them, when the thread is stopped. The new register values are written to the OS *Task Control Block* (TCB) for the selected thread. When that thread is next scheduled, the registers used by the thread are read from the TCB into the processor.

If you are debugging ARM code, the *ARM Architecture Procedure Call Standard* (AAPCS) specifies that the first four parameters to a function are passed in registers. In addition, some local variables are optimized into registers by the compiler for parts of the function. Therefore if you modify a local variable that is stored in a register, the debugger modifies the TCB state to transfer the value into a processor register instead of modifying the target memory allocated to that variable.

Note

If you are modifying a value that you expect to be shared by several threads, for example a global variable, the compiler might have cached that value in a register for one or more of the threads. As a result, the modification you want is not propagated to all of the threads that reference the variable. To make sure that such modifications operate correctly, you must do one of the following:

- in RSD mode, modify the variable, then at the point you have stopped the relevant thread, if any thread has a cached copy of the variable, modify the copy
- in HSD mode, modify the variable, then at the point you have stopped the processor, if any thread has a cached copy of the variable, modify the copy
- in RSD or HSD, declare the variable to be `volatile` and recompile the program.

See also:

- the following in the *RealView Debugger User Guide*:
  - Chapter 14 *Altering the Target Execution Environment*.

7.8 Updating your debug view

During your debugging session, use the following views to monitor execution:

- The Call Stack view displays the call stack.
- The Locals view displays any local, static, and C++ `this` variables that are in scope.
- The Memory view displays the contents of memory and enables you to change those contents. When you first open the view no memory locations are displayed, because no starting address has been specified. If you enter a starting address, values are updated to correspond to the current image status.
- The Watch view enables you to view expressions and their current values, or to change existing watched values.

In these views, you can use the view context menu to specify how the contents are updated:

Update Window

If you have deselected the option **Automatic Update**, you can use this option to update the thread view manually. You can update the display using this option at any time. This enables you to catch any memory updates made externally.

Automatic Update

Updates the display automatically, that is when:

- you change memory from anywhere in RealView Debugger
- a watched value changes
- program execution stops.

This is the default.

Timed Update when Running

If you are in RSD mode, the thread view can be updated at a specified time interval during program execution. Select this option to set this timer according to the update period specified by **Timed Update Period**.

Timed Update Period

Use this to choose the interval, in seconds, between window updates.

Any value you enter here is only used when the option **Timed Update when Running** is enabled.

See also:

- the following in the *RealView Debugger User Guide*:
  - Chapter 13 *Examining the Target Execution Environment*
  - Chapter 14 *Altering the Target Execution Environment*.

7.9 Synchronization behavior when using OS-awareness in RSD mode

If you are running images on multiple processors that are synchronized, any actions performed on one image might affect the running state of the images on the other processors. The running state depends on:

- How you have set up synchronization and cross-triggering for the different processors.
- Whether an image is running on an OS-aware connection. The details are described in this section.

See also:

- *Synchronization behavior when running on an OS-aware connection*
- *Controlling connections running in RSD mode* on page 7-17.

7.9.1 Synchronization behavior when running on an OS-aware connection

If you are running multiple images on a connection that has OS-awareness configured, the behavior of the `HALT`, `GO` and `STOP` commands (or the **Run** and **Stop** buttons) can affect the running state of your other images. The behavior depends on whether or not the connections are configured to run and stop synchronously, and whether or not *Running System Debug* (RSD) is active on one or more of the connections. Connections that are not configured to be synchronous must be started and stopped independently.

The following sections assume that synchronization is configured as follows:

- processors synchronized, without cross-triggering
- run and stop operations enabled, step operation disabled.

RSD is active on the current connection

If RSD is active on the current connection:

- Clicking **Stop** or entering the `HALT` command without a thread identifier either:
  - — halts the currently running thread in the current connection, if the action is performed in an unattached Code window
  - — halts the thread shown, if the action is performed in a Code window attached to that thread.

  If you specify a thread identifier with the `HALT` command, then the specified thread is stopped.
- Clicking **Run** or entering the `GO` command either:
  - — restarts only the currently halted thread in the current connection
  - — causes the action to be pended.
- Entering the `STOP` command affects the targets on other connections that are participating in the synchronization:
  - — For the current connection, and any other connections that are running in RSD mode, the action depends on the `System_Stop` setting in the connection properties of the related Debug Configuration. This setting enables you to control what happens to the RSD connections.
  - — For any connection that is running in HSD mode, or is not configured for OS-awareness, the associated target stops.

——— **Note** ———

Be aware that synchronization control interacts with the current context for each synchronized connection. For an OS-aware connection with RSD active, the current context is the thread where execution has stopped (that is, the current thread). If a Code window is attached to a thread that is not the current thread, then you cannot see the interaction for the associated connection.

RSD is not active on the current connection

If RSD is not active on the current connection:

- Clicking **Stop** or entering the `HALT` command:
  - Halts the target in the current connection.
  - For any target that is running in RSD mode, the action depends on the `System_Stop` setting in the connection properties of the related Debug Configuration. This setting enables you to control what happens to the RSD connections.

    For the default, `Don't_Prompt`, the current thread stops.
- Clicking **Run** or entering the `GO` command:
  - Restarts target on the current connection.
  - For any target that is running in RSD mode, and `System_Stop` set to the default `Don't_Prompt` the currently halted thread restarts.

See also

- *Controlling connections running in RSD mode*
- the following in the *RealView Debugger User Guide*:
  - *Synchronizing multiple processors* on page 7-13.

7.9.2 Controlling connections running in RSD mode

You can use the `System_Stop` setting in connection properties to control what happens when a stop request is received on a connection running in RSD mode:

Never The processor never stops, and the request is ignored.

Don't prompt

The processor stops.

Prompt The following prompt is displayed:

```
The operation will stop the processor!
You will fall back into HSD mode!
Are you sure you want to proceed?
```

If you answer:

Yes This causes the processor on the connection to drop back into *Halted System Debug* (HSD) mode, and the processor on that connection stops running.

No This causes the processor on the connection to remain running.

Table 7-2 summarizes what happens when you enter a `STOP` command (or click the **Stop** button) on a connection that is one of three connections synchronized for stop. For clarity, the processors are identified in the table as HSD, RSD_1, and RSD_2, to show the OS debugging mode for each processor. The bold text in parentheses is the user response to the prompt.

——— **Note** ———

These actions also occur if you enter the `HALT` command on the HSD-mode connection.

Table 7-2 Processor state after a STOP command

System_Stop setting		**Processor state after a STOP command**		
RSD_1	**RSD_2**	**HSD**	**RSD_1**	**RSD_2**
Prompt (**Yes**)	Prompt (**No**)	stopped	stopped	running
Prompt (**No**)	Prompt (**No**)	stopped	running	running
Prompt (**Yes**)	Prompt (**Yes**)	stopped	stopped	stopped
Never	Don’t_Prompt	stopped	running	stopped
Never	Prompt (**Yes**)	stopped	running	stopped
Never	Prompt (**No**)	stopped	running	running
Never	Never	stopped	running	running
Don’t_Prompt	Don’t_Prompt	stopped	stopped	stopped

See also

- the following in the *RealView Debugger User Guide*:
  - *Synchronizing multiple processors* on page 7-13.