ARM® DS-5™

Version 5.4

Using the Debugger

ARM®

ARM DUI 0446D (ID011411)

ARM DS-5
Using the Debugger

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
June 2010	A	Non-Confidential	First release for DS-5
September 2010	B	Non-Confidential	Update for DS-5 version 5.2
November 2010	C	Non-Confidential	Update for DS-5 version 5.3
January 2011	D	Non-Confidential	Update for DS-5 version 5.4

Proprietary Notice

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

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

Web Address

`http://www.arm.com`

Contents
ARM DS-5 Using the Debugger

Chapter 1
Conventions and feedback

The following describes the typographical conventions and how to give feedback:

Typographical conventions

The following typographical conventions are used:

`monospace` Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.

`mono`space 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.

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

bold Highlights interface elements, such as menu names. Also used for emphasis in descriptive lists, where appropriate, and for ARM® processor signal names.

Feedback on this product

If you have any comments and suggestions about this product, contact your supplier and give:

- your name and company

- the serial number of the product
- details of the release you are using
- details of the platform you are using, such as the hardware platform, operating system type and version
- a small standalone sample of code that reproduces the problem
- 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
- the version string of the tools, including the version number and build numbers.

Feedback on content

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

- the title
- the number, ARM DUI 0446D
- if viewing online, the topic names to which your comments apply
- if viewing a PDF version of a document, the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

ARM periodically provides updates and corrections to its documentation on the ARM Information Center, together with knowledge articles and *Frequently Asked Questions* (FAQs).

Other information

- *ARM Information Center*, `http://infocenter.arm.com/help/index.jsp`
- *ARM Technical Support Knowledge Articles*, http://infocenter.arm.com/help/topic/com.arm.doc.faqs
- *Support and Maintenance*, http://www.arm.com/support/services/support-maintenance.php.

Chapter 2
Getting started with the debugger

The following topics introduce you to some of the debugger concepts and explain how to launch the debugger.

Tasks

- *Launching the debugger from Eclipse* on page 2-6
- *Launching the debugger from the command-line console* on page 2-7.

Concepts

- *About the debugger* on page 2-2
- *Debugger concepts* on page 2-4.

Reference

- *DS-5 Debug perspective keyboard shortcuts* on page 2-10
- *DS-5 Debugger command-line console keyboard shortcuts* on page 2-11.

2.1 About the debugger

ARM® DS-5™ Debugger provides a powerful tool for debugging applications on both hardware targets and models using ARM architecture-based processors. You can have complete control over the flow of the execution so that you can quickly isolate and correct errors.

The following features are provided:

- loading images and symbols
- running images
- breakpoints and watchpoints
- source and instruction level stepping
- controlling variables, and register values
- controlling the call stack
- support for handling exceptions and Linux signals
- debug of multi-threaded Linux applications
- debug of Linux kernel modules, boot code and kernel porting.

The debugger supports a comprehensive set of DS-5 Debugger commands that can be executed in the Eclipse *Integrated Development Environment* (IDE), script files, or a command-line console. In addition there is a small subset of CMM-style commands sufficient for running target initialization scripts. CMM is a scripting language supported by some third-party debuggers. To execute CMM-style commands you must create a debugger script file containing the CMM-style commands and then use the DS-5 Debugger `source` command to run the script.

To help you get started, there are some tutorials that you can follow showing you how to run and debug applications using DS-5 tools.

2.1.1 See also

Tasks

- DS-5 tutorials in *Getting Started with DS-5*:
  - *Importing the example projects into Eclipse* on page 3-2
  - *Building the Gnometris project from Eclipse* on page 3-4
  - *Loading the Gnometris application on a Real-Time System Model* on page 3-6
  - *Loading the Gnometris application on to an ARM Linux target* on page 3-7
  - *Using an SSH connection to set up and run Gnometris on an ARM Linux target* on page 3-8
  - *Connecting to the Gnometris application that is already running on a ARM Linux target* on page 3-13
  - *Debugging Gnometris* on page 3-16
  - *Debugging a loadable kernel module* on page 3-17
- *Launching the debugger from Eclipse* on page 2-6
- *Launching the debugger from the command-line console* on page 2-7
- *ARM DS-5 Using Eclipse*:
  - *Integrating ARM plug-ins into a custom Eclipse environment* on page 2-3.

Concepts

- *Debugger concepts* on page 2-4
- *Types of target connections* on page 3-2.

Reference

- *ARM® DS-5™ Debugger Command Reference*:
  - Chapter 2 *DS-5 Debugger commands*

  - Chapter 3 *CMM-style commands supported by the debugger*.
- *ARM® DS-5™ Using Eclipse*:
  - Chapter 3 *Getting started with Eclipse*.

2.2 Debugger concepts

The following concepts are involved when debugging applications.

Debugger A debugger is software running on a host computer that enables you to make use of a debug agent to examine and control the execution of software running on a debug target.

Debug session

A debug session begins when you connect the debugger to a target or a model for debugging software running on the target and ends when you disconnect the host software from the target.

Debug target

At an early stage of product development there might be no hardware so the expected behavior of the hardware is simulated by software. This is referred to in the debugger documentation as a model. Even though you might run a model on the same computer as the debugger, it is useful to think of the target as a separate piece of hardware.

Alternatively, you can build a prototype product on a printed circuit board, including one or more processors on which you run and debug the application. This is referred to in the debugger documentation as a hardware target.

Debug agent A debug agent performs the actions requested by the debugger on the target, for example:

- setting breakpoints
- reading from memory
- writing to memory.

The debug agent is not the application being debugged, or the debugger itself.

Examples include:

- debug hardware agents:
  - ARM DSTREAM™ unit
  - ARM RVI™ unit.
- debug software agents:
  - *Real-Time System Model* (RTSM)
  - `gdbserver`.

Contexts Each processor in the target can have a process currently in execution. Each process uses values stored in variables, registers, and other memory locations. These values can change during the execution of the process.

The context of a process describes its current state, as defined principally by the call stack that lists all the currently active calls. The context changes when:

- a function is called
- a function returns
- an interrupt or an exception occurs.

Because variables can have class, local, or global scope, the context determines which variables are currently accessible. Every process has its own context. When execution of a process stops, you can examine and change values in its current context.

Scope The scope of a variable is determined by the point within an application at which it is defined. Variables can have values that are relevant within:

- a specific class only (*class*)

- a specific function only (*local*)
- a specific file only (*static global*)
- the entire application (*global*).

2.2.1 See also

Tasks

- *Configuring a connection to an RTSM model* on page 3-3
- *Configuring a connection to a Linux target using gdbserver* on page 3-5
- *Configuring a connection to a Linux Kernel* on page 3-7.
- *Configuring a connection to a bare-metal target* on page 3-9.

Concepts

- *Debugger concepts* on page 2-4.

Reference

- *ARM DSTREAM Setting up the Hardware*, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0481-.

2.3 Launching the debugger from Eclipse

To launch the debugger:

1. Launch Eclipse:
   - On Windows, select **Start → All Programs → ARM DS-5 → Eclipse for DS-5**.
   - On Linux:
     — If you installed the shortcut during installation, you can select **Eclipse for DS-5** in the **Applications** menu.
     — If you did not install the shortcut during installation:
       1. Add the *install_directory*/bin directory to your PATH environment variable. If it is already configured then you can skip this step.
       2. Open Unix bash shell.
       3. Enter `eclipse` at the prompt.
2. Select **Window → Open Perspective → DS-5 Debug** from the main menu.
3. If you have not run a debug session before then you must configure a connection between the debugger and the target before you can start any debugging tasks.
4. If you have run a debug session before then you can select a target connection in the Debug Control view and click on the **Connect to Target** toolbar icon.

2.3.1 See also

Tasks

- *Configuring a connection to an RTSM model* on page 3-3
- *Configuring a connection to a Linux target using gdbserver* on page 3-5
- *Configuring a connection to a Linux Kernel* on page 3-7
- *Configuring a connection to a bare-metal target* on page 3-9.

Concepts

- *Types of target connections* on page 3-2.

Reference

- *Debug Control view* on page 11-18
- *Commands view* on page 11-15
- *Breakpoints view* on page 11-8
- *Disassembly view* on page 11-22
- *Variables view* on page 11-48
- *Registers view* on page 11-36
- *Memory view* on page 11-30
- *Debug Configurations - Connection tab* on page 11-59.
- *ARM® DS-5™ Using Eclipse*:
  — *Perspectives and views* on page 3-19.

2.4 Launching the debugger from the command-line console

To launch the debugger:

1. Launch the DS-5 command-line console:

   On Windows, select **Start → All Programs → ARM DS-5 → DS-5 Command Prompt**.

   On Linux:

   a. Add the *install_directory*/bin directory to your PATH environment variable. If it is already configured then you can skip this step.

   b. Open a Unix bash shell.

2. Launch the debugger using the following command-line syntax:

   `debugger --target [--target_device] [option]…`

   Where:

   `--target=host:port|filename`

   For gdbserver connections you can specify the *host*:*port* for the connection between the debugger and gdbserver.

   A serial connection requires an XML file similar to the following example:

Example 2-1 mySerialConfig.xml

```
<?xml version="1.0"?>
<RVConfigUtility>
    <rddi type="rddi-debug-gdb"/>
    <rddigdb>
        <connection>
            <serial>
                <port>COM1</port>
                <speed>115200</speed>
            </serial>
        </connection>
    </rddigdb>
</RVConfigUtility>
```

   `--target_device=number|name`

   Specifies the device number or name. You must launch the debugger with --target_device command-line option when configuring a connection to a target containing multiple devices. If you do not specify --target_device then the debugger lists all the available devices and quits.

   and *option* can be any of the following:

   `--help` Displays a summary of the main command-line options.

   `--script=filename`

   Specifies a script file containing debugger commands to control and debug your target. You can repeat this option if you have several script files. The scripts are run in the order specified.

   `--interactive`

   Specifies interactive mode that redirects standard input and output to the debugger from the current command-line console, for example, Windows Command Prompt or Unix bash shell. This is the default if no script file is specified.

`--target_os`

Specifies the name of the target *Operating System* (OS), for example, `--target_os=linux`. This enables OS support within the debugger for example, shared library support.

`--stop_on_connect=true|false`

Specifies whether the debugger stops the target when it connects to the target device. To leave the target unmodified on connection you must specify `false`. The default is `--stop_on_connect=true`.

`--continue_on_error=true|false`

Specifies whether the debugger stops the target and exits the current script when an error occurs. The default is `--continue_on_error=false`.

`--image=`*`filename`*

Specifies the image file for the debugger to load when it connects to the target.

`--log_config=`*`option`*

Specifies the type of logging configuration to output runtime messages from the debugger.

Where:

`option` Specifies a predefined logging configuration or a user-defined logging configuration file:

- `info` Output messages using the predefined INFO level configuration. This level does not output debug messages. This is the default.
- `debug` Output messages using the predefined DEBUG level configuration. This option outputs both INFO level and DEBUG level messages.
- *`filename`* Specifies a user-defined logging configuration file to customize the output of messages. The debugger supports log4j configuration files.

`--log_file=`*`filename`*

Specifies an output file to receive runtime messages from the debugger. If this option is not used then output messages are redirected to the console.

`--top_mem=`*`address`*

Specifies the stack base, also known as the top of memory. Top of memory is only used for semihosting operations.

`--enable_semihosting`

Enables semihosting operations.

`--disable_semihosting`

Disables semihosting operations.

`--disable_semihosting_console`

Disables all semihosting operations to the debugger console.

`--semihosting_error=`*`filename`*

Specifies a file to write `stderr` for semihosting operations.

`--semihosting_input=`*`filename`*

Specifies a file to read `stdin` for semihosting operations.

`--semihosting_output=`*`filename`*

Specifies a file to write `stdout` for semihosting operations.

Note

Semihosting is used to communicate input/output requests from application code to the host workstation running the debugger.

2.4.1 See also

Task

- *Creating a debugger script file* on page 7-2
- *Controlling the output of logging messages* on page 8-7.

Concepts

- *Types of target connections* on page 3-2
- *About semihosting and top of memory* on page 8-2.

Reference

- *DS-5 Debugger command-line console keyboard shortcuts* on page 2-11
- *ARM® DS-5™ Debugger Command Reference*:
  - Chapter 2 *DS-5 Debugger commands*
  - Chapter 3 *CMM-style commands supported by the debugger*.

Other information

- *Log4j* in *Apache Logging Services*, `http://logging.apache.org`.

2.5 DS-5 Debug perspective keyboard shortcuts

When using the DS-5 Debug perspective, there are keyboard shortcuts that you can use.

In any view or dialog box you can access the dynamic help by using the following:

- On Windows, **F1** key
- On Linux , **Shift+F1** key combination.

The following keyboard shortcuts are available only when you connect to a target:

Commands view

You can use:

Ctrl+Space	Access the content assist for autocompletion of commands.
Enter	Execute the command that is entered in the adjacent field.
DOWN arrow	Navigate down through the command history.
UP arrow	Navigate up through the command history.

Debug Control view

You can use:

F5	Step at source or instruction level including stepping into all function calls where there is debug information.
F6	Step at source or instruction level but stepping over all function calls.
F7	Continue running to the next instruction after the selected stack frame finishes.
F8	Continue running the application after a breakpoint is hit or the target is interrupted.
F9	Interrupt the target and stop the current application if it is running.

2.5.1 See also

Reference

- *Launching the debugger from Eclipse* on page 2-6
- *ARM® DS-5™ Debugger Command Reference*:
  - Chapter 2 *DS-5 Debugger commands*
  - Chapter 3 *CMM-style commands supported by the debugger*.

2.6 DS-5 Debugger command-line console keyboard shortcuts

When using the DS-5 Debugger command line console, there are many useful line editing features provided, including a command history and some common keyboard shortcuts.

Each command you enter is stored in the command history. Use the UP and DOWN arrow keys to navigate through the command history, for example to find and reissue a previous command.

To make editing commands and navigating the command history easier, a number of special keyboard shortcuts are available.

The following is a list of the most common keyboard shortcuts:

Ctrl+A Move the cursor to the start of the line.

Ctrl+D Quit the debugger console.

Ctrl+E Move the cursor to the end of the line.

Ctrl+N Search forward through the command history for the currently entered text.

Ctrl+P Search back through the command history for the currently entered text.

Ctrl+W Delete the last word.

DOWN arrow Navigate down through the command history.

UP arrow Navigate up through the command history.

2.6.1 See also

Reference

- *Launching the debugger from the command-line console* on page 2-7
- *ARM® DS-5™ Debugger Command Reference*:
  - Chapter 2 *DS-5 Debugger commands*
  - Chapter 3 *CMM-style commands supported by the debugger*.

Chapter 3
Configuring and connecting to a target

The following topics describe how to configure and connect to a debug target using ARM® DS-5™ Debugger in the Eclipse *Integrated Development Environment* (IDE).

Tasks

Concepts

3.1 Types of target connections

Before you can debug an application you must set up a connection between the host workstation running the debugger and the target.

There are several types of connections supported by the debugger:

Linux application development

To debug a Linux application you can use a TCP or serial connection:

- to `gdbserver` running on model that is pre-configured to boot ARM Embedded Linux.
- to `gdbserver` running on a hardware target.

This type of development requires `gdbserver` to be installed and running on the target. If `gdbserver` is not installed on the target, either see the documentation for your Linux distribution or check with your provider. Alternatively, you might be able to use the `gdbserver` from the DS-5 installation at *`install_directory`*`/arm`.

Linux kernel development

To debug a Linux kernel module you can use a debug hardware agent connected to the host workstation and the running target.

Bare-metal development

To debug a bare-metal application you can use a debug hardware agent connected to the host workstation and the target.

Snapshot Viewer

The Snapshot Viewer enables you to debug a read-only representation of your application using previously captured state.

Note

Currently DS-5 only supports DS-5 Debugger connections to the Snapshot Viewer using the command-line console.

3.1.1 See also

Tasks

- *Configuring a connection to an RTSM model* on page 3-3
- *Configuring a connection to a Linux target using gdbserver* on page 3-5
- *Configuring a connection to a Linux Kernel* on page 3-7
- *Configuring a connection to a bare-metal target* on page 3-9.

Concepts

- *Debugger concepts* on page 2-4
- *About the Snapshot Viewer* on page 10-5.

3.2 Configuring a connection to an RTSM model

DS-5 supports serial connections and network bridging between an RTSM model and the host machine on both Windows and Linux platforms.

To connect to a *Real-Time System Model* (RTSM):

1. Select **Window → Open Perspective → DS-5 Debug** from the main menu.
2. Select **Debug Configurations...** from the **Run** menu.
3. Select **DS-5 Debugger** from the configuration tree and then click on **New** to create a new configuration. Alternatively you can select an existing DS-5 Debugger configuration and then click on **Duplicate** from the toolbar.
4. In the Name field, enter a suitable name for the new configuration.
5. Click on the **Connection** tab to configure a DS-5 Debugger target connection:
   a. Select the required RTSM platform.
   b. Select the **Linux Application Debug** project type.
   c. Select the required debug operation. For example, if you are using a *Virtual File System* (VFS) then select **Debug target resident application**.
   d. In the Connections panel, a serial connection is automatically configured.
   e. If you are using VFS, select **Enable virtual file system support**. The default VFS mounting point maps the Eclipse workspace root directory to the `/writeable` directory on the model. Leave the default or change as required.

      Note

      VFS is only set-up on initialisation of the model. Changes to the VFS directory structure might require restarting the model.
6. Click on the **Files** tab to define the target environment and select debug versions of the application file and libraries on the host that you want the debugger to use.
   a. In the Target Configuration panel, specify the location of the application on the target. You can also specify the target working directory if required.
   b. In the Files panel, select the files on the host that you want the debugger to use to load the debug information.

   Note

   Options in the **Files** tab are dependent on the type of debug operation that you select.
7. Click on the **Debugger** tab to configure the debugger settings.
   a. Specify the actions that you want the debugger to do after connection to the target.
   b. Configure the host working directory or use the default.
   c. Configure the search paths on the host used by the debugger when it displays source code.
8. If required, click on the **Arguments** tab to enter arguments that are passed to the application when the debug session starts.
9. If required, click on the **Environment** tab to create and configure the target environment variables that are passed to the application when the debug session starts.
10. Click on **Apply** to save the configuration settings.

11. Click on **Debug** if you want to connect to the target and begin debugging immediately.

    Alternatively, click on **Close** to close the Debug Configurations dialog box. Use the Debug Control view to connect to the target associated with this debug configuration.

12. Debugging requires the DS-5 Debug perspective. If the Confirm Perspective Switch dialog box opens, click on **Yes** to switch perspective.

When connected and the DS-5 Debug perspective opens you are presented with all the relevant views and editors.

For more information on these options, use the dynamic help.

3.2.1 See also

Tasks

- *Launching the debugger from Eclipse* on page 2-6
- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Concepts

- *Types of target connections* on page 3-2.

Reference

- *Debug Configurations - Connection tab* on page 11-59
- *Debug Configurations - Files tab* on page 11-63
- *Debug Configurations - Debugger tab* on page 11-67
- *Debug Configurations - Arguments tab* on page 11-71
- *Debug Configurations - Environment tab* on page 11-73.

3.3 Configuring a connection to a Linux target using `gdbserver`

You can connect to an application that is already running on a target using `gdbserver`.

3.3.1 Prerequisites

Before connecting you must:

1. Set up the target with an *Operating System* (OS) installed and booted. See the documentation supplied with the target for more information.
2. Set up the target connection:
   - For a TCP connection, obtain the target IP address or name.
   - For a serial connection, configure the target serial port and baud rate. For example:
     ```
     stty -F /dev/ttyS2 115200 -brkint -icrnl -imaxbel -opost -onlcr -isig
     -icanon -iexten -echo -echoe -echok -echoctl -echoke
     ```

If you are connecting to an already running `gdbserver` you must ensure that you have:

1. `gdbserver` installed and running on the target.

   To run `gdbserver` and the application on the target you can use:

   `gdbserver` *`port path/myApplication`*

   Where:
   - *`port`* is the connection port between `gdbserver` and the application. For example `:5000`.
   - *`path/myApplication`* is the application that you want to debug.
2. An application image loaded and running on the target.

3.3.2 Procedure

To connect to the target:

1. Select **Window → Open Perspective → DS-5 Debug** from the main menu.
2. Select **Debug Configurations...** from the **Run** menu.
3. Select **DS-5 Debugger** from the configuration tree and then click on **New** to create a new configuration. Alternatively you can select an existing DS-5 Debugger configuration and then click on **Duplicate** from the toolbar.
4. In the Name field, enter a suitable name for the new configuration.
5. Click on the **Connection** tab to configure a DS-5 Debugger target connection:
   a. Select the required platform.
   b. Select the **Linux Application Debug** project type.
   c. Select the required debug operation.
   d. Configure the connection between the debugger and `gdbserver`.
6. Click on the **Files** tab to define the target environment and select debug versions of the application file and libraries on the host that you want the debugger to use.
   a. In the Target Configuration panel, select the application on the host that you want to download to the target and specify the location on the target where you want to download the selected file.

b. In the Files panel, select the files on the host that you want the debugger to use to load the debug information. If required, you can also specify other files on the host that you want to download to the target.

Note

Options in the **Files** tab are dependent on the type of debug operation that you select.

7. Click on the **Debugger** tab to configure the debugger settings.
   a. In the Run control panel, specify the actions that you want the debugger to do after connection to the target.
   b. Configure the host working directory or use the default.
   c. In the Paths panel, specify any source or library search directories on the host that the debugger uses when it displays source code.
8. If required, click on the **Arguments** tab to enter arguments that are passed to the application when the debug session starts.
9. If required, click on the **Environment** tab to create and configure the target environment variables that are passed to the application when the debug session starts.
10. Click on **Apply** to save the configuration settings.
11. Click on **Debug** to connect to the target.
12. Debugging requires the DS-5 Debug perspective. If the Confirm Perspective Switch dialog box opens, click **Yes** to switch perspective.

When connected and the DS-5 Debug perspective opens you are presented with all the relevant views and editors.

For more information on these options, use the dynamic help.

3.3.3 See also

Tasks

- *Launching the debugger from Eclipse* on page 2-6
- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Concepts

- *Types of target connections* on page 3-2.

Reference

- *Debug Configurations - Connection tab* on page 11-59
- *Debug Configurations - Files tab* on page 11-63
- *Debug Configurations - Debugger tab* on page 11-67
- *Debug Configurations - Arguments tab* on page 11-71
- *Debug Configurations - Environment tab* on page 11-73.

3.4 Configuring a connection to a Linux Kernel

You can connect to running target using a debug hardware agent.

Note

By default for this type of connection, all processor exceptions are handled by Linux on the target. You can use the Manage Signals dialog box in the Breakpoints view menu to modify the default handler settings.

3.4.1 Prerequisites

Before connecting you must ensure that you have the target IP address or name for the connection between the debugger and the debug hardware agent.

3.4.2 Procedure

To connect to the target:

1. Select **Window → Open Perspective → DS-5 Debug** from the main menu.
2. Select **Debug Configurations...** from the **Run** menu.
3. Select **DS-5 Debugger** from the configuration tree and then click on **New** to create a new configuration. Alternatively you can select an existing DS-5 Debugger configuration and then click on **Duplicate** from the toolbar.
4. In the Name field, enter a suitable name for the new configuration.
5. Click on the **Connection** tab to configure a DS-5 Debugger target connection:
   a. Select the required platform.
   b. Select the **Linux Kernel and/or Device Driver Debug** project type.
   c. Select the required debug operation.
   d. Configure the connection between the debugger and the debug hardware agent.
6. Click on the **Files** tab and select compiled debug versions of the:
   a. Operating System kernel image.
   b. Module image, if required.
7. Click on the **Debugger** tab to configure the debugger settings.
   a. In the Run control panel, select **Connect only** and set up initialization scripts as required.
   b. Configure the host working directory or use the default.
   c. In the Paths panel, specify any source search directories on the host that the debugger uses when it displays source code.
8. Click on **Apply** to save the configuration settings.
9. Click on **Debug** to connect to the target.
10. Debugging requires the DS-5 Debug perspective. If the Confirm Perspective Switch dialog box opens, click **Yes** to switch perspective.

When connected and the DS-5 Debug perspective opens you are presented with all the relevant views and editors.

For more information on these options, use the dynamic help.

3.4.3 See also

Tasks

- *Handling processor exceptions* on page 4-24
- *Launching the debugger from Eclipse* on page 2-6
- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Concepts

- *Types of target connections* on page 3-2
- *About debugging Linux kernel modules* on page 6-7.

Reference

- *Debug Configurations - Connection tab* on page 11-59
- *Debug Configurations - Files tab* on page 11-63
- *Debug Configurations - Debugger tab* on page 11-67.

3.5 Configuring a connection to a bare-metal target

You can download and connect to an application running on a target using a debug hardware agent.

3.5.1 Prerequisites

Before connecting you must ensure that you have the target IP address or name for the connection between the debugger and the debug hardware agent.

3.5.2 Procedure

To connect to the target:

1. Select **Window → Open Perspective → DS-5 Debug** from the main menu.
2. Select **Debug Configurations...** from the **Run** menu.
3. Select **DS-5 Debugger** from the configuration tree and then click on **New** to create a new configuration. Alternatively you can select an existing DS-5 Debugger configuration and then click on **Duplicate** from the toolbar.
4. In the Name field, enter a suitable name for the new configuration.
5. Click on the **Connection** tab to configure a DS-5 Debugger target connection:
   a. Select the required platform.
   b. Select the **Bare Metal Debug** project type.
   c. Select the required debug operation.
   d. Configure the connection between the debugger and the debug hardware agent.
6. Click on the **Files** tab to define the target environment and select debug versions of the application file and libraries on the host that you want the debugger to use.
   a. In the Target Configuration panel, select the application on the host that you want to download to the target.
7. Click on the **Debugger** tab to configure the debugger settings.
   a. In the Run control panel, specify the actions that you want the debugger to do after connection to the target.
   b. Configure the host working directory or use the default.
   c. In the Paths panel, specify any source search directories on the host that the debugger uses when it displays source code.
8. If required, click on the **Arguments** tab to enter arguments that are passed to the application when the debug session starts.
9. If required, click on the **Environment** tab to create and configure the target environment variables that are passed to the application when the debug session starts.
10. Click on **Apply** to save the configuration settings.
11. Click on **Debug** to connect to the target.
12. Debugging requires the DS-5 Debug perspective. If the Confirm Perspective Switch dialog box opens, click **Yes** to switch perspective.

When connected and the DS-5 Debug perspective opens you are presented with all the relevant views and editors.

For more information on these options, use the dynamic help.

3.5.3 See also

Tasks

- *Launching the debugger from Eclipse* on page 2-6
- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Concepts

- *Types of target connections* on page 3-2.

Reference

- *Debug Configurations - Connection tab* on page 11-59
- *Debug Configurations - Files tab* on page 11-63
- *Debug Configurations - Debugger tab* on page 11-67
- *Debug Configurations - Arguments tab* on page 11-71
- *Debug Configurations - Environment tab* on page 11-73.

3.6 Disconnecting from a target

In the Debug Control view you can click on the **Disconnect from Target** toolbar icon. Alternatively, in the Commands view you can enter `quit` in the Command field and then click **Submit**.

3.6.1 See also

Tasks

- *Configuring a connection to an RTSM model* on page 3-3
- *Configuring a connection to a Linux target using gdbserver* on page 3-5
- *Configuring a connection to a Linux Kernel* on page 3-7
- *Configuring a connection to a bare-metal target* on page 3-9.

Reference

- *Debug Control view* on page 11-18
- *Commands view* on page 11-15
- *DS-5 Debugger menu and toolbar icons* on page 11-75.
- *ARM® DS-5™ Debugger Command Reference*:
  - *quit, exit* on page 2-120.

Chapter 4
Controlling execution

The following topics show how to stop the target execution when certain events occur, and when certain conditions are met.

Tasks

Concepts

4.1 About loading an image on to the target

Before you can start debugging your application image, you must load the files on to the target. The files on your target must be the same as those on your local host workstation. The code layout must be identical, but the files on your target do not need to contain debug information.

You can manually load the files on to the target or you can configure a debugger connection to automatically do this after a connection is established. Some target connections do not support load operations and the relevant menu options are therefore disabled.

After connecting to the target you can also use the Debug Control view menu entry **Load...** to load files as required. The following options for loading an image are available:

Load Image Only Loads the application image on to the target.

Load Image and Debug Info

Loads the application image on to the target and debug information from the same image into the debugger.

Hex Offset Specifies the offset that is added to all addresses within the image.

Enable on-demand loading

Specifies how you want the debugger to load debug information. Enabling this option can provide a faster load and use less memory but debugging might be slower.

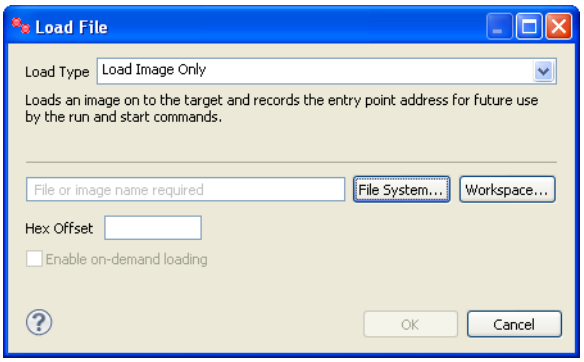

Figure 4-1 Load File dialog box

4.1.1 See also

Tasks

- *Configuring a connection to an RTSM model* on page 3-3
- *Configuring a connection to a Linux target using gdbserver* on page 3-5
- *Configuring a connection to a bare-metal target* on page 3-9
- *Disconnecting from a target* on page 3-11
- *Running an image* on page 4-6.

Concepts

- *About loading debug information into the debugger* on page 4-4.

Reference

- *Debug Control view* on page 11-18
- *Commands view* on page 11-15

- *Debug Configurations - Connection tab* on page 11-59.
- *ARM® DS-5™ Debugger Command Reference*:
  - *load* on page 2-101
  - *loadfile* on page 2-102.

4.2 About loading debug information into the debugger

An executable image contains symbolic references, such as function and variable names, in addition to the application code and data. These symbolic references are generally referred to as debug information. Without this information the debugger is unable to debug at the source level.

To debug an application at source level, the image file and shared object files must be compiled with debug information, and a suitable level of optimization. For example, when compiling with either the ARM® or the GNU compiler you can use the following options:

```
-g -O0
```

Debug information is not loaded when a file is loaded, but is a separate action. A typical load sequence is:

1. Load the main application image.
2. Load any shared objects.
3. Load the symbols for the main application image
4. Load the symbols for shared objects on-demand.

Images and shared objects might be preloaded onto the target, such as an image in a ROM device or an OS-aware target. The corresponding image file and any shared object files must contain debug information, and be accessible from your local host workstation. You then configure a connection to the target loading only the debug information from these files. Use the **Load symbols from file** option on the debug configuration **Files** tab as appropriate for the target environment.

After connecting to the target you can also use the view menu entry **Load...** in the Debug Control view to load files as required. The following options for loading debug information are available:

Add Symbols File Loads additional debug information into the debugger.

Load Debug Info Loads debug information into the debugger.

Load Image and Debug Info

Loads the application image on to the target and debug information from the same images into the debugger.

Hex Offset Specifies the offset that is added to all addresses within the image.

Enable on-demand loading

Specifies how you want the debugger to load debug information. Enabling this option can provide a faster load and use less memory but debugging might be slower.

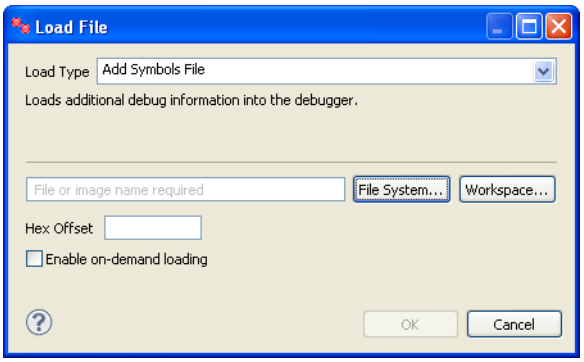

Figure 4-2 Load additional debug information dialog box

The debug information in an image or shared object also contains the path of the sources used to build it. When execution stops at an address in the image or shared object, the debugger attempts to open the corresponding source file. If this path is not present or the required source file is not found, then you must inform the debugger where the source file is located. You do this by setting up a substitution rule to associate the path obtained from the image with the path to the required source file that is accessible from your local host workstation.

4.2.1 See also

Tasks

- *Configuring a connection to an RTSM model* on page 3-3
- *Configuring a connection to a Linux target using gdbserver* on page 3-5
- *Configuring a connection to a bare-metal target* on page 3-9
- *Disconnecting from a target* on page 3-11
- *Running an image* on page 4-6
- *Configuring the debugger path substitution rules* on page 4-26.

Concepts

- *About loading an image on to the target* on page 4-2.

Reference

- *Debug Control view* on page 11-18
- *Commands view* on page 11-15
- *Debug Configurations - Connection tab* on page 11-59
- *ARM® DS-5™ Debugger Command Reference*:
  - *add-symbol-file* on page 2-22
  - *discard-symbol-file* on page 2-46
  - *file, symbol-file* on page 2-55
  - *set substitute-path* on page 2-147.

4.3 Running an image

You must run an application image to be able to monitor how it is executed on a target.

Before you can run an image it must be loaded onto the target. An image can either be preloaded on a target or loaded onto the target as part of the debug session.

Note

The files that resides on the target do not have to contain debug information however, to be able to debug them you must have the corresponding files with debug information on your local host workstation.

Use the Debug Configurations dialog box to set up a connection and define the run control options that you want the debugger to do after connection. You do this by selecting **Debug Configurations...** from the **Run** menu.

After connection, you can control the debug session by using the toolbar icons in the Debug Control view.

4.3.1 See also

Tasks

- Chapter 5 *Examining the target*.

Reference

- *Debug Control view* on page 11-18
- *Commands view* on page 11-15
- *Debug Configurations - Connection tab* on page 11-59
- *Debug Configurations - Files tab* on page 11-63
- *Debug Configurations - Debugger tab* on page 11-67
- *ARM® DS-5™ Debugger Command Reference*:
  - *continue* on page 2-38
  - *run* on page 2-125
  - *start* on page 2-180.

4.4 About breakpoints and watchpoints

Breakpoints and watchpoints enable you to stop the target when certain events occur, and when certain conditions are met. When execution stops you can then choose to examine the contents of memory, registers, or variables, or you might have specified other actions to be taken before execution resumes.

The debugger provides the following types:

Breakpoints A breakpoint enables you to interrupt your application when execution reaches a specific address. A breakpoint is always related to a particular memory address, regardless of what might be stored there. When execution reaches the breakpoint, normal execution stops before any instruction stored there is performed.

You can set:

- software breakpoints that trigger when a particular instruction is executed at a specific address
- hardware breakpoints that trigger when the processor attempts to execute an instruction that is fetched from a specific memory address
- conditional breakpoints that trigger when an expression evaluates to true or when an ignore counter is reached
- temporary software or hardware breakpoints that are subsequently deleted when the breakpoint is hit.

The type of breakpoints you can set depends on the:

- memory region and the related access attributes
- hardware support provided by your target processor
- debug interface used to maintain the target connection
- running state if you are debugging an OS-aware application.

Watchpoints A watchpoint is similar to a breakpoint, but it is the address or value of a data access that is monitored rather than an instruction being executed from a specific address. You specify a register or a memory address to identify a location that is to have its contents tested. Watchpoints are sometimes known as data breakpoints, emphasizing that they are data dependent. Execution of your application stops when the address being monitored is accessed by your application.

You can set:

- watchpoints that trigger when a particular memory location is accessed in a particular way
- conditional watchpoints that trigger when an expression evaluates to true or when an ignore counter is reached.

4.4.1 Considerations when setting breakpoints and watchpoints

Be aware of the following when setting breakpoints and watchpoints:

- The number of hardware breakpoints available depends on the target.
- If an image is compiled with a high optimization level or perhaps contains C++ templates then the effect of setting a breakpoint in the source code depends on where you set the breakpoint. For example, if you set a breakpoint on an inlined function or a C++ template, then a breakpoint is created for each instance of that function or template. Therefore the target can run out of breakpoint resources.

- Enabling a *Memory Management Unit* (MMU) might set a memory region to read-only. If that memory region contains a software breakpoint, then that software breakpoint cannot be removed. Therefore, make sure you clear software breakpoints before enabling the MMU.
- Watchpoints are only supported on global/static data symbols because they are always in scope. Local variables are not available when you step out of a function.
- Some targets do not support watchpoints. Currently you can only use watchpoint commands on a hardware target using a debug hardware agent.
- The address of the instruction that triggers the watchpoint might not be the address shown in the PC register. This is because of pipelining effects in the processor.
- When debugging an application that uses shared objects, breakpoints that are set within a shared object are deleted when the shared object is unloaded.

4.4.2 See also

Tasks

- *Setting an execution breakpoint* on page 4-9
- *Setting a data watchpoint* on page 4-11
- *Setting a conditional breakpoint* on page 4-13
- *Settinga breakpoint on a specific thread* on page 4-16
- *Pending breakpoints and watchpoints* on page 4-18.

Concepts

- *Debugger concepts* on page 2-4.

Reference

- *ARM® DS-5™ Debugger Command Reference*:
  - *advance* on page 2-24
  - *awatch* on page 2-26
  - *break* on page 2-28
  - *break-stop-on-threads* on page 2-32
  - *clear* on page 2-34
  - *condition* on page 2-37
  - *delete breakpoints* on page 2-40
  - *disable breakpoints* on page 2-43
  - *enable breakpoints* on page 2-51
  - *hbreak* on page 2-63
  - *ignore* on page 2-67
  - *info breakpoints, info watchpoints* on page 2-72
  - *resolve* on page 2-123
  - *rwatch* on page 2-126
  - *set breakpoint* on page 2-133
  - *tbreak* on page 2-187
  - *thbreak* on page 2-189
  - *watch* on page 2-197.

4.5 Setting an execution breakpoint

The debugger enables you to set software or hardware breakpoints, depending on your target memory type. Software breakpoints are implemented by the debugger replacing the instruction at the breakpoint address with a special instruction opcode. Because the debugger requires write access to application memory, software breakpoints can only be set in RAM. Hardware breakpoints are implemented by EmbeddedICE® logic that monitors the address and data buses of your processor. For simulated targets, hardware breakpoints are implemented by your simulator software.

To set an execution breakpoint double click in the left-hand margin of the C/C++ editor or the Disassembly view at the position where you want to set the breakpoint. To delete a breakpoint, double-click on the breakpoint marker.

The following figure shows an example of breakpoints, in the C/C++ editor and in the Disassembly view. These are also visible in the Breakpoints view.

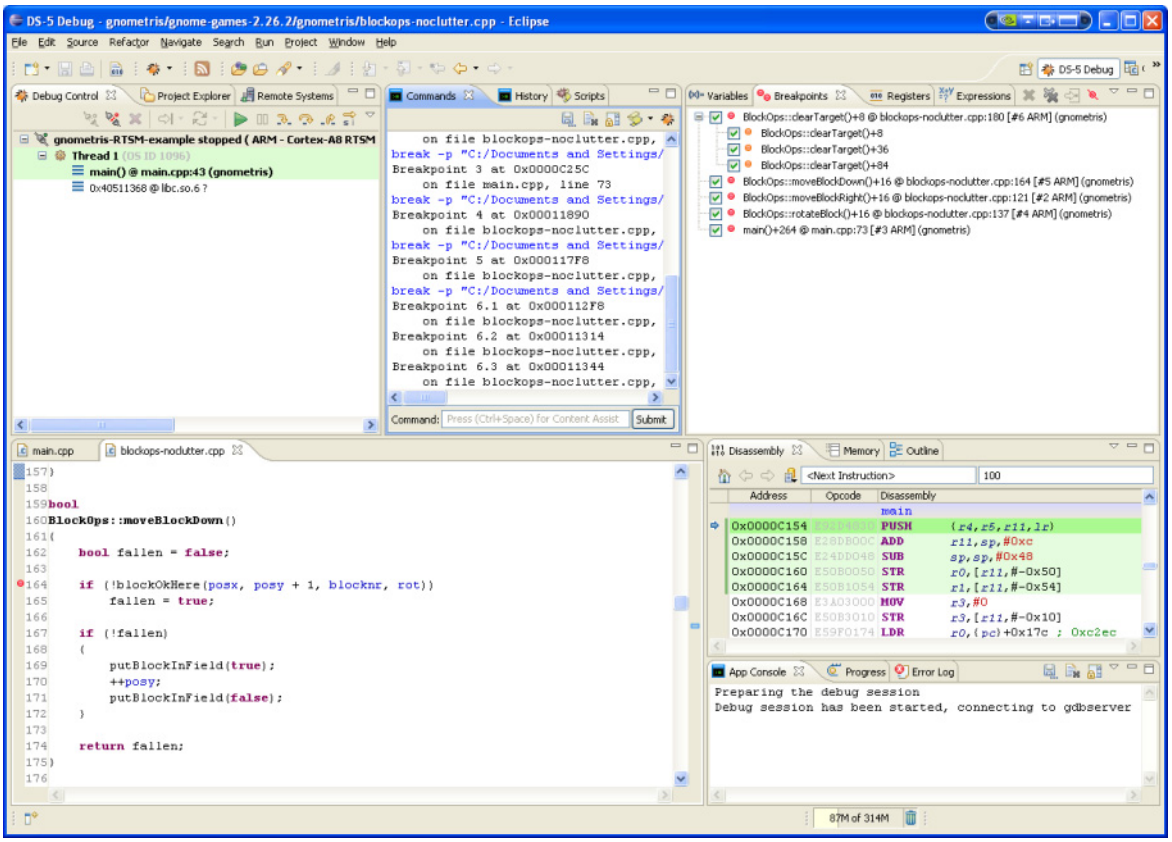

Figure 4-3 Setting an execution breakpoint

4.5.1 See also

Tasks

- *Setting a conditional breakpoint* on page 4-13
- *Settinga breakpoint on a specific thread* on page 4-16
- *Pending breakpoints and watchpoints* on page 4-18.

Concept

- *About breakpoints and watchpoints* on page 4-7.

Reference

- *Breakpoints view* on page 11-8
- *Commands view* on page 11-15
- *Debug Control view* on page 11-18
- *Disassembly view* on page 11-22
- *Registers view* on page 11-36
- *Memory view* on page 11-30
- *Variables view* on page 11-48.
- *ARM® DS-5™ Debugger Command Reference*:
  - *advance* on page 2-24
  - *break* on page 2-28
  - *break-stop-on-threads* on page 2-32
  - *clear* on page 2-34
  - *condition* on page 2-37
  - *delete breakpoints* on page 2-40
  - *disable breakpoints* on page 2-43
  - *enable breakpoints* on page 2-51
  - *hbreak* on page 2-63
  - *ignore* on page 2-67
  - *info breakpoints, info watchpoints* on page 2-72
  - *resolve* on page 2-123
  - *set breakpoint* on page 2-133
  - *tbreak* on page 2-187
  - *thbreak* on page 2-189.

4.6 Setting a data watchpoint

There are times when you want to monitor the values of specific variables or expressions in your source code when running an application. You can do this by setting watchpoints.

In the Variables view, right-click on a data symbol and select **Toggle Watchpoint**. The watchpoint is visible in the Variables view and also in the Breakpoints view.

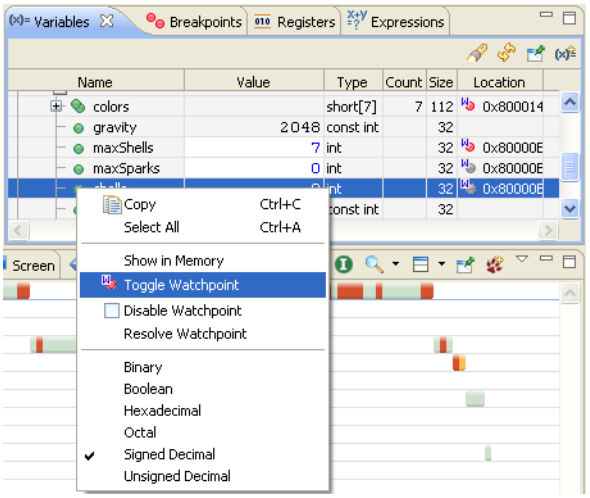

Figure 4-4 Setting a data watchpoint

Note

Watchpoints are only supported on scalar values.

Some targets do not support watchpoints. Currently you can only set a watchpoint on a hardware target using a debug hardware agent.

The address of the instruction that triggers the watchpoint might not be the address shown in the PC register. This is because of pipelining effects.

4.6.1 See also

Tasks

- *Pending breakpoints and watchpoints* on page 4-18.

Concept

- *Debugger concepts* on page 2-4
- *About breakpoints and watchpoints* on page 4-7.

Reference

- *Breakpoints view* on page 11-8
- *Commands view* on page 11-15
- *Debug Control view* on page 11-18
- *Disassembly view* on page 11-22
- *Expressions view* on page 11-25
- *Memory view* on page 11-30
- *Registers view* on page 11-36
- *Variables view* on page 11-48.

- *ARM® DS-5™ Debugger Command Reference*:
  - *awatch* on page 2-26
  - *clearwatch* on page 2-36
  - *delete breakpoints* on page 2-40
  - *disable breakpoints* on page 2-43
  - *enable breakpoints* on page 2-51
  - *info breakpoints, info watchpoints* on page 2-72
  - *rwatch* on page 2-126
  - *set breakpoint* on page 2-133
  - *watch* on page 2-197.

4.7 Setting a conditional breakpoint

Conditional breakpoints have properties assigned to test for conditions that must be satisfied to trigger the breakpoint. For example, you can:

- test a variable for a given value
- execute a function a set number of times
- trigger a breakpoint only on a specific thread.

Conditional breakpoints can be very intrusive and lower the performance if they are hit frequently. This is because the debugger stops the target every time the breakpoint triggers. The specified condition is checked and if it evaluates to true then the target remains in the stopped state, otherwise execution resumes.

Note

You must not assign a script to a breakpoint that has sub-breakpoints. If you do, the debugger attempts to execute the script for each sub-breakpoint. If this happens, an error message is displayed.

You can assign conditions to an existing breakpoint in the Breakpoint Properties dialog box:

1. In the Breakpoints view, right-click on the breakpoint that you want modify to display the context menu.
2. Select **Properties...** to display the Breakpoint Properties dialog box.
3. If you want to set a conditional expression for a specific breakpoint then enter a C-style expression in the **Stop Condition** field. For example, if your application has a variable `x`, then you can specify:

   ```
   x == 10
   ```

4. If you want the debugger to delay hitting the breakpoint until a specific number of passes has occurred, then enter the number of passes in the **Ignore Count** field. For example, if you have a loop that performs 100 iterations, and you want a breakpoint in that loop to be hit after 50 passes, then enter **50**.
5. If you want to run a script when the selected breakpoint is triggered then specify the script file in the **On break, run script** field:
   - enter the location and file name in the field provided
   - click on **File System...** to locate the file in an external directory from the workspace
   - click on **Workspace...** to locate the file in a project directory or sub-directory within the workspace.

   Note

   Take care with the commands used in a script file that is attached to a breakpoint. For example, if the script file contains the `quit` command, the debugger disconnects from the target when the breakpoint is hit.
6. If you want to enable the debugger to automatically continue running the application on completion of all the breakpoint actions then select the **Continue Execution** checkbox.

   Alternatively you can enter the `continue` command as the last command in a script file, that is attached to a breakpoint.
7. If you want to set a breakpoint in one or more threads for a multi-threaded application:
   a. Select **Break on Selected Threads** to enable thread selection.

b. Select the checkbox for each thread that you want to assign the breakpoint to.

8. Click **OK** to save the changes.

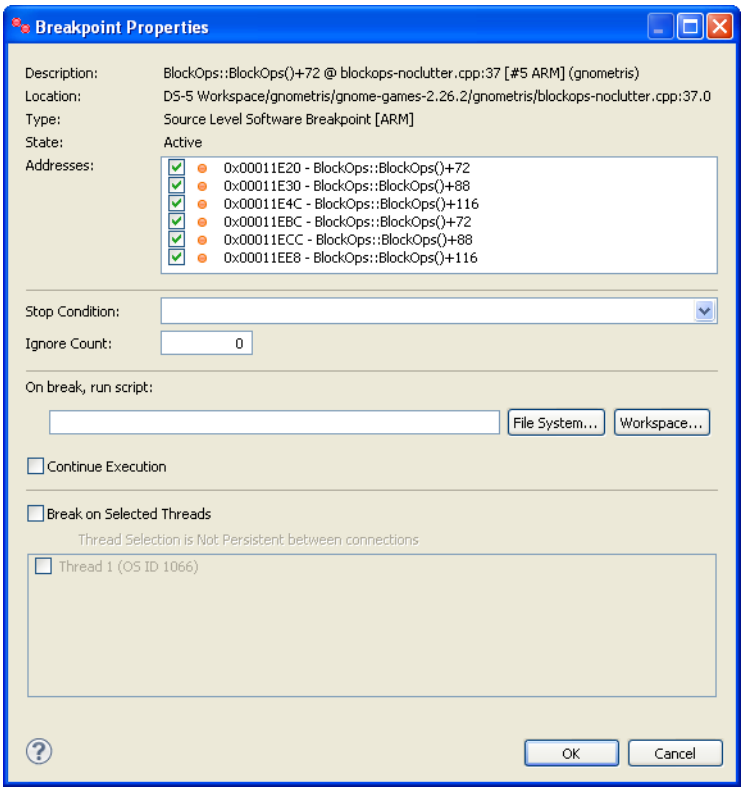

Figure 4-5 Breakpoint Properties dialog box

Breakpoints that are set on a single line of source code with multiple statements are assigned as sub-breakpoints to a parent breakpoint. You can enable, disable, and view the properties of each sub-breakpoint in the same way as a single statement breakpoint. Conditions are assigned to top level breakpoints only and therefore affect both the parent breakpoint and sub-breakpoints.

4.7.1 Considerations when setting multiple conditions on a breakpoint

Be aware of the following when setting multiple conditions on a breakpoint:

- If you set a Stop Condition and an Ignore Count, then the Ignore Count is not decremented until the Stop Condition is met. For example, you might have a breakpoint in a loop that is controlled by the variable `c` and has 10 iterations .If you set the Stop Condition `c==5` and the Ignore Count to `3`, then the breakpoint might never get hit if the loop is reached fewer times than the ignore count value.
- If you choose to break on selected threads, then the Stop Condition and Ignore Count are checked only for the selected threads.

4.7.2 See also

Tasks

- *Setting an execution breakpoint* on page 4-9
- *Settinga breakpoint on a specific thread* on page 4-16
- *Pending breakpoints and watchpoints* on page 4-18.

Concept

- *About breakpoints and watchpoints* on page 4-7.

Reference

- *Breakpoints view* on page 11-8
- *Breakpoint properties dialog box* on page 11-52
- *Commands view* on page 11-15
- *Debug Control view* on page 11-18
- *Disassembly view* on page 11-22
- *Memory view* on page 11-30
- *Registers view* on page 11-36
- *Variables view* on page 11-48.
- *ARM® DS-5™ Debugger Command Reference*:
  - *advance* on page 2-24
  - *break* on page 2-28
  - *break-stop-on-threads* on page 2-32
  - *clear* on page 2-34
  - *condition* on page 2-37
  - *delete breakpoints* on page 2-40
  - *disable breakpoints* on page 2-43
  - *enable breakpoints* on page 2-51
  - *hbreak* on page 2-63
  - *ignore* on page 2-67
  - *info breakpoints, info watchpoints* on page 2-72
  - *resolve* on page 2-123
  - *set breakpoint* on page 2-133
  - *tbreak* on page 2-187
  - *thbreak* on page 2-189.

4.8 Settinga breakpoint on a specific thread

Breakpoints apply to all threads by default, but you can modify the properties for a breakpoint to restrict it to a specific thread:

1. In the Breakpoints view, right-click on the breakpoint that you want modify to display the context menu.
2. Select **Properties...** to display the Breakpoint Properties dialog box.
3. Assign breakpoint conditions as required.
4. Select the **Break on Selected Threads** checkbox to enable thread selection.
5. Select the checkbox for each thread that you want to assign the breakpoint to.
6. Click **OK** to save the changes.

Note

If you set a breakpoint for a specific thread, then any conditions you set for the breakpoint are checked only for that thread.

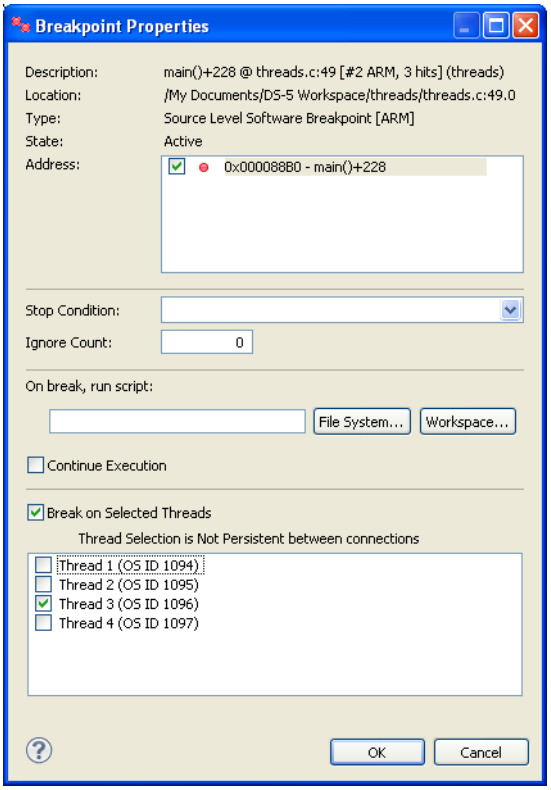

Figure 4-6 Setting a breakpoint on a specific thread

4.8.1 See also

Tasks

- *Setting an execution breakpoint* on page 4-9
- *Setting a conditional breakpoint* on page 4-13
- *Pending breakpoints and watchpoints* on page 4-18.

Reference

- *Breakpoints view* on page 11-8
- *Commands view* on page 11-15
- *Debug Control view* on page 11-18
- *Disassembly view* on page 11-22
- *Memory view* on page 11-30
- *Registers view* on page 11-36
- *Variables view* on page 11-48.
- *ARM® DS-5™ Debugger Command Reference*:
  - *break* on page 2-28
  - *break-stop-on-threads* on page 2-32
  - *info threads* on page 2-94
  - *thread* on page 2-191.

4.9 Pending breakpoints and watchpoints

Breakpoints and watchpoints can be set when debug information is available. Pending breakpoints and watchpoints however, enable you to set breakpoints and watchpoints before the associated debug information is available.

The debugger automatically re-evaluates all pending breakpoints and watchpoints when debug information changes. Those with addresses that can be resolved are set and the others remain pending.

In the Breakpoints view you can force the resolution of a pending breakpoint or watchpoint. For example, this might be useful if you have manually modified the shared library search paths. To do this:

1. Right-click on the pending breakpoint or watchpoint that you want to resolve.
2. Click on **Resolve** to attempt to find the address and set the breakpoint or watchpoint.

To manually set a pendable breakpoint or watchpoint you can use the `-p` option with any of these commands, `advance`, `awatch`, `break`, `hbreak`, `rwatch`, `tbreak`, `thbreak`, `watch`. You can enter debugger commands in the Commands view.

For example:

```
break -p lib.c:20         # Sets a pending breakpoint at line 20 in lib.c
awatch -p *0x80D4         # Sets a pending read/write watchpoint on address 0x80D4
```

4.9.1 See also

Tasks

- *Setting an execution breakpoint* on page 4-9
- *Setting a data watchpoint* on page 4-11
- *Setting a conditional breakpoint* on page 4-13
- *Settinga breakpoint on a specific thread* on page 4-16.

Concept

- *About breakpoints and watchpoints* on page 4-7.

Reference

- *Breakpoints view* on page 11-8
- *Commands view* on page 11-15
- *Debug Control view* on page 11-18
- *Disassembly view* on page 11-22
- *Memory view* on page 11-30
- *Registers view* on page 11-36
- *Variables view* on page 11-48.
- *ARM® DS-5™ Debugger Command Reference*:
  - *advance* on page 2-24
  - *awatch* on page 2-26
  - *break* on page 2-28
  - *break-stop-on-threads* on page 2-32
  - *clear* on page 2-34
  - *condition* on page 2-37
  - *delete breakpoints* on page 2-40

- *disable breakpoints* on page 2-43
- *enable breakpoints* on page 2-51
- *hbreak* on page 2-63
- *ignore* on page 2-67
- *info breakpoints, info watchpoints* on page 2-72
- *resolve* on page 2-123
- *rwatch* on page 2-126
- *set breakpoint* on page 2-133
- *tbreak* on page 2-187
- *thbreak* on page 2-189
- *watch* on page 2-197.

4.10 Stepping through an application

The debugger enables you to finely control the execution of an image by sequentially stepping through an application at the source level or the instruction level.

Note

You must compile your code with debug information to use the source level stepping commands. By default, source level calls to functions with no debug information are stepped over. Use the `set step-mode` command to change the default setting.

There are several ways to step through an application. You can choose to step:

- into or over all function calls
- at source level or instruction level
- through multiple statements in a single line of source code, for example a *for* loop.

Be aware that when stepping at the source level, the debugger uses temporary breakpoints to stop execution at the specified location. These temporary breakpoints might require the use of hardware breakpoints, especially when stepping through code in ROM or Flash. If there are not enough hardware breakpoint resources available, then the debugger displays an error message.

You can use the stepping toolbar in the Debug Control view to step through the application either by source line or instruction.

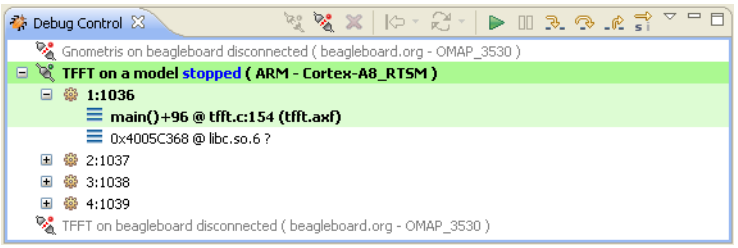

Figure 4-7 Debug Control view

To step a specified number of times you must use the Commands view to manually execute one of the stepping commands with a number. For example:

```
steps 5                         # Execute five source statements
stepi 5                         # Execute five instructions
```

4.10.1 See also

Tasks

- *Examining the target execution environment* on page 5-2
- *Examining the call stack* on page 5-4
- *About debugging multi-threaded applications* on page 6-2
- *About debugging shared libraries* on page 6-4
- *Handling Unix signals* on page 4-22
- *Handling processor exceptions* on page 4-24.

Concepts

- *About breakpoints and watchpoints* on page 4-7.

Reference

- *Breakpoints view* on page 11-8

- *Commands view* on page 11-15
- *Debug Control view* on page 11-18
- *Disassembly view* on page 11-22
- *Memory view* on page 11-30
- *Registers view* on page 11-36
- *Variables view* on page 11-48.
- *ARM® DS-5™ Debugger Command Reference*:
  - *next* on page 2-112
  - *nexti* on page 2-113
  - *nexts* on page 2-114
  - *finish* on page 2-56
  - *set step-mode* on page 2-145
  - *show step-mode* on page 2-172
  - *step* on page 2-182
  - *stepi* on page 2-183
  - *steps* on page 2-184.

4.11 Handling Unix signals

For Linux applications, ARM processors have the facility to trap Unix signals. These are managed in the debugger with the **handle** command or you can use the Manage Signals dialog box in the Breakpoints view menu to modify the default handler settings.

The default handler settings are dependant on the type of debug activity. For example, by default on a Linux kernel connection, all signals are handled by Linux on the target.

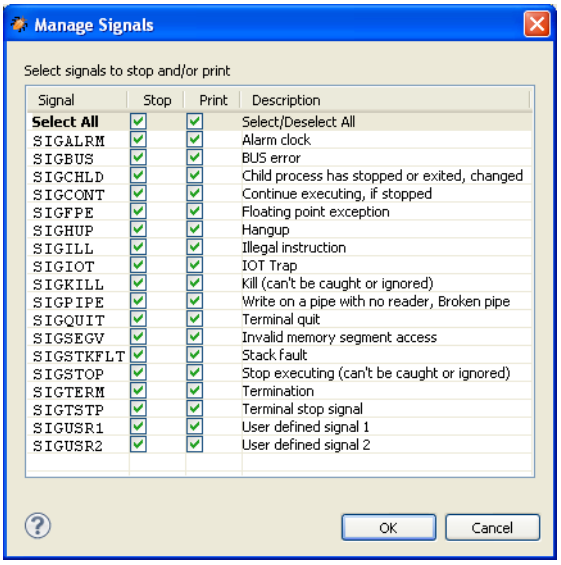

Figure 4-8 Managing signal handler settings

Note

Unix signals `SIGINT` and `SIGTRAP` cannot be debugged in the same way as other signals because they are used internally by the debugger for asynchronous stopping of the process and breakpoints respectively.

4.11.1 Example

If you want the application to ignore a signal but log the event when it is triggered then you must enable stopping on a signal. In the following example, a `SIGHUP` signal occurs causing the debugger to stop and print a message. No signal handler is invoked when using this setting and the application being debugged ignores the signal and continues.

Example 4-1 Ignoring a `SIGHUP` signal

```
handle SIGHUP stop print                # Enable stop and print on SIGHUP signal
```

The following example shows how to debug a signal handler. To do this you must disable stopping on a signal and then set a breakpoint in the signal handler. This is because if stopping on a signal is disabled then the handling of that signal is performed by the process that passes signal to the registered handler. If no handler is registered then the default handler runs and the application generally exits.

Example 4-2 Debugging a `SIGHUP` signal

```
handle SIGHUP nostop noprint            # Disable stop and print on SIGHUP signal
```

4.11.2 See also

Tasks

- *Stepping through an application* on page 4-20
- *Examining the target execution environment* on page 5-2
- *Examining the call stack* on page 5-4
- *About debugging multi-threaded applications* on page 6-2
- *About debugging shared libraries* on page 6-4
- *Handling processor exceptions* on page 4-24.

Concepts

- *About breakpoints and watchpoints* on page 4-7.

Reference

- *Commands view* on page 11-15
- *Breakpoints view* on page 11-8
- *Manage Signals dialog box* on page 11-57.
- *ARM® DS-5™ Debugger Command Reference*:
  - *handle* on page 2-62
  - *info signals, info handle* on page 2-89.

4.12 Handling processor exceptions

For bare-metal, ARM processors have the facility to trap processor exceptions. When enabled, the effect is similar to placing a breakpoint on the selected vector table entry. This is called vector catch. Vector catching is managed in the debugger with the **handle** command or you can use the Manage Signals dialog box in the Breakpoints view menu to modify the default handler settings.

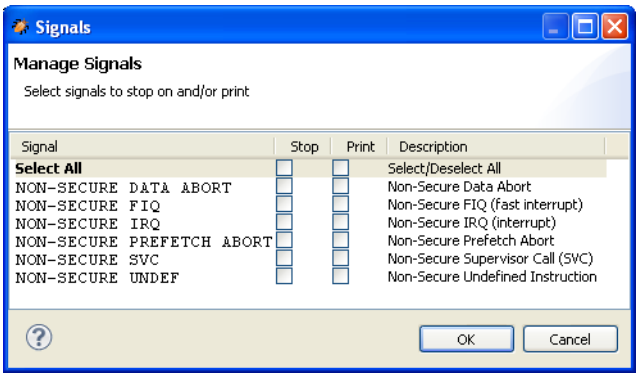

Figure 4-9 Manage exception handler settings

4.12.1 Example

If you want the debugger to catch the exception, log the event, and stop the application when the exception occurs then you must enable stopping on an exception. In the following example, an FIQ exception occurs causing the debugger to stop and print a message. You can then step or run to the handler, if present.

Example 4-3 Debugging an exception handler

```
handle FIQ stop              # Enable stop and print on an FIQ exception
```

If you want the exception to invoke the handler without stopping then you must disable stopping on an exception.

Example 4-4 Ignoring an exception

```
handle FIQ nostop            # Disable stopping on an FIQ exception
```

4.12.2 See also

Tasks

- *Stepping through an application* on page 4-20
- *Examining the target execution environment* on page 5-2
- *Examining the call stack* on page 5-4
- *About debugging multi-threaded applications* on page 6-2.
- *About debugging shared libraries* on page 6-4
- *Handling Unix signals* on page 4-22.

Concepts

- *About breakpoints and watchpoints* on page 4-7.

Reference

- *Commands view* on page 11-15
- *Breakpoints view* on page 11-8
- *Manage Signals dialog box* on page 11-57.
- *ARM® DS-5™ Debugger Command Reference*:
  - *handle* on page 2-62
  - *info signals, info handle* on page 2-89.

4.13 Configuring the debugger path substitution rules

The debugger might not be able to locate the source file when debug information is loaded because:

- The path specified in the debug information is not present on your workstation, or that path does not contain the required source file.
- The source file is not in the same location on your workstation as the image containing the debug information. The debugger attempts to use the same path as this image by default.

Therefore, you must modify the search paths used by the debugger when it executes any of the commands that look up and display source code.

To modify the search paths:

1. Open the Path Substitution dialog box:
   - If a source file cannot be located, the following prompt is displayed in the C/C++ editor. Click on **Set Path Substitution**.

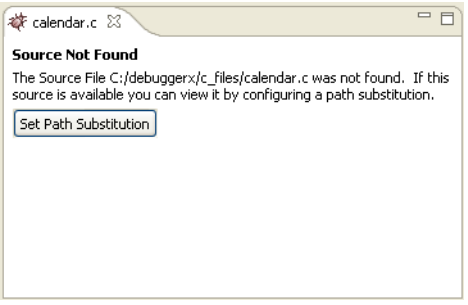

Figure 4-10 Source Not Found warning

   - In the Debug Control view, select **Path Substitution** from the view menu.

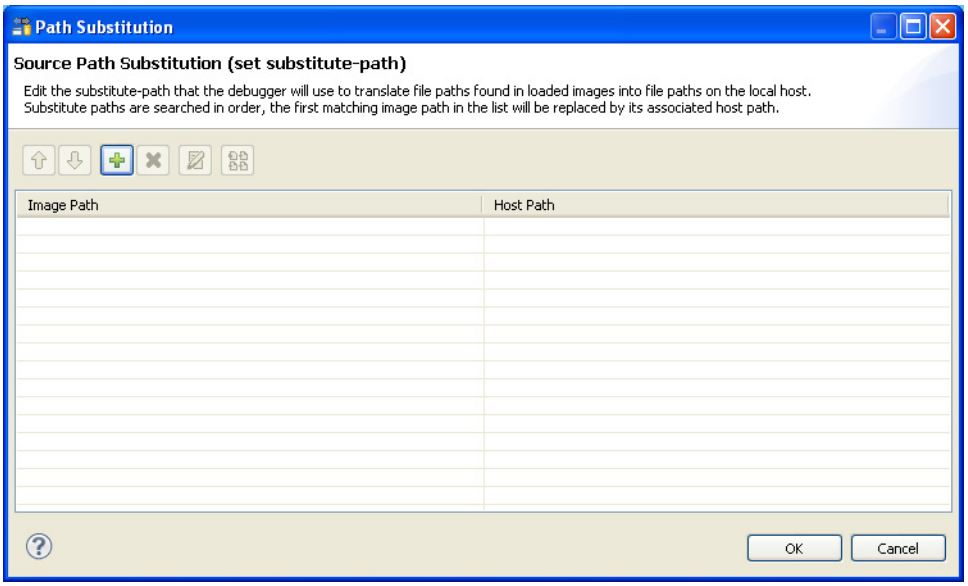

Figure 4-11 Path Substitution dialog box

2. Click on the toolbar icons in the Path Substitution dialog box to add, edit, or duplicate substitution rules:
   a. Enter the original path for the source files in the Image Path field or click on **Select...** to select from the compilation paths.
   b. Enter the current location of the source files in the Host Path field or click on:
      - **File System...** to locate the source files in an external folder
      - **Workspace...** to locate the source files in a workspace project.
   c. Click **OK**.

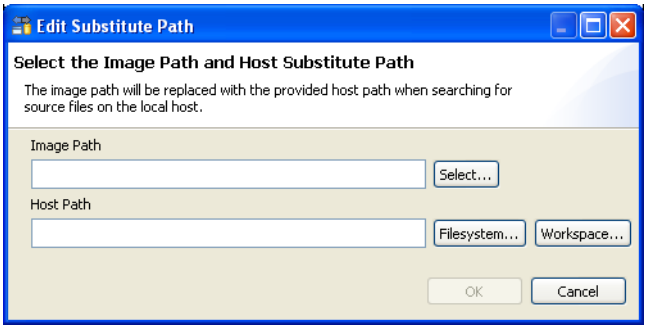

Figure 4-12 Edit Substitute Path dialog box

3. If required, you can use the toolbar icons in the Path Substitution dialog box to change the order of the substitution rules or delete rules that are no longer required.
4. Click **OK** to pass the substitution rules to the debugger and close the dialog box.

4.13.1 See also

Tasks

- *About loading debug information into the debugger* on page 4-4
- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *C/C++ editor* on page 11-12
- *Debug Control view* on page 11-18
- *ARM® DS-5™ Debugger Command Reference*:
  - *set substitute-path* on page 2-147.

Chapter 5
Examining the target

The following topics show how to examining registers, variables, memory, and the call stack.

Tasks

5.1 Examining the target execution environment

During a debug session you might want to display for example, the value of a register or variable, the address of a symbol, the data type of a variable, or the content of memory.

The DS-5 Debug perspective provides the essential debugger views showing the curent values. All the views are associated with the active connection and are updated as you step through the application. You can move any of the views to a different position in the perspective by clicking on the tab and dragging to a new position. You can also double-click on a tab to maximize or reset a view for closer analysis of the view content.

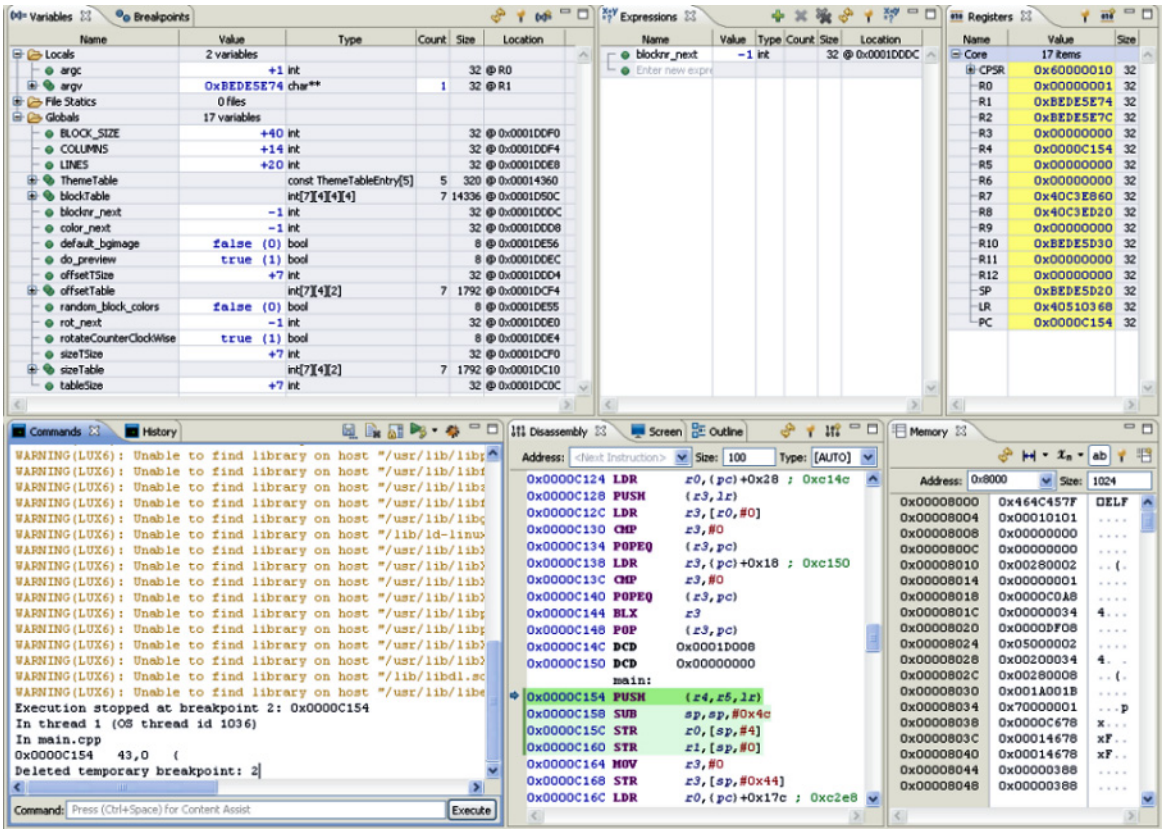

Figure 5-1 Examine the target execution

Alternatively you can use debugger commands to display the required information. In the Commands view you can execute individual commands or you can execute a sequence of commands by using a script file.

5.1.1 See also

Tasks

- *Setting an execution breakpoint* on page 4-9
- *Setting a data watchpoint* on page 4-11
- *Setting a conditional breakpoint* on page 4-13
- *Pending breakpoints and watchpoints* on page 4-18
- *Stepping through an application* on page 4-20
- *Examining the call stack* on page 5-4
- *About debugging multi-threaded applications* on page 6-2
- *About debugging shared libraries* on page 6-4

- *Handling Unix signals* on page 4-22
- *Handling processor exceptions* on page 4-24.

Reference

- *App Console view* on page 11-3
- *ARM Asm Info view* on page 11-5
- *ARM assembler editor* on page 11-6
- *Breakpoints view* on page 11-8
- *C/C++ editor* on page 11-12
- *Commands view* on page 11-15
- *Debug Control view* on page 11-18
- *Disassembly view* on page 11-22
- *Expressions view* on page 11-25
- *History view* on page 11-28
- *Memory view* on page 11-30
- *Registers view* on page 11-36
- *Screen view* on page 11-39
- *Scripts view* on page 11-41
- *Target view* on page 11-43
- *Trace view* on page 11-45
- *Variables view* on page 11-48.
- *ARM® DS-5™ Debugger Command Reference*:
  - *disassemble* on page 2-45
  - *info address* on page 2-70
  - *info functions* on page 2-77
  - *info locals* on page 2-79
  - *info registers* on page 2-86
  - *info symbol* on page 2-92
  - *info variables* on page 2-95
  - *print, inspect* on page 2-118
  - *whatis* on page 2-198
  - *x* on page 2-201.

5.2 Examining the call stack

The call stack, or runtime stack, is an area of memory used to store function return information and local variables. As each function is called, a record is created on the call stack. This record is commonly known as a stack frame.

The debugger can display the calling sequence of any functions that are still in the execution path because their calling addresses are still on the call stack. However:

- When a function completes execution the associated stack frame is removed from the call stack and the information is no longer available to the debugger.
- If the call stack contains a function for which there is no debug information, the debugger might not be able to trace back up the calling stack frames. Therefore you must compile all your code with debug information to successfully view the full call stack.

If you are debugging multi threaded applications, a separate call stack is maintained for each thread.

All the views in the DS-5 Debug perspective are associated with the current stack frame and are updated when you select another frame. The current stack frame is shown in bold text.

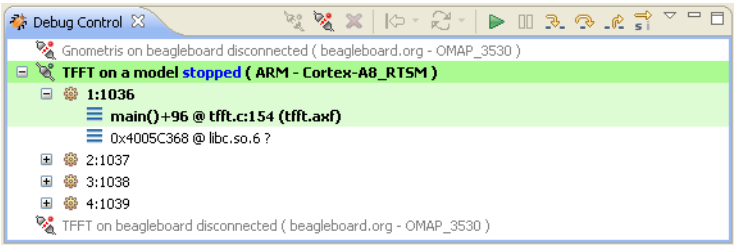

Figure 5-2 Debug Control view

5.2.1 See also

Tasks

- *Setting an execution breakpoint* on page 4-9
- *Setting a data watchpoint* on page 4-11
- *Setting a conditional breakpoint* on page 4-13
- *Pending breakpoints and watchpoints* on page 4-18
- *Stepping through an application* on page 4-20
- *Examining the target execution environment* on page 5-2
- *About debugging multi-threaded applications* on page 6-2
- *About debugging shared libraries* on page 6-4
- *Handling Unix signals* on page 4-22
- *Handling processor exceptions* on page 4-24.

Reference

- *App Console view* on page 11-3
- *ARM Asm Info view* on page 11-5
- *ARM assembler editor* on page 11-6
- *Breakpoints view* on page 11-8
- *C/C++ editor* on page 11-12
- *Commands view* on page 11-15

- *Debug Control view* on page 11-18
- *Disassembly view* on page 11-22
- *Expressions view* on page 11-25
- *History view* on page 11-28
- *Memory view* on page 11-30
- *Registers view* on page 11-36
- *Screen view* on page 11-39
- *Scripts view* on page 11-41
- *Target view* on page 11-43
- *Trace view* on page 11-45
- *Variables view* on page 11-48.
- *ARM® DS-5™ Debugger Command Reference*:
  - *down* on page 2-47
  - *down-silently* on page 2-48
  - *frame* on page 2-61
  - *info frame* on page 2-76
  - *info registers* on page 2-86
  - *info stack, backtrace, where* on page 2-91
  - *select-frame* on page 2-127
  - *set backtrace* on page 2-132
  - *up* on page 2-194
  - *up-silently* on page 2-195.

5.3 About trace support in DS-5

ARM® DS-5™ enables you to perform tracing on your application or system. You can capture in real-time a historical, non-intrusive trace of instructions and data accesses. Tracing is a powerful tool that enables you to investigate problems while the system runs at full speed. These problems can be intermittent, and are difficult to identify through traditional debugging methods that require starting and stopping the processor. Tracing is also useful when trying to identify potential bottlenecks or to improve performance-critical areas of your application.

Before the debugger can trace function executions in your application you must ensure that:

- you have a debug hardware agent, for example, an ARM DSTREAM™ unit with a connection to a trace stream
- the debugger is connected to the debug hardware agent.

5.3.1 See also

Reference

- *Trace view* on page 11-45
- *Export trace report dialog box* on page 11-51
- *Connecting the DSTREAM unit*, http://infocenter.arm.com/help/topic/com.arm.doc.dui0481-/I1004916.html.

Chapter 6
Debugging embedded systems

The following topics give an introduction to debugging embedded systems.

Tasks

6.1 About debugging multi-threaded applications

The debugger tracks the current thread using the debugger variable, `$thread`. You can use this variable in print commands or in expressions. Threads are displayed in the Debug Control view with a unique ID that is used by the debugger and a unique ID from the *Operating System* (OS). For example:

```
Thread 1 (OS ID 1036)
```

where **`Thread 1`** is the ID used by the debugger and `OS ID 1036` is the ID from the OS.

A separate call stack is maintained for each thread and the selected stack frame is shown in bold text. All the views in the DS-5 Debug perspective are associated with the selected stack frame and are updated when you select another frame.

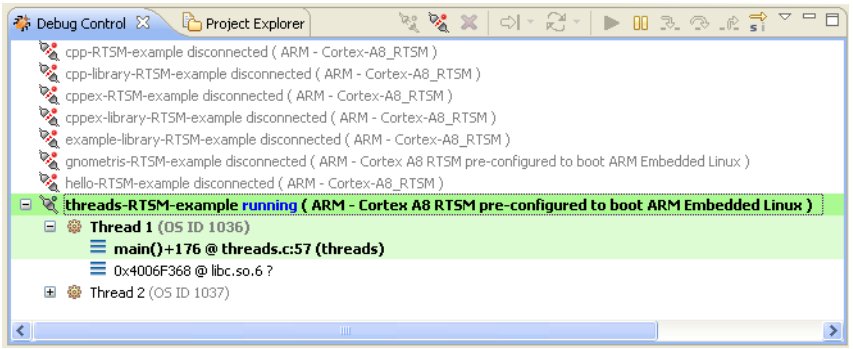

Figure 6-1 Threading call stacks in the Debug Control view

6.1.1 See also

Tasks

- *Settinga breakpoint on a specific thread* on page 4-16
- *Setting an execution breakpoint* on page 4-9
- *Setting a data watchpoint* on page 4-11
- *Setting a conditional breakpoint* on page 4-13
- *Pending breakpoints and watchpoints* on page 4-18
- *Stepping through an application* on page 4-20
- *Examining the target execution environment* on page 5-2
- *Examining the call stack* on page 5-4
- *About debugging shared libraries* on page 6-4
- *Handling Unix signals* on page 4-22
- *Handling processor exceptions* on page 4-24.

Reference

- *Breakpoints view* on page 11-8
- *Commands view* on page 11-15
- *Debug Control view* on page 11-18
- *Disassembly view* on page 11-22
- *Memory view* on page 11-30
- *Modules view* on page 11-33
- *Registers view* on page 11-36
- *Variables view* on page 11-48.

- *ARM® DS-5™ Debugger Command Reference*:
  - *break* on page 2-28
  - *break-stop-on-threads* on page 2-32
  - *info threads* on page 2-94
  - *thread* on page 2-191.

6.2 About debugging shared libraries

Shared libraries enable parts of your application to be dynamically loaded at runtime. You must ensure that the shared libraries on your target are the same as those on your host. The code layout must be identical, but the shared libraries on your target do not need to contain debug information.

You can set standard execution breakpoints in a shared library but not until it is loaded by the application and the debug information is loaded into the debugger. Pending breakpoints however, enable you to set execution breakpoints in a shared library before it is loaded by the application.

When a new shared library is loaded the debugger re-evaluates all pending breakpoints, those with addresses that it can resolve, are set as standard execution breakpoints. Unresolved addresses remain as pending breakpoints.

The debugger automatically changes any breakpoints in a shared library to a pending breakpoint when the library is unloaded by your application.

You can load shared libraries in the Debug Configurations dialog box. If you have one library file then you can use the **Load symbols from file** option on the **Files** tab.

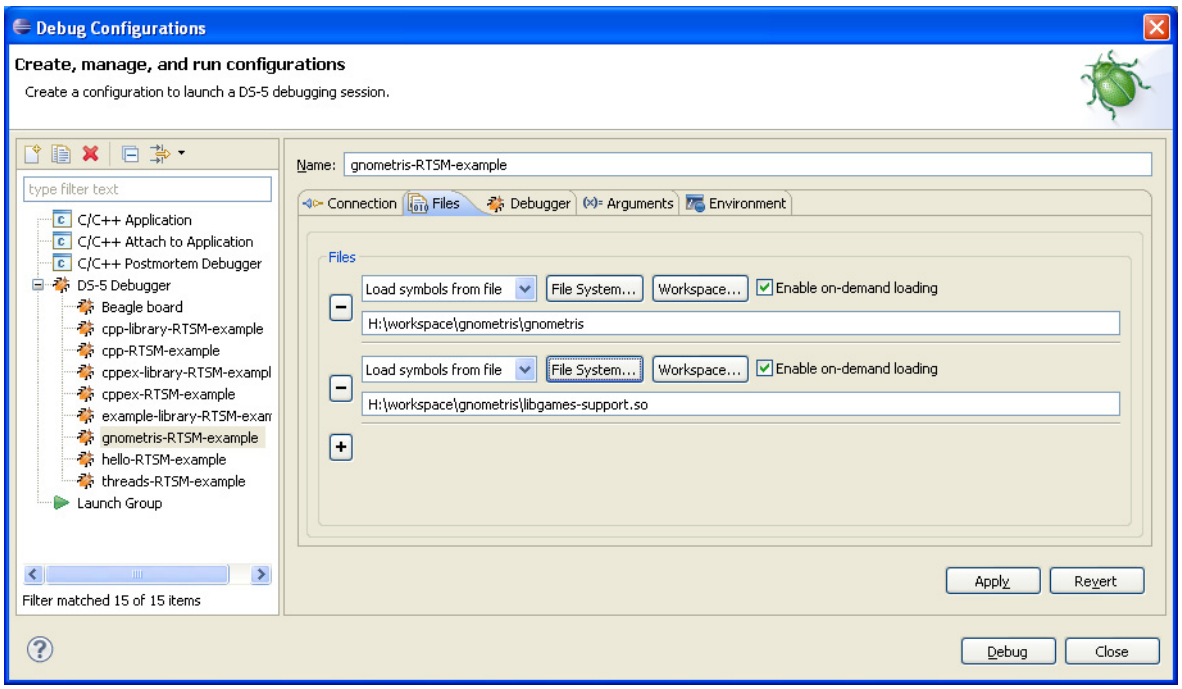

Figure 6-2 Adding individual shared library files

Alternatively if you have multiple library files then it is probably more efficient to modify the search paths in use by the debugger when searching for shared libraries. To do this you can use the **Shared library search directory** option in the Paths panel of the **Debugger** tab.

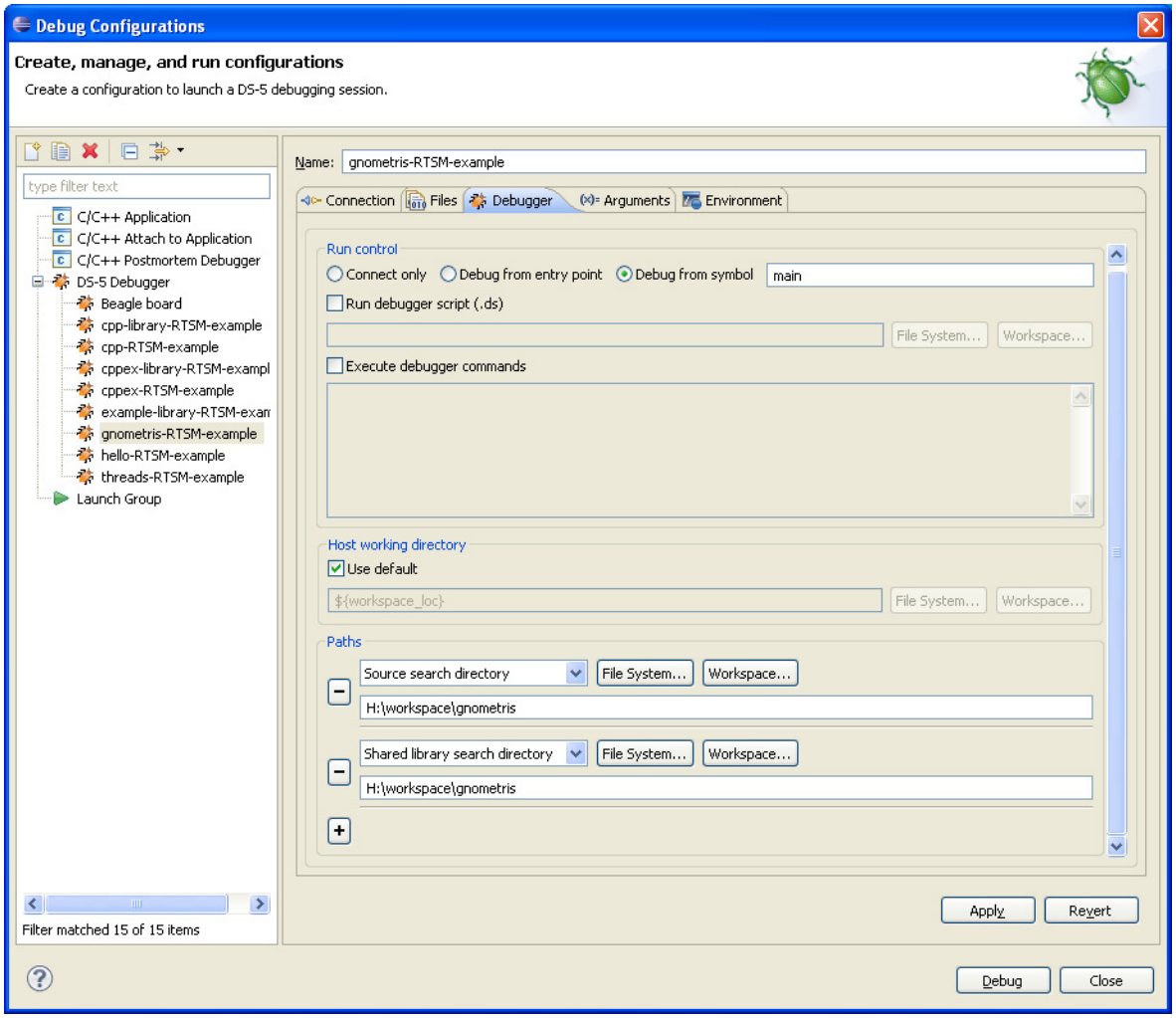

Figure 6-3 Modifying the shared library search paths

For more information on the options in the Debug Configurations dialog box, use the dynamic help.

6.2.1 Modules view

Shared libraries are displayed in the Modules view. During a debug session you can use the context menu in the Modules view to load and discard debug information for shared libraries.

The Modules view is not visible by default. To add this view:

1. Ensure that you are in the DS-5 Debug perspective.
2. Select **Window → Show View** to open the Show View dialog box.
3. Select **Modules** view.

6.2.2 See also

Tasks

- *Configuring a connection to a Linux target using gdbserver* on page 3-5
- *Setting an execution breakpoint* on page 4-9
- *Setting a data watchpoint* on page 4-11

- *Setting a conditional breakpoint* on page 4-13
- *Pending breakpoints and watchpoints* on page 4-18
- *Stepping through an application* on page 4-20
- *Examining the target execution environment* on page 5-2
- *Examining the call stack* on page 5-4
- *About debugging multi-threaded applications* on page 6-2
- *Handling Unix signals* on page 4-22
- *ARM® DS-5™ Getting Started with DS-5*:
  - *Debugging Gnometris* on page 3-16.

Concepts

- *Launching the debugger from Eclipse* on page 2-6
- *About breakpoints and watchpoints* on page 4-7.

Reference

- *Breakpoints view* on page 11-8
- *Commands view* on page 11-15
- *Debug Control view* on page 11-18
- *Disassembly view* on page 11-22
- *Memory view* on page 11-30
- *Modules view* on page 11-33
- *Registers view* on page 11-36
- *Variables view* on page 11-48
- *Debug Configurations - Files tab* on page 11-63
- *Debug Configurations - Debugger tab* on page 11-67.
- *ARM® DS-5™ Debugger Command Reference*:
  - *add-symbol-file* on page 2-22
  - *info sharedlibrary* on page 2-88
  - *nosharedlibrary* on page 2-115
  - *set auto-solib-add* on page 2-131
  - *set solib-search-path* on page 2-144
  - *set stop-on-solib-events* on page 2-146
  - *set sysroot, set solib-absolute-prefix* on page 2-148
  - *sharedlibrary* on page 2-151.

6.3 About debugging Linux kernel modules

Linux kernel modules provide a way to extend the functionality of the kernel, and are typically used for things such as device and file system drivers. Modules can either be built into the kernel or can be compiled as a loadable module and then dynamically inserted and removed from a running kernel during development without the need to frequently recompile the kernel. However, some modules must be built into the kernel and are not suitable for loading dynamically. An example of a built-in module is one that is required during kernel boot and must be available prior to the root file system being mounted.

You can set source-level breakpoints in a module provided that the debug information is loaded into the debugger. Attempts to set a breakpoint in a module before it is inserted into the kernel results in the breakpoint being pended.

When debugging a module, you must ensure that the module on your target is the same as that on your host. The code layout must be identical, but the module on your target does not need to contain debug information.

6.3.1 Built-in module

To debug a module that has been built into the kernel, the procedure is the same as for debugging the kernel itself:

1. Compile the kernel together with the module.
2. Load the kernel image on to the target.
3. Load the related kernel image with debug information into the debugger
4. Debug the module as you would for any other kernel code.

Built-in modules are indistinguishable from the rest of the kernel code, so are not listed by the `info os-modules` command and do not appear in the Modules view.

6.3.2 Loadable module

The procedure for debugging a loadable kernel module is more complex. From a Linux terminal shell, you can use the `insmod` and `rmmod` commands to insert and remove a module. Debug information for both the kernel and the loadable module must be loaded into the debugger. When you insert and remove a module the debugger automatically resolves memory locations for debug information and existing breakpoints. To do this, the debugger intercepts calls within the kernel to insert and remove modules. This introduces a small delay for each action whilst the debugger stops the kernel to interrogate various data structures. For more information on debugging a loadable kernel module, see the tutorial in *Getting Started with DS-5*.

Note

A connection must be established and operating system support must be activated within the debugger before a loadable module can be detected. You can use the `set os` command to control operating system support in the debugger.

6.3.3 Modules view

When the debugger detects a loadable module it displays it in the Modules view.

The Modules view is not visible by default. To add this view:

1. Ensure that you are in the DS-5 Debug perspective.
2. Select **Window → Show View** to open the Show View dialog box.
3. Select **Modules** view.

6.3.4 See also

Tasks

- *Configuring a connection to a Linux Kernel* on page 3-7
- *ARM® DS-5™ Getting Started with DS-5*:
  - *Debugging a loadable kernel module* on page 3-17.

Reference

- *Breakpoints view* on page 11-8
- *Commands view* on page 11-15
- *Debug Control view* on page 11-18
- *Disassembly view* on page 11-22
- *Memory view* on page 11-30
- *Modules view* on page 11-33
- *Registers view* on page 11-36
- *Variables view* on page 11-48
- *Debug Configurations - Files tab* on page 11-63
- *Debug Configurations - Debugger tab* on page 11-67
- *ARM® DS-5™ Debugger Command Reference*:
  - *info os-log* on page 2-82
  - *info os-modules* on page 2-83
  - *info os-version* on page 2-84
  - *info processes* on page 2-85
  - *set os* on page 2-139
  - *show os* on page 2-166.

Chapter 7
Debugging with command scripts

The following topics describe how to use scripts containing debugger commands to enable you to automate debugging operations.

Tasks

7.1 Creating a debugger script file

A debugger script file is a text file containing commands to control and debug your target.

7.1.1 Procedure

You can work through a debug session using all the toolbar icons and menu options as required. A full list of all the DS-5 Debugger commands generated during the current debug session is recorded in the History view. Before closing Eclipse, you can select the commands that you want in your script file and click on **Export the selected lines as a script file** to save them to a file.

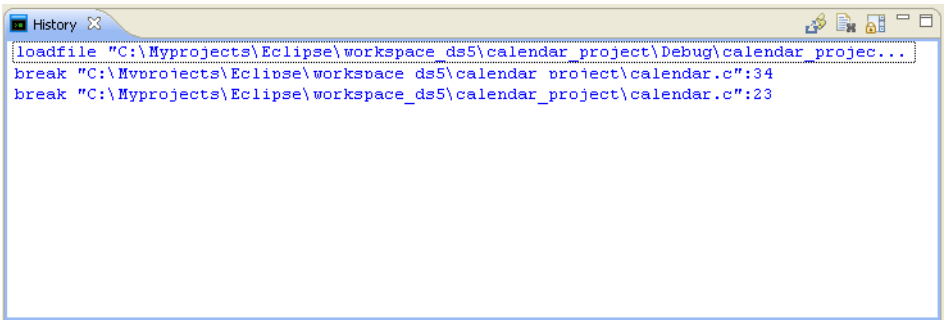

Figure 7-1 History view

Alternatively you can manually create a script file with one debugger command on each line. Each command can be identified with comments if required. It is recommended that you use `.ds` file extensions for DS-5 Debugger script files and `.cmm` or `.t32` file extensions for CMM-style script files.

The following examples show script files using different types of commands that are available for use.

Example 7-1 DS-5 Debugger script file

```
# Initialization commands
load "struct_array.axf"       # Load image
file "struct_array.axf"       # Load symbols

break main                    # Set breakpoint at main()
break *0x814C                 # Set breakpoint at address 0x814C

# Run to breakpoint and print required values
run                           # Start running device
wait 0.5s                     # Wait or time-out after half a second
info stack                    # Display call stack
info registers                # Display info for all registers

# Continue to next breakpoint and print required values
continue                      # Continue running device
wait 0.5s                     # Wait or time-out after half a second
info functions                # Displays info for all functions
info registers                # Display info for all registers
x/3wx 0x8000                  # Display 3 words of memory from 0x8000 (hex)

...
```

```
# Shutdown commands
delete 1                    # Delete breakpoint assigned number 1
delete 2                    # Delete breakpoint assigned number 2
```

Example 7-2 CMM-style script file

```
system.up                   ; Connect to target and device
data.load.elf "hello.axf"   ; Load image and symbols

// Setup breakpoints and registers
break.set main /disable     ; Set breakpoint and immediately disabled
break.set 0x8048            ; Set breakpoint at specified address
break.set 0x8060            ; Set breakpoint at specified address
register.set R0 15          ; Set register R0
register.set PC main        ; Set PC register to symbol address

...

break.enable main           ; Enable breakpoint at specified symbol

// Run to breakpoint and display required values
go                          ; Start running device
var.print "Value is: " myVar ; Display string and variable value
print %h r(R0)              ; Display register R0 in hexadecimal

// Run to breakpoint and print stack
go                          ; Run to next breakpoint
var.frame /locals /caller   ; Display all variables and function callers

...

// Shutdown commands
break.delete main           ; Delete breakpoint at address of main()
break.delete 0x8048         ; Delete breakpoint at address
break.delete 0x8060         ; Delete breakpoint at specified address
system.down                 ; Disconnect from target
```

7.1.2 See also

Tasks

- *Launching the debugger from the command-line console* on page 2-7
- *Running a debugger script file in Eclipse* on page 7-4.

Reference

- *Commands view* on page 11-15
- *History view* on page 11-28
- *Scripts view* on page 11-41
- *DS-5 Debugger command-line console keyboard shortcuts* on page 2-11
- *ARM® DS-5™ Debugger Command Reference*:
  - Chapter 2 *DS-5 Debugger commands*
  - Chapter 3 *CMM-style commands supported by the debugger*.

7.2 Running a debugger script file in Eclipse

To run a script

1. Launch Eclipse.
2. Configure a connection to the target. A DS-5 Debugger configuration can include the option to run a script file immediately after the debugger connects to the target. To do this select the script file in the Debugger tab of the DS-5 Debug configuration dialog box.
3. Connect to the target.
4. To run a script file whilst a debug session is in progress:
   a. Click on the Scripts view
   b. Import one or more script files in the order that you want them to be executed.
   c. Select the scripts that you want to execute.
   d. Click on the **Execute Selected Scripts** toolbar icon.

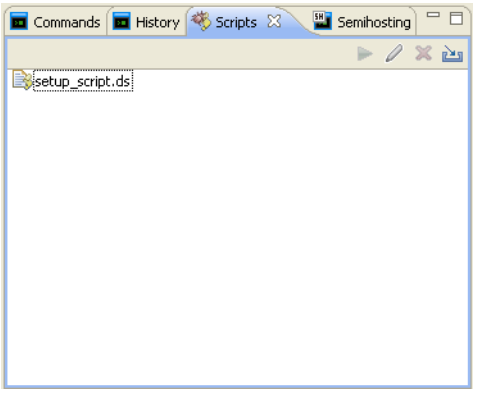

Figure 7-2 Scripts view

Note

To execute CMM-style commands within Eclipse you must create a CMM-style script file containing the relevant commands and then use the DS-5 Debugger `source` command to run the script. For example:

```
source myScripts\myFile.cmm
```

7.2.1 See also

Tasks

- *Launching the debugger from Eclipse* on page 2-6
- *Configuring a connection to an RTSM model* on page 3-3
- *Configuring a connection to a Linux target using gdbserver* on page 3-5
- *Configuring a connection to a bare-metal target* on page 3-9
- *Creating a debugger script file* on page 7-2.

Reference

- *Commands view* on page 11-15
- *History view* on page 11-28
- *Scripts view* on page 11-41

- *ARM® DS-5™ Debugger Command Reference*:
  - *source* on page 2-179.

Chapter 8
Controlling runtime messages

The following topics describe semihosting and how to control runtime messages.

Tasks

Concepts

8.1 About semihosting and top of memory

Semihosting is typically used when debugging an application that is using the C library and running without an operating system. This enables functions in the C library, such as `printf()` and `scanf()`, to use the screen and keyboard on the host workstation instead of having a screen and keyboard on the target system.

Semihosting uses stack base and heap base addresses to determine the location and size of the stack and heap.

The stack base, also known as the top of memory, is an address that is by default 64K from the end of the heap base.

The heap base is by default contiguous to the application code.

The following figure shows a typical layout for an ARM® target.

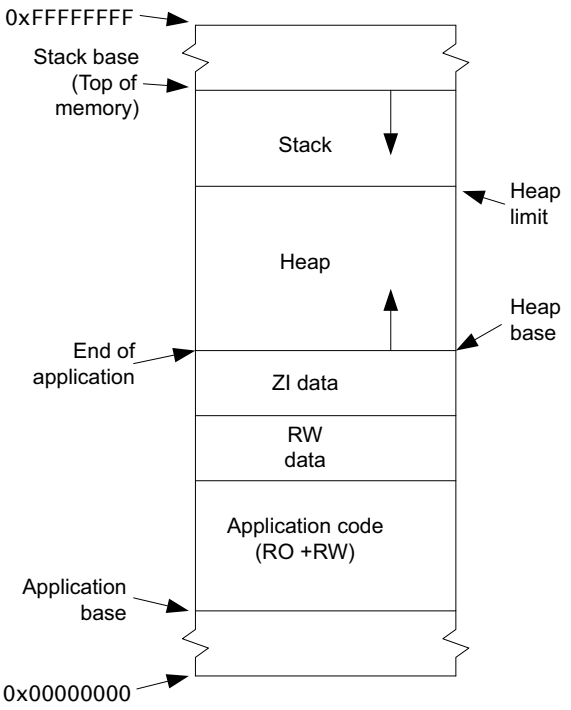

Figure 8-1 Typical layout between top of memory, stack, and heap

8.1.1 See also

Task

- *Working with semihosting* on page 8-4
- *Enabling automatic semihosting support in the debugger* on page 8-5
- *Controlling semihosting messages using the command-line console* on page 8-6.

Reference

- *App Console view* on page 11-3
- *Debug Configurations - Connection tab* on page 11-59
- *ARM® DS-5™ Debugger Command Reference*:
  - *set semihosting* on page 2-141

— *show semihosting* on page 2-168.

8.2 Working with semihosting

Semihosting is supported by the debugger in both the command-line console and in Eclipse.

Command-line console

By default all semihosting messages (`stdout` and `stderr`) are output to the console. When using this console interactively with debugger commands you must use the `stdin` command to send input messages (`stdin`) to the application.

Alternatively, you can disable semihosting in the console and use a separate telnet session to interact directly with the application. During start up, the debugger creates a semihosting server socket and displays the port number to use for the telnet session.

Eclipse The App Console view within the DS-5 Debug perspective controls all the semihosting input/output requests (`stdin`, `stdout`, and `stderr`) between the application code and the debugger.

8.2.1 See also

Tasks

- *Enabling automatic semihosting support in the debugger* on page 8-5
- *Controlling semihosting messages using the command-line console* on page 8-6.

Concepts

- *About semihosting and top of memory* on page 8-2.

Reference

- *App Console view* on page 11-3
- *ARM® DS-5™ Debugger Command Reference:*
  - *set semihosting* on page 2-141
  - *show semihosting* on page 2-168.

8.3 Enabling automatic semihosting support in the debugger

By default, semihosting support is not enabled in the debugger. However, you can create a special semihosting function, `__auto_semihosting(void)` in your C code with an alias to another function. This places the required symbol in the debug information but not in the main application image. When the debugger detects that symbol then it automatically enables semihosting operations if the target supports it.

Example 8-1 Create a special semihosting function with an alias to another function

```
#include <stdio.h>
void __auto_semihosting(void) __attribute__ ((alias("main")));
                                        //mark as alias for main() to declare
                                        //semihosting symbol in debug information only
int main(void)
{
    printf("Hello world\n");
    return 0;
}
```

8.3.1 See also

Tasks

- *Working with semihosting* on page 8-4
- *Controlling semihosting messages using the command-line console* on page 8-6.

Concepts

- *About semihosting and top of memory* on page 8-2.

Reference

- *App Console view* on page 11-3
- *ARM® DS-5™ Debugger Command Reference:*
  - *set semihosting* on page 2-141
  - *show semihosting* on page 2-168.

8.4 Controlling semihosting messages using the command-line console

You can control input/output requests from application code to a host workstation running the debugger. These are called semihosting messages.

By default, all messages are output to the command-line console but you can choose to redirect them when launching the debugger by using one or more of the following:

`--disable_semihosting`

Disables all semihosting operations.

`--disable_semihosting_console`

Disables all semihosting operations to the debugger console.

`--semihosting_error=`*`filename`*

Specifies a file to write `stderr` for semihosting operations.

`--semihosting_input=`*`filename`*

Specifies a file to read `stdin` for semihosting operations.

`--semihosting_output=`*`filename`*

Specifies a file to write `stdout` for semihosting operations.

8.4.1 See also

Tasks

- *Working with semihosting* on page 8-4.

Concepts

- *About semihosting and top of memory* on page 8-2.

Reference

- *Launching the debugger from the command-line console* on page 2-7.

8.5 Controlling the output of logging messages

You can control logging messages from the debugger. By default, all messages are output to the Console view but you can control the output and redirection of logging messages by using the `log config` and `log file` debugger commands:

`log config=`*`option`*

Specifies the type of logging configuration to output runtime messages from the debugger:

Where:

`option` Specifies a predefined logging configuration or a user-defined logging configuration file:

- `info` Output messages using the predefined INFO level configuration. This is the default.
- `debug` Output messages using the predefined DEBUG level configuration.
- *`filename`* Specifies a user-defined logging configuration file to customize the output of messages. The debugger supports log4j configuration files.

`log file=`*`filename`*

Output messages to a file in addition to the console.

8.5.1 See also

Tasks

- *Customizing the output of logging messages from the debugger* on page 8-9.

Reference

- *Commands view* on page 11-15
- *ARM® DS-5™ Debugger Command Reference*:
  - *log config* on page 2-104
  - *log file* on page 2-105.

Other information

- *log4j logging services on the Apache website*, `http://logging.apache.org`.

8.6 About Log4j configuration files

In general, the predefined logging configurations provided by the debugger are sufficient for most debugging tasks. However, if you want finer control then you can specify your own customized logging configuration by creating a log4j configuration file. Log4j is an open source logging system for the Java platform and the debugger currently uses version 1.2.

Log4j uses a hierarchy of logging levels to control messages with each level inheriting all lower levels. The following logging levels are currently supported by the debugger:

- DEBUG
- INFO
- WARN
- ERROR
- FATAL.

Messages are assigned to a specific logging level and can be redirected to different output locations using one or more of the following log4j components:

Table 8-1 Log4j Components

Component	Description
Logger	Defines the level of logging.
Appender	Defines the output destination.
Layout	Defines the message format.

8.6.1 See also

Tasks

- *Controlling the output of logging messages* on page 8-7
- *Customizing the output of logging messages from the debugger* on page 8-9.

Other information

- *log4j logging services on the Apache website*, `http://logging.apache.org`.

8.7 Customizing the output of logging messages from the debugger

To create a customized log4j configuration file:

1. Create an `Appender` instance for the required logging type. The following types are supported:
   - `ConsoleAppender`
   - `RollingFileAppender`.
2. Suppress the `Threshold` logging level, if required.
3. If the `Appender` instance outputs to a file, define the layout for the `Appender` instance. The following layouts are supported:

`PatternLayout`	Textual format.
`HTMLLayout`	HTML format.

4. If the `Appender` instance outputs to a file, define the file components. The following components are supported:

`File`	File name
`MaxFileSize`	Long integer or string, for example `10KB`.
`MaxBackupIndex`	Maximum number of log files to use. The default is 1.

5. If you use the layout `PatternLayout`, you can enhance the format of a message by using an additional `ConversionPattern` component. The following patterns are supported:

`%c`	Logging category
`%C`	Class name
`%d`	Date
`%F`	Filename
`%l`	Caller location
`%L`	Line number
`%m`	Logging message
`%M`	Method name
`%n`	End of line character
`%p`	Logging level. For alignment, you can also supply the number of characters, for example: `%-5p`.
`%r`	Elapsed time (milliseconds)
`%t`	Thread name.

6. Define the `name` component for the `Appender` instance, if required.
7. Define the logging level for the `rootLogger` and assign to the required `Appender` instance.
8. To pass the log4j configuration file to the debugger you can use:
   - `--log_config=`*`filename`* command-line option when launching the debugger from the command-line console.
   - `log config` *`filename`* debugger command if the debugger is already running.

8.7.1 Example showing how to log messages to the console

The following example shows how to log messages to the console. This sets the default logging level to DEBUG. All the logging for this example is output to the console. However the output of error and warning messages are sent to the error stream, and debug and info messages are sent to the output stream.

Example 8-2 Logging messages to the console

```
# Setup logConsole to be a ConsoleAppender
log4j.appender.logConsole=org.apache.log4j.ConsoleAppender
log4j.appender.logConsole.layout=org.apache.log4j.PatternLayout
log4j.appender.logConsole.layout.ConversionPattern=%m%n
log4j.appender.logConsole.name=Console

# Send all DEBUG level logs to the console
log4j.rootLogger=DEBUG, console
```

8.7.2 Example showing how to log messages to a file

The following example shows how to log messages to a file. This sets the default logging level to DEBUG. However some packages only write logs at the INFO level. All the logging for this example is output to a file.When the file reaches 10MB, it is renamed by adding `.1` file extension and logging continues to write to a new file with the original name. This happens multiple times, but only ten backup files are stored.

Example 8-3 Logging messages to a file

```
# Setup logFile to be a RollingFileAppender
log4j.appender.logFile=org.apache.log4j.RollingFileAppender
log4j.appender.logFile.File=output.log
log4j.appender.logFile.MaxFileSize=10MB
log4j.appender.logFile.MaxBackupIndex=10
log4j.appender.logFile.layout=org.apache.log4j.PatternLayout
log4j.appender.logFile.layout.ConversionPattern=%d %-5p %t %c - %m%n

# Send all DEBUG level logs to a file: logFile
log4j.rootLogger=DEBUG, logFile

# Send all INFO level logs in the debug packages to the file: logFile
log4j.logger.com.arm.debug.logging=INFO, logFile
```

8.7.3 Example showing how to combine the logging of messages to the console and a file

The following example shows a combination of the previous examples. This sets the default logging level to INFO. All the INFO level logging for this example is output to the console. However, a selection of messages are also sending output to two files.

Example 8-4 Combination of logging messages

```
# Setup logConsole to be a ConsoleAppender
log4j.appender.logConsole=org.apache.log4j.ConsoleAppender
# Suppress all logs to the console that are lower than the threshold
log4j.appender.logConsole.Threshold=INFO
log4j.appender.logConsole.layout=org.apache.log4j.PatternLayout
log4j.appender.logConsole.layout.ConversionPattern=%m%n
log4j.appender.logConsole.name=Console

# Setup logConnFile to be a RollingFileAppender
log4j.appender.logConnFile=org.apache.log4j.RollingFileAppender
# Suppress all logs to the file that are lower than the threshold
log4j.appender.logConnFile.Threshold.DEBUG
```

```
log4j.appender.logConnFile.File=connection.log
log4j.appender.logConnFile.MaxFileSize=10MB
log4j.appender.logConnFile.MaxBackupIndex=10
log4j.appender.logConnFile.layout=org.apache.log4j.PatternLayout
log4j.appender.logConnFile.layout.ConversionPattern=%d %-5p %t %c - %m%n

# Setup logTAccessFile to be a RollingFileAppender
log4j.appender.logTAccessFile=org.apache.log4j.RollingFileAppender
# Suppress all logs to the file that are lower than the threshold
log4j.appender.logTAccessFile.Threshold.DEBUG
log4j.appender.logTAccessFile.File=target_acccess.log
log4j.appender.logTAccessFile.MaxFileSize=10MB
log4j.appender.logTAccessFile.MaxBackupIndex=10
log4j.appender.logTAccessFile.layout=org.apache.log4j.PatternLayout
log4j.appender.logTAccessFile.layout.ConversionPattern=%d %-5p %t %c - %m%n

# Send all INFO logs to the console
log4j.rootLogger=INFO, logConsole

# Send all DEBUG logs in the connection package to the file: logConnFile
log4j.logger.com.arm.debug.core.engine.connection=DEBUG, logConnFile

# Send all DEBUG logs in the targetaccess package to the file: logTAccessFile
log4j.logger.com.arm.debug.core.targetaccess.rvi=DEBUG, logTAccessFile
```

8.7.4 See also

Tasks

- *Controlling the output of logging messages* on page 8-7.

Concepts

- *About Log4j configuration files* on page 8-8.

Reference

- *Commands view* on page 11-15
- *ARM® DS-5™ Debugger Command Reference*:
  - *log config* on page 2-104
  - *log file* on page 2-105.

Other information

- *log4j logging services on the Apache website*, `http://logging.apache.org`.

Chapter 9
Reading and writing to flash memory

The following topics describe how to write complete binaries to flash, or to specific locations in flash.

Tasks

9.1 Registering a new flash algorithm

Note

To use this feature you must have a valid ARM® Compiler license file.

Before carrying out any flash programming operations you must register your flash algorithm with the debugger. You can use the `flash list` command to display all the registered flash algorithms.

Flash algorithms are comprised of one or more object files containing code that runs on the target to access the flash device. An XML file describes these object files along with some additional meta data. The XML file must be registered with the debugger before the flash algorithm can be used.

9.1.1 Prerequisites

Before registering a flash algorithm you must ensure that you have a set of object files that are generated by compiling the algorithm source files.

9.1.2 Procedure

To register a new flash algorithm:

1. Create an XML file that describes the algorithm that you want to register. The XML tags include the following:

   `id` — Unique name containing only alphanumeric characters, hyphens or underscore characters.

   `type` — NOR or NAND. Used to represent the type of flash device.

   `byteOrder` — LE, BE8, or BE32. Used to represent the endianness of the flash device.

   `microlib` — True or false. Specifies whether to use the microlib C library or the standard C library.

   `name` — User-friendly name. You can specify multiple name tags with different language attributes.

   `param` — Dependent on the algorithm. List consisting of `key` and `defaultValue` attributes that are required by the algorithm.

   `object` — Dependent on the algorithm. Lists all the object files and relative paths to the XML file.

Example 9-1 Typical XML configuration file containing two devices

```
<flashDevices>
    <flashDevice id="IntegratorAPDevice"
        type="NOR" cpu="ARM7TDMI" byteOrder="LE" microLib="false">
        <name value="Intel DT28F320S3 2Mx16 x2 x4" lang="en"/>
        <params>
            <param key="baseAddress" defaultValue="0x24000000"/>
            <param key="endAddress" defaultValue="0x24200000"/>
        </params>
        <objects>
            <object file="integratorAP\IntegratorAP.o"/>
        </objects>
    </flashDevice>
    <flashDevice id="IntegratorCPDeviceLE"
```

```
        type="NOR" cpu="ARM7TDMI" byteOrder="LE" microLib="false">
        <name value="Intel DT28F640J3A 64Mbit x2 (le)" lang="en"/>
        <params>
            <param key="baseAddress" defaultValue="0x24000000"/>
            <param key="endAddress" defaultValue="0x24200000"/>
        </params>
        <objects>
            <object file="integratorCP\IntegratorCP-le.o"/>
        </objects>
    </flashDevice>
</flashDevices>
```

2. Use the `flash register` command to find algorithms that are described in XML files and register them with the debugger. For example:

```
flash register "usr\algorithms"     # Register flash algorithms from the
                                     # directory: usr\algorithms
```

9.1.3 See also

Tasks

- *Writing an image to flash memory* on page 9-4.

Reference

- *Commands view* on page 11-15
- *History view* on page 11-28
- *Scripts view* on page 11-41
- *ARM® DS-5™ Debugger Command Reference*:
  - *flash list* on page 2-58
  - *flash register* on page 2-59
  - *flash unregister* on page 2-60.

9.2 Writing an image to flash memory

You must define a suitable memory map for your algorithm before the debugger can carry out flash programming operations.

9.2.1 Prerequisites

Before carrying out any flash programming operations you must register your flash algorithm with the debugger.

Note

To use this feature you must have a valid Compiler license file.

9.2.2 Procedure

To set up the memory map and write an image to flash memory:

1. Use the `mem flash` command to define a region of flash memory and insert it into the memory map for the target. For example:

   ```
   memory flash 0x08000000 0x0801FFFF Keil.STM32F10x_128.FLX clockSpeed=8000000
              # Defines a flash region for Keil.STM32F10x_128.FLX algorithm
   ```

2. Use the `mem` command to define a region of RAM and insert it into the memory map for the target. For example:

   ```
   memory 0x20000000 0x20004FFF     # specify RW region 0x20000000 0x20004FFF
   ```

3. Use the `info mem` command to verify the memory map for the target.

4. Use the `set flash-buffer` command to define a region of RAM memory for use as a buffer when programming flash memory. The memory range is written to as part of flash programming, and the original content is not restored afterwards. For example, to define the buffer shown in the figure you can use the following:

   ```
   set flash-buffer 0x20000100 0x20004EFF # set flash buffer
   ```

5. When all the regions are set up then you can use the same commands that you normally use when writing to RAM.

   For example:

   ```
   load myFlashImage.axf                 # Image that loads into flash
                                         # memory addresses
   restore myFile.bin binary 0x08000000  # Restore content of binary file
                                         # myFile.bin starting at 0x08000000
   ```

The following figure shows the memory map for this example.

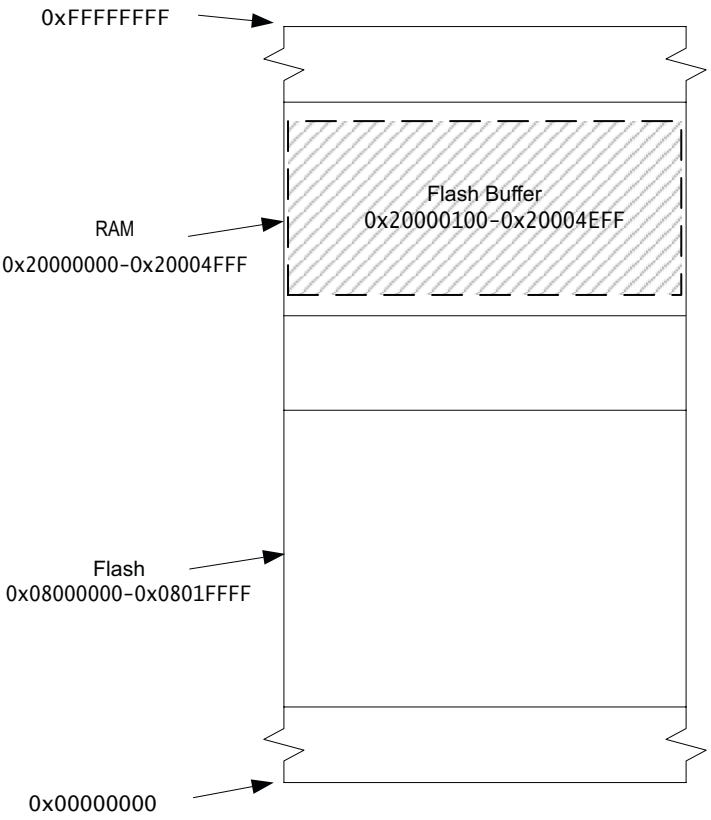

Figure 9-1 Memory map showing flash regions

9.2.3 See also

Tasks

- *Registering a new flash algorithm* on page 9-2.

Reference

- *Commands view* on page 11-15
- *History view* on page 11-28
- *Scripts view* on page 11-41
- *ARM® DS-5™ Debugger Command Reference*:
  - *info memory* on page 2-81
  - *load* on page 2-101
  - *memory* on page 2-106
  - *memory flash* on page 2-110
  - *set flash-buffer* on page 2-137
  - *show flash-buffer* on page 2-164
  - *restore* on page 2-124.

Chapter 10
Working with the Snapshot Viewer

The following topics describe how to use the Snapshot Viewer.

Tasks

Concepts

10.1 Creating a Snapshot Viewer initialization file

The Snapshot Viewer initialization file is a simple text file consisting of one or more sections that emulate the state of the original system. Each section uses an *option*=*value* structure.

Note

You must use `.ini` for the file extension.

10.1.1 Prerequisites

Before creating a Snapshot Viewer initialization file you must ensure that you have:

- One or more binary files containing a snapshot of the application that you want to analyze.

  Note

  The binary files must be formatted correctly in accordance with the following restrictions.

- Details of the type of processor.
- Details of the memory region addresses and offset values.
- Details of the last known register values.

10.1.2 Procedure

To create a Snapshot Viewer initialization file, you must add grouped sections as required from the following list.

`[global]` A section for global settings. The following option can be used:

- `core` The selected processor, for example, `core=Cortex-M3`.

`[dump]` One or more sections for contiguous memory regions stored in a binary file. The following options can be used:

- `file` Location of the binary file.
- `address` Memory start address for the specified region.
- `length` Length of the region. If none specified then the default is the rest of file from the offset value.
- `offset` Offset of the specified region from the start of the file. If none specified then the default is zero.

`[regs]` A section for standard ARM® register names and values, for example, `R0=0x0`.

Banked registers can be explicitly specified using their names from the *ARM Architecture Reference Manual*, for example, `R13_fiq`. In addition, the current mode is determined from the *Program Status Registers* (PSRs), allowing register names without mode suffixes to be identified with the appropriate banked registers.

The values of the PSRs and PC registers must always be provided. The values of other registers need only be provided if it is intended to read them from the debugger.

Consider:

```
[regs]
CPSR=0x600000D2 ; IRQ
SP=0x8000
R14_irq=0x1234
```

Reading the registers named SP, R13, or R13_irq all yield the value `0x8000`.

Reading the registers named LR, R14, or R14_irq all yield the value `0x1234`.

Note

All registers are 32-bits.

10.1.3 Restrictions

The following restrictions apply:

- If you require a global section then it must be the first in the file.
- Consecutive bytes of memory must appear as consecutive bytes in one or more dump files.
- Address ranges representing memory regions must not overlap.

10.1.4 Example

Example 10-1 Snapshot Viewer Initialization file

```
; All sections are optional

[global]
core=Cortex-M3               ; Selected processor

; Location of a contiguous memory region stored in a dump file
[dump]
file="path/dumpfile1.bin"    ; File location (full path must be specified)
address=0x8000               ; Memory start address for specific region
length=0x0090                ; Length of region
                             ; (optional, default is rest of file from offset)

; Location of another contiguous memory region stored in a dump file
[dump]
file="path/dumpfile2.bin"    ; File location
address=0x8090               ; Memory start address for specific region
offset=0x0024                ; Offset of region from start of file
                             ; (optional, default is 0)

; ARM registers
[regs]
R0=0x000080C8
R1=0x0007C000
R2=0x0007C000
R3=0x0007C000
R4=0x00000363
R5=0x00008EEC
R6=0x00000000
R7=0x00000000
R8=0x00000000
R9=0xB3532737
R10=0x00008DE8
R11=0x00000000
R12=0x00000000
```

```
SP=0x0007FFF8
LR=0x0000808D
PC=0x000080B8
```

10.1.5 See also

Tasks

- *Launching the debugger from Eclipse* on page 2-6
- *Connecting to the Snapshot Viewer* on page 10-7
- *Considerations when creating debugger scripts for the Snapshot Viewer* on page 10-8.

Concepts

- *About the Snapshot Viewer* on page 10-5.

Reference

- *Launching the debugger from the command-line console* on page 2-7
- *DS-5 Debugger command-line console keyboard shortcuts* on page 2-11
- *ARM Architecture Reference Manual*, http://infocenter.arm.com/help/topic/com.arm.doc.set.architecture.

10.2 About the Snapshot Viewer

The Snapshot Viewer can be used to analyze a snapshot representation of the application state in scenarios where interactive debugging is not possible. To enable debugging of an application using the Snapshot Viewer, you must have the following data:

- register values
- memory values
- debug symbols.

If you are unable to provide all of this data then the level of debug that is available is compromised. Capturing this data is specific to your application, and no tools are provided to help with this. You might have to install exception or signal handlers to catch erroneous situations in your application and dump the required data out.

You must also consider how to get the dumped data from your device onto a workstation that is accessible by the debugger. Some suggestions on how to do this are to:

- write the data to a file on the host workstation using semihosting
- send the data over a UART to a terminal
- send the data over a socket using TCP/IP.

10.2.1 Register values

Register values are used to emulate the state of the original system at a particular point in time. The most important registers are those in the current processor mode. For example, on an ARMv4 architecture processor these registers are R0-R15 and also the *Program Status Register*s (PSRs):

- *Current Program Status Register* (CPSR)
- *Application Program Status Register* (APSR)
- *Saved Program Status Register* (SPSR).

Be aware that on many ARM® processors, an exception, for example a data abort, causes a switch to a different processor mode. In this case you must ensure that the register values you use reflect the correct mode in which the exception occurred, rather than the register values within your exception handler.

If your application uses floating-point data and your device contains vector floating-point hardware, then you must also provide the Snapshot Viewer with the contents of the vector floating-point registers. The important registers to capture are:

- *Floating-point Status and Control Register* (FPSCR)
- *Floating-Point EXCeption register* (FPEXC)
- *Single precision registers* (S*n*)
- *Double precision registers* (D*n*)
- *Quad precision registers* (Q*n*).

10.2.2 Memory values

The majority of the application state is usually stored in memory in the form of global variables, the heap and the stack. Due to size constraints, it is often difficult to provide the Snapshot Viewer with a copy of the entire contents of memory. In this case you must carefully consider the areas of memory that are of particular importance.

If you are debugging a crash, the most useful information to find out is often the call stack, because this shows the calling sequence of each function prior to the exception and the values of all the respective function parameters. To show the call stack the debugger must know the current stack pointer and have access to the contents of the memory that contains the stack. By

default, on ARM processors the stack grows downwards, so you need to provide the memory starting from the current stack pointer and going up in memory until the beginning of the stack is reached. If you are unable to provide the entire contents of the stack, then a smaller portion starting at the current stack pointer is still useful because it provides the most recent function calls.

If your application uses global (**`extern`** or file **`static`**) data, then providing the corresponding memory values enables you to view the variables within the debugger.

If you have local or global variables that point to heap data, then you might want to follow the relevant pointers in the debugger to examine the data, and in order to do this you must have provided the contents of the heap to the Snapshot Viewer. Be aware that heaps can often occupy a large memory range, so it might not be possible to capture the entire heap. The layout of the heap in memory and the data structures that control heap allocation are often specific to the application or the C library, see the relevant documentation for more information.

In order to debug at the disassembly level, the debugger must have access to the memory values where the application code is located. It is often not necessary to capture the contents of the memory containing the code, because identical data can often be extracted directly from the image using processing tools such as `fromelf`. However, some complications to be aware of are:

- self-modifying code where the values in the image and memory can vary
- dynamic relocation of the memory address within the image at runtime.

10.2.3 Debug symbols

The debugger needs to access debug information in order to display high-level information about your application, for example:

- source code
- variable values and types
- structures
- call stack.

This information is stored by the compiler and linker within the application image, so you must ensure that you have a local debug copy of the same image that you are running on your device. The amount of debug information that is stored in the image, and therefore the resulting quality of your debug session, can be affected by the debug and optimisation settings passed to the compiler and linker.

It is common to strip out as much of the debug information as possible when running an image on an embedded device. In such cases, try to use the original unstripped image for debugging purposes.

10.2.4 See also

Tasks

- *Connecting to the Snapshot Viewer* on page 10-7
- *Creating a Snapshot Viewer initialization file* on page 10-2
- *Considerations when creating debugger scripts for the Snapshot Viewer* on page 10-8.

10.3 Connecting to the Snapshot Viewer

A Snapshot Viewer provides a virtual target that you can use to analyze a snapshot of a known system state using the debugger.

10.3.1 Prerequisites

Before connecting you must ensure that you have a Snapshot Viewer initialization file containing static information about a target at a specific point in time. For example, the contents of registers, memory and processor state.

10.3.2 Procedure

Launch the debugger in the command-line console using `--target` command-line option to pass the Snapshot Viewer initialization file to the debugger. For example:

```
debugger --target=int.ini --script=int.cmm
```

10.3.3 See also

Tasks

- *Launching the debugger from Eclipse* on page 2-6
- *Creating a Snapshot Viewer initialization file* on page 10-2
- *Considerations when creating debugger scripts for the Snapshot Viewer* on page 10-8.

Concepts

- *About the Snapshot Viewer* on page 10-5.

Reference

- *Launching the debugger from the command-line console* on page 2-7
- *DS-5 Debugger command-line console keyboard shortcuts* on page 2-11.

10.4 Considerations when creating debugger scripts for the Snapshot Viewer

The Snapshot Viewer uses an initialization file that emulates the state of the original system. The symbols are loaded from the image using the `data.load.elf` command with the `/nocode /noreg` arguments.

The snapshot data and registers are read-only and so the commands you can use are limited.

The following example shows a script using CMM-style commands to analyze the contents of the `types_m3.axf` image.

Example 10-2 CMM-style script file

```
var.print "Connect and load symbols:"
system.up
data.load.elf "types_m3.axf"  /nocode /noreg

;Arrays and pointers to arrays
var.print ""
var.print "Arrays and pointers to arrays:"
var.print "Value of i_array[9999] is " i_array[9999]
var.print "Value of *(i_array+9999) is " *(i_array+9999)
var.print "Value of d_array[1][5] is " d_array[1][5]
var.print "Values of *((*d_array)+9) is " *((*d_array)+9)
var.print "Values of *((*d_array)) is " *((*d_array))
var.print "Value of &d_array[5][5] is " &d_array[5][5]

;Display 0x100 bytes from address in register PC
var.print ""
var.print "Display 0x100 bytes from address in register PC:"
data.dump r(PC)++0x100

;Structures and bit-fields
var.print ""
var.print "Structures and bit-fields:"
var.print "Value of values2.no is " values2.no
var.print "Value of ptr_values->no is " ptr_values->no
var.print "Value of values2.name is " values2.name
var.print "Value of ptr_values->name is " ptr_values->name
var.print "Value of values2.name[0] is " values2.name[0]
var.print "Value of (*ptr_values).name is " (*ptr_values).name
var.print "Value of values2.f1 is " values2.f1
var.print "Value of values2.f2 is " values2.f2
var.print "Value of ptr_values->f1 is " ptr_values->f1

var.print ""
var.print "Disconnect:"
system.down
```

10.4.1 See also

Tasks

- *Launching the debugger from Eclipse* on page 2-6
- *Connecting to the Snapshot Viewer* on page 10-7
- *Creating a Snapshot Viewer initialization file* on page 10-2.

Concepts

- *About the Snapshot Viewer* on page 10-5.

Reference

- *Launching the debugger from the command-line console* on page 2-7
- *DS-5 Debugger command-line console keyboard shortcuts* on page 2-11.

Chapter 11
DS-5 Debug perspective and views

The following topics describe the DS-5 Debug perspective and related views in the Eclipse *Integrated Development Environment* (IDE).

Reference

- *Breakpoint properties dialog box* on page 11-52
- *Manage Signals dialog box* on page 11-57
- *Debug Configurations - Connection tab* on page 11-59
- *Debug Configurations - Files tab* on page 11-63
- *Debug Configurations - Debugger tab* on page 11-67
- *Debug Configurations - Arguments tab* on page 11-71
- *Debug Configurations - Environment tab* on page 11-73
- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Using Eclipse*:
  - *Remote Systems view* on page 6-3
  - *Remote System Details view* on page 6-4
  - *Remote Scratchpad view* on page 6-5
  - *Terminals view* on page 6-6.

11.1 App Console view

This view enables you to interact with the console I/O capabilities provided by the semihosting implementation in the ARM® C libraries. To use this feature, semihosting support needs to be compiled into your application. A standalone target programme with no operating system and using the standard ARM C library has semihosting enabled by default. Many other environments, such as Linux applications, do not use semihosting and instead have alternate means of console and host access.

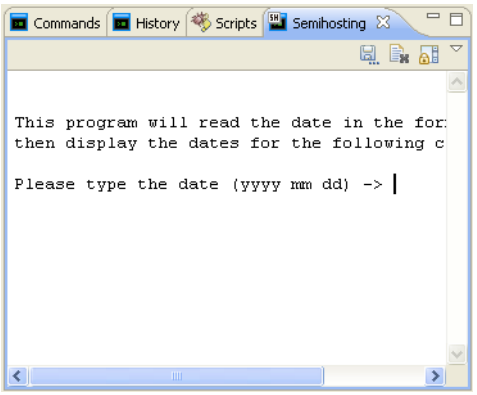

Figure 11-1 App Console view

Note

Default settings for this view are controlled by a DS-5 Debugger setting in the Preferences dialog box. For example, default locations for specific files or the maximum number of lines to display. You can access these settings by selecting **Preferences...** from the **Window** menu.

11.1.1 Toolbar and context menu options

The following options are available from the toolbar or context menu:

Save Console Buffer

Save the contents of the Semihosting Console view to a text file.

Clear Console

Clear the contents of the Semihosting Console view.

Scroll Lock Enable or disable the automatic scrolling of messages in the Semihosting Console view.

View Menu This menu contains the following option:

Bring to Front for Write

If enabled, the debugger automatically changes the focus to this view when a Semihosting application prompts for input.

Copy Copy the selected text.

Paste Paste text that you have previously copied. You can paste text only when the application displays a semihosting prompt.

Select All Select all text.

11.1.2 See also

Tasks

- *Working with semihosting* on page 8-4
- *ARM DS-5 Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Concepts

- *About semihosting and top of memory* on page 8-2.

Reference

- *Target view* on page 11-43
- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Using Eclipse*:
  - *Perspectives and views* on page 3-19.

11.2 ARM Asm Info view

This view enables you to view more information on an ARM® or Thumb® instruction or directive.

When you are editing assembly language source files (`.s`) using the ARM assembler editor, you can access more information by:

1. selecting an instruction or directive
2. pressing F3.

The related documentation is displayed in the ARM Asm Info view. The ARM Asm Info view is automatically displayed when you press F3 as described above.

To manually add this view:

1. Ensure that you are in the DS-5 Debug perspective.
2. Select **Window → Show View → Other...** to open the Show View dialog box.
3. Select the **ARM Asm Info** view from the **DS-5 Debugger** group.

11.2.1 See also

Tasks

- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *ARM assembler editor* on page 11-6
- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Using Eclipse*:
  - *Perspectives and views* on page 3-19.

11.3 ARM assembler editor

The ARM® assembler editor provides syntax highlighting, formatting of code and content assistance for labels in ARM assembly language source files. This editor enables you to:

- edit source code
- view the syntax highlighting
- select an instruction or directive and press F3 to view the related ARM assembler reference information
- use content assist for auto-completion
- create, delete, enable or disable a breakpoint.

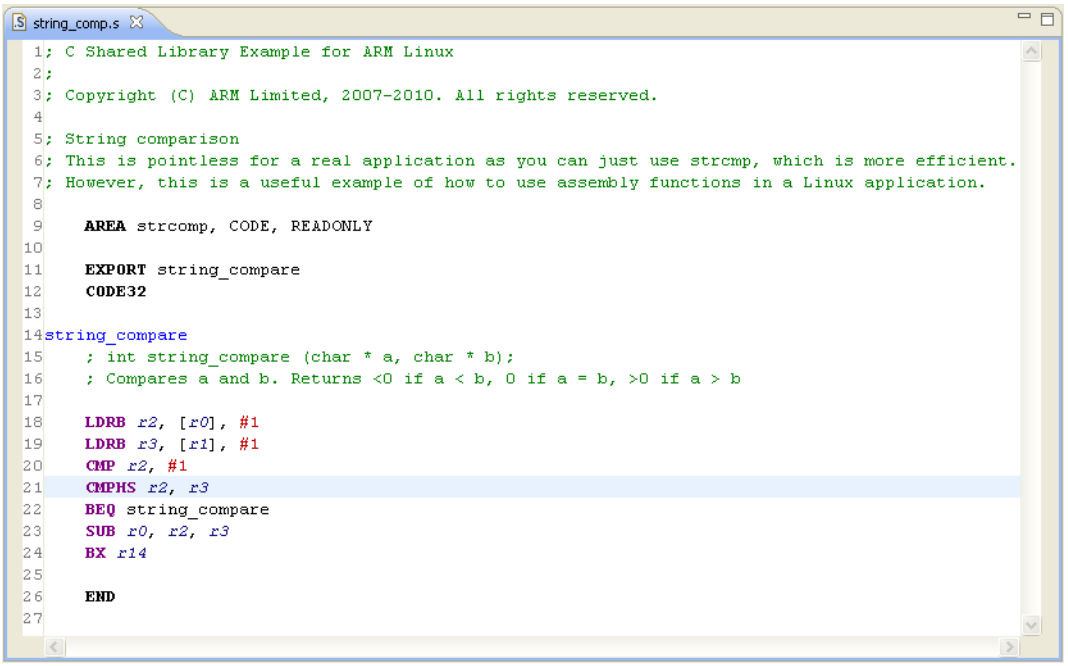

Figure 11-2 ARM assembler editor

In the left-hand margin of each editor tab you can find a marker bar that displays view markers associated with specific lines in the source code.

To set a breakpoint, double click in the marker bar at the position where you want to set the breakpoint. To delete a breakpoint, double-click on the breakpoint marker.

11.3.1 Action context menu options

Right-click in the vertical bar, or the line number column if visible, to display the action context menu for the ARM assembler editor. The options available include:

DS-5 Breakpoints menu

The following breakpoint options are available:

Toggle Breakpoint

Add a new breakpoint, or remove a selected breakpoint.

Disable Breakpoint, Enable Breakpoint

Disable or enable the selected breakpoint.

Breakpoint Properties...

Display the Breakpoint Properties dialog box for the selected breakpoint. This enables you to control breakpoint activation.

Default Breakpoint Type

The following breakpoint options are available:

C/C++ Breakpoints

Select to use the C/C++ perspective breakpoint scheme.

DS-5 C/C++ Breakpoint

Select to use the DS-5 Debug perspective breakpoint scheme. This is the default for the DS-5 Debug perspective.

DS-5 breakpoint markers are red to distinguish them from the blue C/C++ perspective breakpoint markers.

Note

The **Default Breakpoint Type** selected causes the top-level **Toggle Breakpoint** menu in this context menu and the double-click action in the left-hand ruler to toggle either CDT Breakpoints or DS-5 Breakpoints. This menu is also available from the **Run** menu in the main menu bar at the top of the C/C++, Debug, and DS-5 Debug perspectives.

The menu options under **DS-5 Breakpoints** do not honor this setting and always refer to DS-5 Breakpoints.

Show Line Numbers

Show or hide line numbers.

For more information on the other options not listed here, see the dynamic help.

11.3.2 See also

Tasks

- *Examining the target execution environment* on page 5-2
- *Examining the call stack* on page 5-4
- *ARM® DS-5™ Using Eclipse*:
  - *Editing source code* on page 3-25
  - *Accessing the dynamic help* on page 3-35.

Reference

- *ARM Asm Info view* on page 11-5
- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Using Eclipse*:
  - *Perspectives and views* on page 3-19.

11.4 Breakpoints view

This view enables you to:

- disable, enable, or delete breakpoints and watchpoints
- import or export a list of breakpoints and watchpoints
- display the source file containing the line of code where the selected breakpoint is set
- display the disassembly where the selected breakpoint is set
- display the memory where the selected watchpoint is set
- delay breakpoint activation by setting properties for the breakpoint
- control the handling and output of messages for all Unix signals and processor exception handlers
- change the access type for the selected watchpoint.

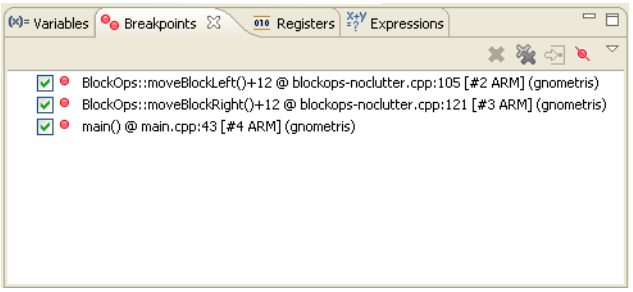

Figure 11-3 Breakpoints view

11.4.1 Syntax of a breakpoint entry

A breakpoint entry has the following syntax:

function+offset `@` *source_file*`:`*linenum* `[#`*ID instruction_type*`, ignore = `*num*`/`*count*`, `*nHits* `hits, (`*condition*`)] (`*image_name*`)`

where:

function+offset

The name of the function in which the breakpoint is set and the number of bytes from the start of the function. For example, `accumulate()+52` shows that the breakpoint is 52 bytes from the start of the `accumulate()` function.

source_file`:`*linenum*

If the source file is available, the file name and line number in the file where the breakpoint is set, for example `threads.c:115`.

ID

The breakpoint ID number, *#N*. In some cases, such as in a **for** loop, a breakpoint might comprise a number of sub-breakpoints. These are identified as *N.n*, where *N* is the number of the parent. The description of a sub-breakpoint in this dialog box is shown as

`main()+132sub-breakpoint ofmain()+132 @ threads.c:56 [#14 ARM] (threads)`

instruction_type

The type of instruction at the address of the breakpoint, `ARM` or `Thumb`.

`ignore = `*num*`/`*count*

An `ignore` count if set, where:

num equals `count` initially, and decrements on each pass until it reaches zero.

count is the value you have specified for the `ignore` count.

nHits `hits` A counter that increments each time the breakpoint is hit. This is not displayed until the first hit. If you set an `ignore` count, `hits` count does not start incrementing until the `ignore` count reaches zero.

condition The stop condition you have specified, for example `(i==3)`.

image_name The name of the image.

11.4.2 Syntax of a watchpoint entry

A watchpoint entry has the following syntax:

`*`*address type*`[`*#ID*`]`

where:

address The address of the variable where the watchpoint is set.

type The access type of the watchpoint.

ID The watchpoint ID number.

11.4.3 Toolbar and context menu options

The following options are available from the toolbar or context menu:

Remove Remove the selected breakpoints and watchpoints.

Remove All Remove all breakpoints and watchpoints.

Go to File Display the source file containing the line of code where the selected breakpoint is set. This option is disabled for a watchpoint.

Go to Disassembly

Display the disassembly where the selected breakpoint is set. This option is disabled for a watchpoint.

Go to Memory

Display the memory where the selected watchpoint is set. This option is disabled for a breakpoint.

Skip All Breakpoints

Deactivate all breakpoints or watchpoints that are currently set. The debugger remembers the enabled and disabled state of each breakpoint or watchpoint, and restores that state when you reactivate them again.

Enable Breakpoints

Enable the selected breakpoints and watchpoints.

Disable Breakpoints

Disable the selected breakpoints and watchpoints.

Resolve Re-evaluate the address of the selected breakpoint or watchpoint. If the address can be resolved the breakpoint or watchpoint is set, otherwise it remains pending.

Properties...

Display the Breakpoint Properties dialog box for the selected breakpoint. This enables you to control breakpoint activation.

Alternatively, display the Watchpoint Properties dialog box for the selected watchpoint. This enables you to change the access type for the selected watchpoint.

Copy Copy the selected breakpoints and watchpoints. You can also use the standard keyboard shortcut to do this.

Paste Paste the copied breakpoints and watchpoints. The breakpoints or watchpoints are enabled by default. You can also use the standard keyboard shortcut to do this.

Select all Select all breakpoints or watchpoints. You can also use the standard keyboard shortcut to do this.

View Menu The following **View Menu** options are available:

Alphanumeric Sort

Sort the list alphanumerically based on the string displayed in the view.

Export Breakpoints

Export the current list of breakpoints and watchpoints to a file.

Import Breakpoints

Import a list of breakpoints and watchpoints from a file.

Manage Signals

Display the Manage Signal dialog box.

Ordered Sort

Sort the list in the order they have been set.

11.4.4 See also

Tasks

- *Setting an execution breakpoint* on page 4-9
- *Setting a data watchpoint* on page 4-11
- *Setting a conditional breakpoint* on page 4-13
- *Pending breakpoints and watchpoints* on page 4-18
- *Handling Unix signals* on page 4-22
- *Handling processor exceptions* on page 4-24
- *Examining the target execution environment* on page 5-2
- *Examining the call stack* on page 5-4
- *ARM DS-5 Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *Breakpoint properties dialog box* on page 11-52
- *Target view* on page 11-43
- *Manage Signals dialog box* on page 11-57
- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Debugger Command Reference*:
  - *awatch* on page 2-26

  - — *clear* on page 2-34
  - — *delete breakpoints* on page 2-40
  - — *disable breakpoints* on page 2-43
  - — *enable breakpoints* on page 2-51
  - — *ignore* on page 2-67
  - — *resolve* on page 2-123
  - — *rwatch* on page 2-126
  - — *set breakpoint* on page 2-133
  - — *set substitute-path* on page 2-147
  - — *watch* on page 2-197.
- *ARM® DS-5™ Using Eclipse*:
  - — *Perspectives and views* on page 3-19.

11.5 C/C++ editor

This editor enables you to:

- edit source code
- view the syntax highlighting
- use content assist for auto-completion
- create, delete, enable or disable a breakpoint.

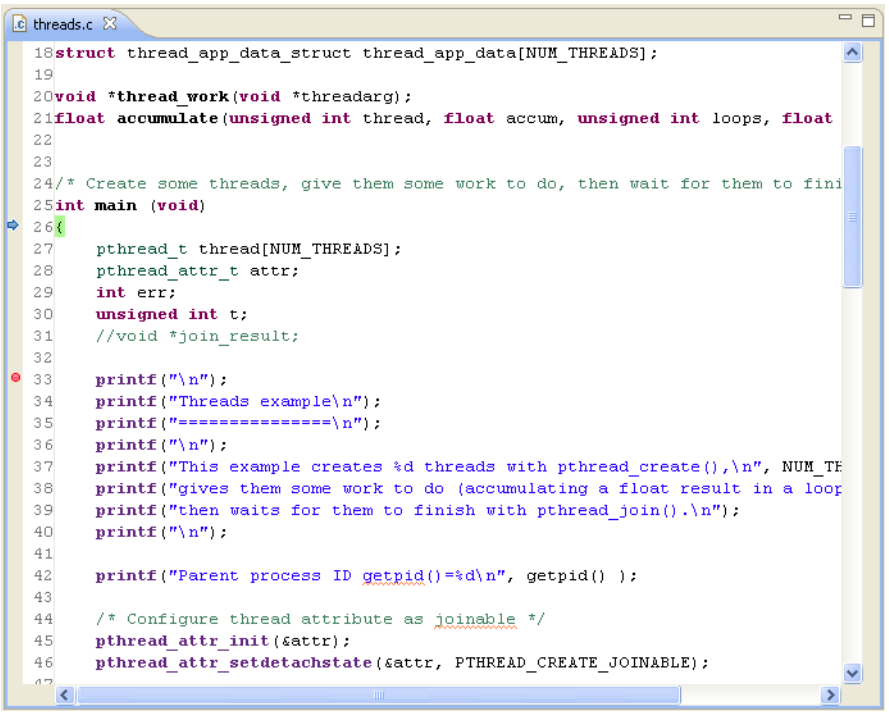

Figure 11-4 C/C++ editor

In the left-hand margin of each editor tab you can find a marker bar that displays view markers associated with specific lines in the source code.

To set a breakpoint, double click in the marker bar at the position where you want to set the breakpoint. To delete a breakpoint, double-click on the breakpoint marker.

Note

If you have sub-breakpoints to a parent breakpoint then double-clicking on the marker also deletes the related sub-breakpoints.

11.5.1 Action context menu options

Right-click in the marker bar, or the line number column if visible, to display the action context menu for the C/C++ editor. The options available include:

DS-5 Breakpoints menu

The following breakpoint options are available:

Toggle Breakpoint Set or remove a breakpoint at the selected address.

Toggle Hardware Breakpoint

Set or remove a hardware breakpoint at the selected address.

Resolve Breakpoint resolve a pending breakpoint at the selected address.

Enable Breakpoint Enable the breakpoint at the selected address.

Disable Breakpoint Disable the breakpoint at the selected address.

Breakpoint Properties...

Display the Breakpoint Properties dialog box for the selected breakpoint. This enables you to control breakpoint activation.

Default Breakpoint Type

The default type causes the top-level context menu entry, **Toggle Breakpoint** and the double-click action in the marker bar to toggle either CDT Breakpoints or DS-5 Breakpoints. When using DS-5 Debugger you must select **DS-5 C/C++ Breakpoint**. DS-5 breakpoint markers are red to distinguish them from the blue CDT breakpoint markers.

Show Line Numbers

Show or hide line numbers.

For more information on the other options not listed here, see the dynamic help.

11.5.2 Editor context menu

Right-click on any line of source to display the editor context menu for the C/C++ editor. The following options are enabled when you connect to a target:

Set PC to Selection

Set the PC to the address of the selected source line.

Run to Selection

Run to the selected source line.

Show disassembly

This option:

1. Opens a new instance of the Disassembly view.
2. Highlights the addresses and instructions associated with the selected source line. A vertical bar and shaded highlight shows the related disassembly.

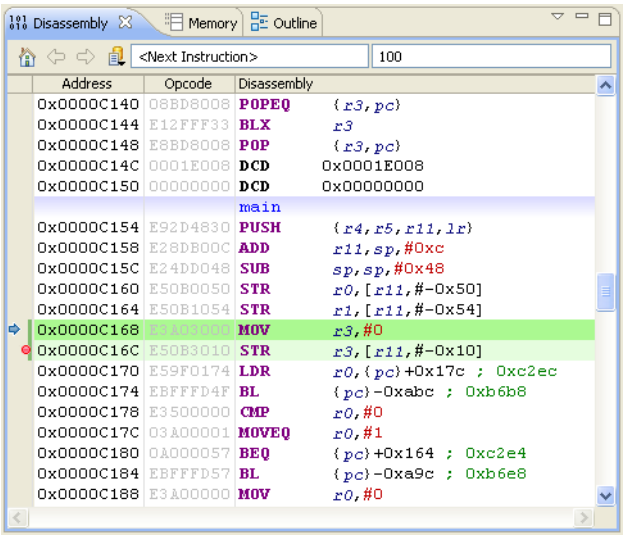

Figure 11-5 Show disassembly for selected source line

For more information on the other options not listed here, see the dynamic help.

11.5.3 See also

Tasks

- *Setting an execution breakpoint* on page 4-9
- *Setting a conditional breakpoint* on page 4-13
- *Settinga breakpoint on a specific thread* on page 4-16
- *Configuring the debugger path substitution rules* on page 4-26
- *Examining the target execution environment* on page 5-2
- *Examining the call stack* on page 5-4
- *ARM® DS-5™ Using Eclipse*:
  - *Editing source code* on page 3-25
  - *Accessing the dynamic help* on page 3-35.

Concepts

- *ARM® DS-5™ Using Eclipse*:
  - *Overview of the C/C++ editor* on page 5-2.

Reference

- *Disassembly view* on page 11-22
- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Debugger Command Reference*:
  - *set substitute-path* on page 2-147.
- *ARM® DS-5™ Using Eclipse*:
  - *Perspectives and views* on page 3-19.

11.6 Commands view

This view enables you to:

- enter debugger commands
- run command scripts
- see messages output by the debugger
- save the contents to a text file.

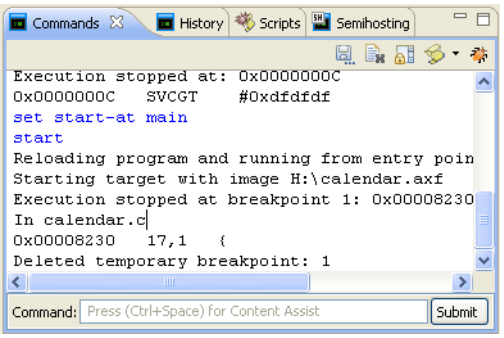

Figure 11-6 Commands view

You can execute DS-5 Debugger commands by entering the command in the field provided, then click **Submit**. This feature is not available until you connect to a target.

You can use the content assist keyboard combination **Ctrl+Space** in the Command field to display a list of DS-5 Debugger commands. Filtering is also possible by entering a partial command. For example, enter `pr` followed by the keyboard combination **Ctrl+Space** to search for the `print` command.

To display sub-commands you must filter on the top level command. For example, enter `info` followed by the keyboard combination **Ctrl+Space** to display all the `info` sub-commands.

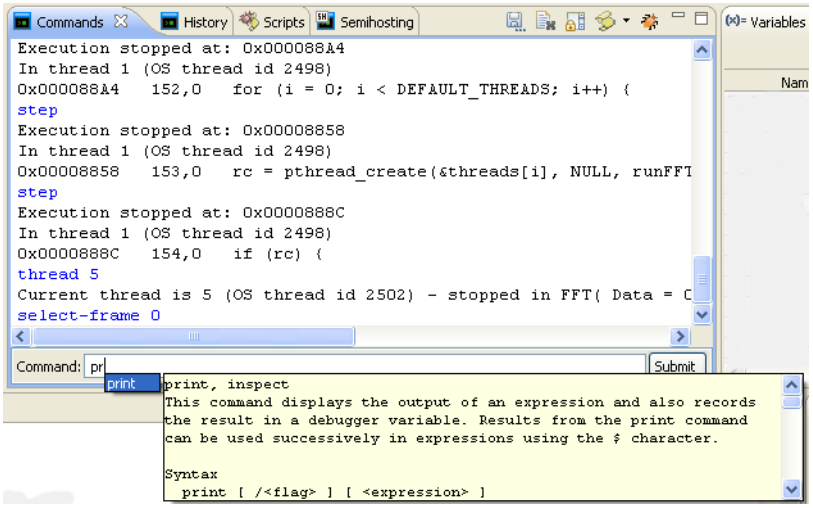

Figure 11-7 Content Assist

Note

Default settings for this view are controlled by a DS-5 Debugger setting in the Preferences dialog box. For example, default locations for specific files or the maximum number of lines to display. You can access these settings by selecting **Preferences...** from the **Window** menu.

11.6.1 Toolbar and context menu options

The following options are available from the toolbar or context menu:

Save Console Buffer

Save the contents of the Commands view to a text file.

Clear Console

Clear the contents of the Commands view.

Scroll Lock Enable or disable the automatic scrolling of messages in the Commands view.

Script menu

A menu of options that enable you to manage and run command scripts:

<Recent scripts list>

A list of the recently run scripts.

<Recent favorites list>

A list of the scripts you have added to your favorites list.

Run Script File...

Display the Open dialog box to select and run a script file.

Organize Favorites...

Display the Scripts view, where you can organize your scripts.

Show Command History View

Display the History view.

Copy Copy the selected commands. You can also use the standard keyboard shortcut to do this.

Paste Paste the command that you have previously copied into the Command field. You can also use the standard keyboard shortcut to do this.

Select all Select all output in the Commands view. You can also use the standard keyboard shortcut to do this.

Save the selected lines as a script...

Display the Save As dialog box to save the selected commands to a script file.

When you click **Save** on the Save As dialog box, you are given the option to add the script file to your favorites list. Click **OK** to add the script to your favorites list. Favorites are displayed in the Scripts view.

Execute selected lines

Run the selected commands.

11.6.2 See also

Tasks

- *ARM® DS-5™ Using Eclipse*:
  - — *Accessing the dynamic help* on page 3-35.

Reference

- *History view* on page 11-28
- *Scripts view* on page 11-41
- *DS-5 Debugger menu and toolbar icons* on page 11-75

- *ARM® DS-5™ Debugger Command Reference*:
  - *DS-5 Debugger commands listed in alphabetical order* on page 2-18
  - *General syntax and usage of DS-5 Debugger commands* on page 2-2.
- *ARM® DS-5™ Using Eclipse*:
  - *Perspectives and views* on page 3-19.

11.7 Debug Control view

This view enables you to:

- view a list of running threads and user space processes as applicable
- view the call stack showing stack elements for each thread or process as applicable
- connect to and disconnect from a target
- load an application image on to the target and debug information when required by the debugger
- start the application and stop at a specific address
- run the application from the beginning
- stop the application
- reset the target
- continue running the application after a breakpoint is hit or the target is suspended
- control the execution of an image by sequentially stepping through an application at the source or instruction level
- set the working directory
- modify the search paths used by the debugger when it executes any of the commands that look up and display source code.

The Debug Control view displays target connections with a hierarchical layout of running threads, user space processes, and related call stacks. All the views in the DS-5 Debug perspective are associated with the current stack frame, shown in bold text, and are updated when the stack frame changes.

Connection states are identified with different icons and background highlighting and are also displayed in the view status bar. The following example shows a connection in the connected state and the others in the disconnected state. If you want to add another configuration to the view then you can use the Debug Control view menu.

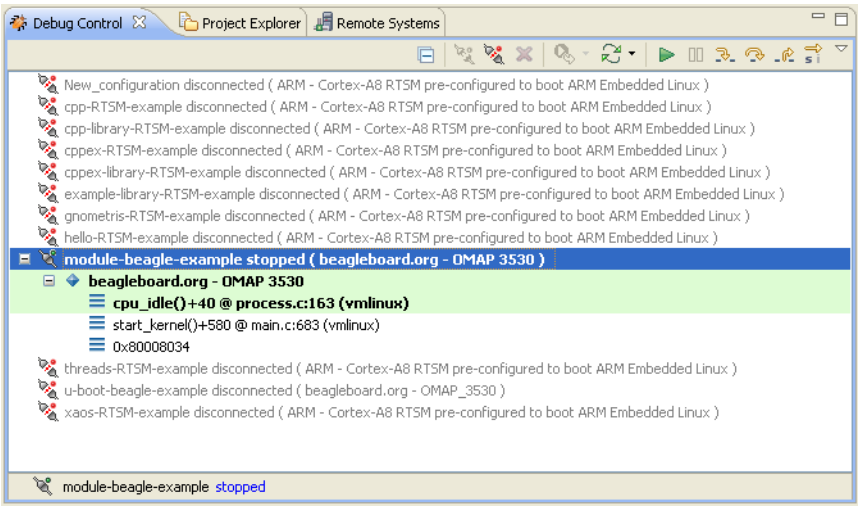

Figure 11-8 Debug Control view

11.7.1 Toolbar and context menu options

The following options are available from the toolbar or context menu:

Collapse All

Collapse all expanded stack trace configurations.

Connect to Target

Connect to the selected target using the same launch configuration settings as the previous connection.

Disconnect from Target

Disconnect from the selected target.

Remove Connection

Remove the selected target connection from the Debug Control view.

Debug from menu

This menu lists the different types of actions that you can perform when a connection is established.

Reset menu

This menu lists the different types of reset that are available on your target.

Run/Continue

Start running the application image from the current location.

After a breakpoint is hit or the target is interrupted, continue running the application.

Note

For a **Connect only** connection you might need to set the PC register to the start of the image before running it.

Suspend Interrupt the target and stop the current application.

Step Source Line, Step Instruction

This option depends on the stepping mode selected:

- If source line mode is selected, step at the source level including stepping into all function calls where there is debug information.
- If instruction mode is selected, step at the instruction level including stepping into all function calls.

Step Over Source Line, Step Over Instruction

This option depends on the stepping mode selected:

- If source line mode is selected, step at the source level but stepping over all function calls.
- If instruction mode is selected, step at the instruction level but stepping over all function calls.

Step Out Continue running to the next instruction after the selected stack frame finishes.

Stepping by Source Line (press to step by instruction), Stepping by Instruction (press to step by source line)

Toggle the stepping mode between source line and instruction.

The Disassembly view and the Source view are automatically displayed when you step in instruction mode.

The Source view is automatically displayed when you step in source line mode. If the target stops in code such as a shared library, and the corresponding source is not available, then the Source view is not displayed.

Debug Configurations...

Displays the Debug Configurations dialog box, with the configuration for the selected connection displayed.

Step Out to This Frame

Continue running to the selected stack frame.

View Menu The following **View Menu** options are available:

Add Configuration (without connecting)...

Display the Add Launch Configuration dialog box. The dialog box lists any configurations that are not already listed in the Debug Control view.

Select one or more configurations, then click **OK**. The selected configurations are added to the Debug Control view, but remain unconnected.

Load... Displays a dialog box where you can select whether to load an image, debug information, an image and debug information, or additional debug information. This option might be disabled for targets where this functionality is not supported.

Set Working Directory...

Display the Current Working Directory dialog box. Enter a new location for the current working directory, then click **OK**.

Path Substitution...

Display the Path Substitution and Edit Substitute Path dialog box.

Use the Edit Substitute Path dialog box to associate the image path with a source file path on the host. Click **OK**. The image and host paths are added to the Path Substitution dialog box. Click **OK** when finished.

Threads Presentation

Display either a flat or hierarchical presentation of the threads in the stack trace.

11.7.2 See also

Tasks

- *Configuring a connection to an RTSM model* on page 3-3
- *Configuring a connection to a Linux target using gdbserver* on page 3-5
- *Configuring a connection to a bare-metal target* on page 3-9
- *Disconnecting from a target* on page 3-11
- *Running an image* on page 4-6
- *Stepping through an application* on page 4-20
- *Configuring the debugger path substitution rules* on page 4-26
- *Examining the call stack* on page 5-4

- *ARM DS-5 Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Concepts

- *Types of target connections* on page 3-2
- *About loading an image on to the target* on page 4-2
- *About loading debug information into the debugger* on page 4-4.

Reference

- *C/C++ editor* on page 11-12
- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Debugger Command Reference*:
  - *add-symbol-file* on page 2-22
  - *continue* on page 2-38
  - *file, symbol-file* on page 2-55
  - *interrupt, stop* on page 2-98
  - *load* on page 2-101
  - *loadfile* on page 2-102
  - *next* on page 2-112
  - *nexti* on page 2-113
  - *nexts* on page 2-114
  - *reset* on page 2-121
  - *run* on page 2-125
  - *set debug-from* on page 2-134
  - *start* on page 2-180
  - *step* on page 2-182
  - *stepi* on page 2-183
  - *steps* on page 2-184
  - *quit, exit* on page 2-120.
- *ARM® DS-5™ Using Eclipse*:
  - *Perspectives and views* on page 3-19.

11.8 Disassembly view

This view enables you to:

- See a disassembly view of the target memory
- Specify the start address for the Disassembly view. You can use expressions in this field, for example `$r3`, or drag and drop a register from the Registers view into the Disassembly view to see the disassembly at the address in that register.
- Select the instruction set for the disassembly view
- Create, delete, enable or disable a breakpoint or watchpoint at a memory location
- Freeze the selected view to prevent the values being updated by a running target.

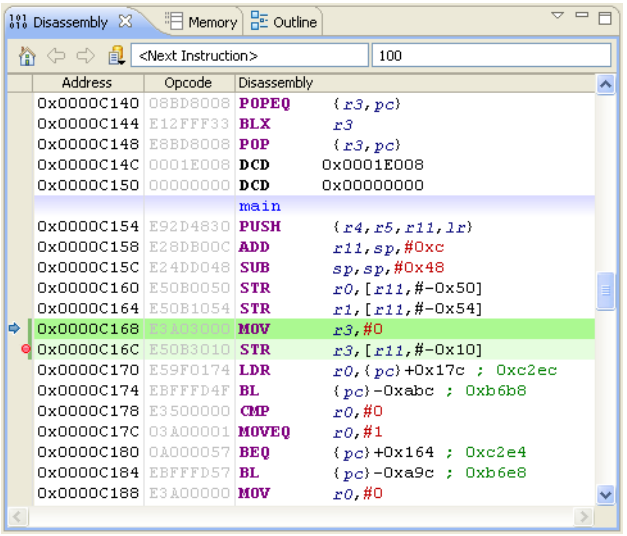

Figure 11-9 Disassembly view

Gradient shading in the Disassembly view shows the start of each function.

Solid shading in the Disassembly view shows the instruction at the address of the current PC register followed by any related instructions that correspond to the current source line.

In the left-hand margin of the disassembly view you can find a marker bar that displays view markers associated with specific locations in the disassembly code.

To set a breakpoint, double click in the marker bar at the position where you want to set the breakpoint. To delete a breakpoint, double-click on the breakpoint marker.

Note

If you have sub-breakpoints to a parent breakpoint then double-clicking on the marker also deletes the related sub-breakpoints.

11.8.1 Toolbar and context menu options

The following options are available from the toolbar or context menu:

Refresh Refresh the view.

Back, Forward

Navigate through the history list.

<Next Instruction> Navigate to the selected stack frame in the Debug Control view.

$LR Navigate to the LR register.

expression Navigate to the address specified by an expression. For example `$PC+256`.

address Navigate to the specified address.

History Addresses and expressions you specify in the Address field are added to the drop down box, and persist until you clear the history list or exit Eclipse. If you want to keep an expression for later use, add it to the Expressions view.

Clear History, <Clear Data>

Clears the list of addresses and expressions in the History drop-down box.

Address field Enter the address where you want to view the disassembly.

Context menu options are available for editing this field.

Size field The number of instructions to display before and after the location pointed to by the program counter.

Context menu options are available for editing this field.

Instruction Set The instruction set to show in the view by default. Select one of the following:

[AUTO] autodetect the instruction set from the image.

ARM ARM® instruction set.

Thumb Thumb® instruction set.

Freeze Data Toggle the freezing of the data in the view. This also disables and enables the Size and Type fields and the **Refresh** option.

New Disassembly View

Display a new instance of the Disassembly view.

Action context menu

When you right-click in the left margin, the corresponding address and instruction is selected and this context menu is displayed. The available options are:

Toggle Breakpoint Set or remove a breakpoint at the selected address.

Toggle Hardware Breakpoint

Set or remove a hardware breakpoint at the selected address.

Resolve Breakpoint Resolve a pending breakpoint at the selected address.

Enable Breakpoint Enable the breakpoint at the selected address.

Disable Breakpoint Disable the breakpoint at the selected address.

Copy Copy the selected address.

Paste Paste into the Address field the last address that you copied.

Select All Select all disassembly in the range specified by the Size field.

If you want to copy the selected lines of disassembly, you cannot use the **Copy** option on this menu. Instead, use the copy keyboard shortcut for your host, for example Ctrl+C on Windows.

Run to Here Run to the selected address

Set PC to Here Set the PC register to the selected address.

Show source If source code is available:

1. Opens the corresponding source file in the C/C++ editor view, if necessary.
2. Highlights the line of source associated with the selected address.

Show in register view

If the memory address corresponds to a register, then displays the Registers view with the related register selected.

Editing context menu options

The following options are available on the context menu when you select the Address field or Size field for editing:

Cut Copy and delete the selected text.

Copy Copy the selected text.

Paste Paste text that you previously cut or copied.

Delete Delete the selected text.

Undo Undo the last change.

Select All Select all the text.

11.8.2 See also

Tasks

- *Setting an execution breakpoint* on page 4-9
- *Setting a conditional breakpoint* on page 4-13
- *Settinga breakpoint on a specific thread* on page 4-16
- *Examining the target execution environment* on page 5-2
- *Examining the call stack* on page 5-4
- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *C/C++ editor* on page 11-12
- *Registers view* on page 11-36
- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Using Eclipse*:
  - *Perspectives and views* on page 3-19.

11.9 Expressions view

This view enables you to:

- add expressions that you use regularly or that you want to examine in more detail
- edit, and delete expressions
- freeze the selected view to prevent the values being updated by a running target.

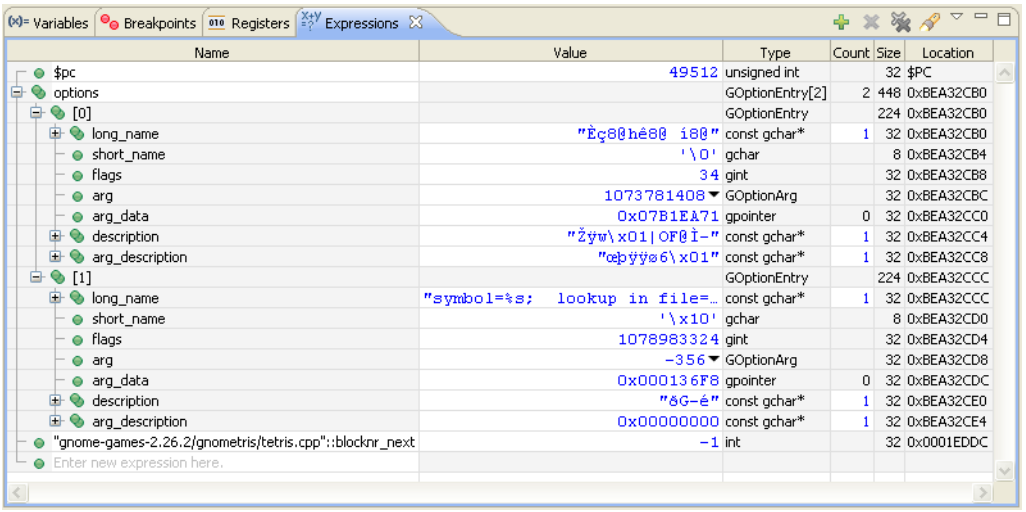

Figure 11-10 Expressions view

Note

If your expression contains side-effects when evaluating the expression, the results are unpredictable. Side-effects occur when the state of one or more inputs to the expression changes when the expression is evaluated.

For example, instead of `x++` or `x+=1` you must use `x+1`.

Right-click on the column headers to select the columns that you want displayed:

Name An expression that resolves to an address, such as `main+1024`.

Value The value of the expression. You can modify a value that has a white background. A yellow background indicates the value has changed.

If you freeze the view, then you cannot change a value.

Type The type associated with the value at the address identified by the expression.

Count The number of array or pointer elements. You can edit a pointer element count.

Size The size of memory in bits.

Location The address in hexadecimal identified by the expression, or the name of a register, if expression contains only a single register name.

All columns are displayed by default.

11.9.1 Toolbar and context menu options

The following options are available from the toolbar or context menu:

Add New Expression

Add a new expression to the expression list.

Remove Selected Expression

Remove the selected expression from the list.

Remove All Expressions

Remove all expressions from the list.

Search

Search for an expression.

Refresh

Refresh the view.

Freeze Data

Toggle the freezing of the data in the view. This also disables and enables the **Refresh** option.

New Expression View

Display a new instance of the Expressions view.

Cut Copy and remove the selected expression.

Copy Copy the selected expression.

To copy an expression for use in the Disassembly view or Memory view, first select the expression in the Name field.

Paste Paste expressions that you have previously cut or copied.

Delete Delete the selected expression.

Select All Select all expressions.

Show in memory view

Where enabled, displays the Memory view with the address set to either:

- the value of the selected expression, if the expression translates to an address, for example the address of an array, `&name`
- the location of the expression, for example the name of an array, `name`.

The memory size is set to the size of the variable, using the **`sizeof`** keyword.

Show in register view

If the expression corresponds to a register, then displays the Registers view with that register selected. This might be:

- an expression that consists only of a single register, for example `$pc`
- a variable that is currently held in a register, for example, the variable `t` might be held in register R5.

Send to Selection

Enables you to add register filters to an Expression view. Displays a sub menu that enables you to add to a specific Expressions view.

format list A list of formats you can use for the expression value.

11.9.2 See also

Tasks

- *Examining the target execution environment* on page 5-2
- *Examining the call stack* on page 5-4
- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *Memory view* on page 11-30
- *Registers view* on page 11-36
- *Variables view* on page 11-48
- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Using Eclipse*:
  - *Perspectives and views* on page 3-19.

11.10 History view

This view enables you to:

- See a full list of commands generated during the current debug session.
- Clear the contents of the view.
- Save the selected commands to a script file. You can also add the script file to your favorites list when you click **Save**. Favorites are displayed in the Scripts view.
- Enable or disable the automatic scrolling of messages in the History view.

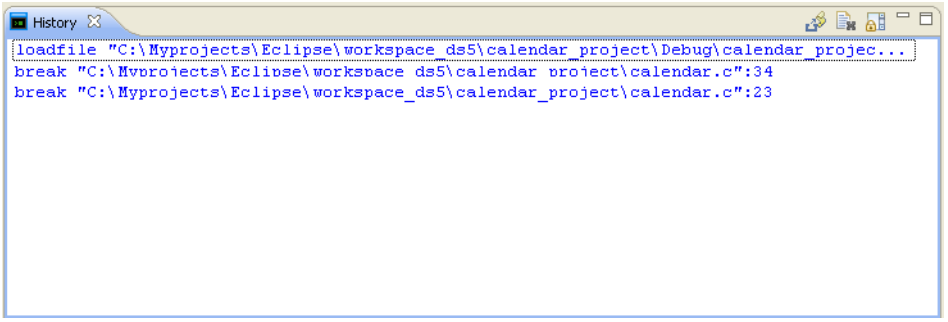

Figure 11-11 History view

Note

Default settings for this view are controlled by a DS-5 Debugger setting in the Preferences dialog box. For example, default locations for specific files. You can access these settings by selecting **Preferences...** from the **Window** menu.

11.10.1 Toolbar and context menu options

The following options are available from the toolbar or context menu:

Exports the selected lines as a script

Display the Save As dialog box to save the selected commands to a script file.

When you click **Save** on the Save As dialog box, you are given the option to add the script file to your favorites list. Click **OK** to add the script to your favorites list. Favorites are displayed in the Scripts view.

Clear Console

Clear the contents of the History view.

Toggles Scroll Lock

Enable or disable the automatic scrolling of messages in the History view.

Copy Copy the selected commands.

Select All Select all commands.

Save the selected lines as a script...

Display the Save As dialog box to save the selected commands to a script file.

When you click **Save** on the Save As dialog box, you are given the option to add the script file to your favorites list. Click **OK** to add the script to your favorites list. Favorites are displayed in the Scripts view.

Execute selected lines

Run the selected commands.

11.10.2 See also

Tasks

- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *Commands view* on page 11-15
- *Scripts view* on page 11-41
- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Debugger Command Reference*:
  - *DS-5 Debugger commands listed in alphabetical order* on page 2-18.
- *ARM® DS-5™ Using Eclipse*:
  - *Perspectives and views* on page 3-19.

11.11 Memory view

This view enables you to:

- Modify memory content.
- Specify the start address for the Memory view, either as an absolute address or as an expression, for example `$pc`. Previous entries are listed in the drop-down list. This list is cleared when you exit Eclipse.
- Specify the display size of the Memory view in bytes, either as an offset value from the start address, or as an address held in a register by dragging and dropping the register from the Registers view into the Memory view.
- Specify the format of the memory cell values. The default is hexadecimal.
- Set the width of the memory cells in the Memory view. The default is four bytes.
- Display the ASCII character equivalent of the memory values.
- Freeze the selected view to prevent the view being updated by a running target.

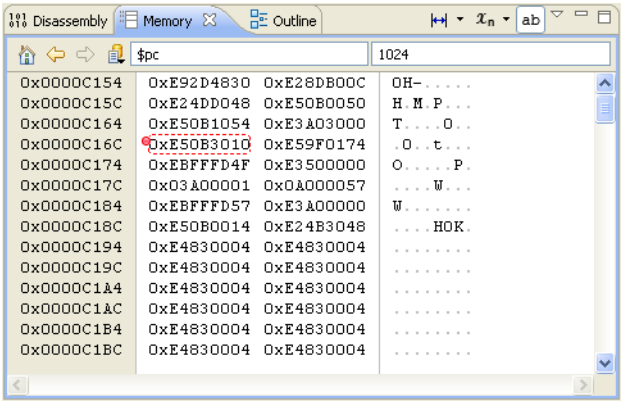

Figure 11-12 Memory view

The Memory view only provides the facility to modify how memory is displayed in this view. It is not possible to specify the use of byte, half-word, word or double read/write instructions to access memory from the Memory view. To control the memory access width you can use:

- the `memory` command to configure access widths for a region of memory, followed by the `x` command to read memory according to those access widths and display the contents
- the `memory set` command to write to memory with an explicit access width.

11.11.1 Toolbar and context menu options

The following options are available from the toolbar or context menu:

Refresh

Refresh the view.

Back, Forward

Navigate through the history list.

History Addresses and expressions you specify in the Address field are added to the drop down box, and persist until you clear the history list or exit Eclipse. If you want to keep an expression for later use, add it to the Expressions view.

Clear History, <Clear Data>

Clears the list of addresses and expressions in the History drop-down box.

Display Width

Click to cycle through the memory cell widths in the Memory view, or select a width from the drop-down menu. The default is four bytes.

Format Click to cycle through the memory cell formats, or select a format from the drop-down menu. The default is hexadecimal.

Showing characters - click to hide the character display, Not showing characters - click to show the character display

Toggle the display of ASCII character equivalents for the memory values.

Freeze Data

Toggle the freezing of the data in the view. This also disables or enables the Address and Size fields and the **Refresh** option.

Show Tooltips

Toggle the display of tooltips on memory cell values.

New Memory View

Display a new instance of the Memory view.

Address field

Enter the address where you want to start viewing the target memory. Alternatively, you can enter an expression that evaluates to an address. For example `$PC+256`.

Addresses and expressions you specify are added to the drop down list, and persist until you exit Eclipse. If you want to keep an expression for later use, add it to the Expressions view.

Context menu options are available for editing this field.

Size field The number of bytes to display.

Context menu options are available for editing this field.

Editing context menu options

The following options are available on the context menu when you select a memory cell value, the Address field, or the Size field for editing:

Cut Copy and delete the selected value.

Copy Copy the selected value.

Paste Paste a value that you have previously cut or copied into the selected memory cell or field.

Delete Delete the selected value.

Undo Undo the last change you made to the selected memory cell or field. This is disabled for the Address field.

Toggle Breakpoint

Set or remove a breakpoint at the selected address.

11.11.2 See also

Tasks

- *Examining the target execution environment* on page 5-2
- *Examining the call stack* on page 5-4
- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Using Eclipse*:
  - *Perspectives and views* on page 3-19.
- *ARM® DS-5™ Debugger Command Reference*:
  - *memory set* on page 2-111
  - *x* on page 2-201.

11.12 Modules view

This view is only populated when connected to a Linux target. It enables you to:

- see a tabular view of the shared libraries used by the application
- see a tabular view of loaded *Operating System* (OS) modules since connection
- load and unload debug information for a specific module or shared library.

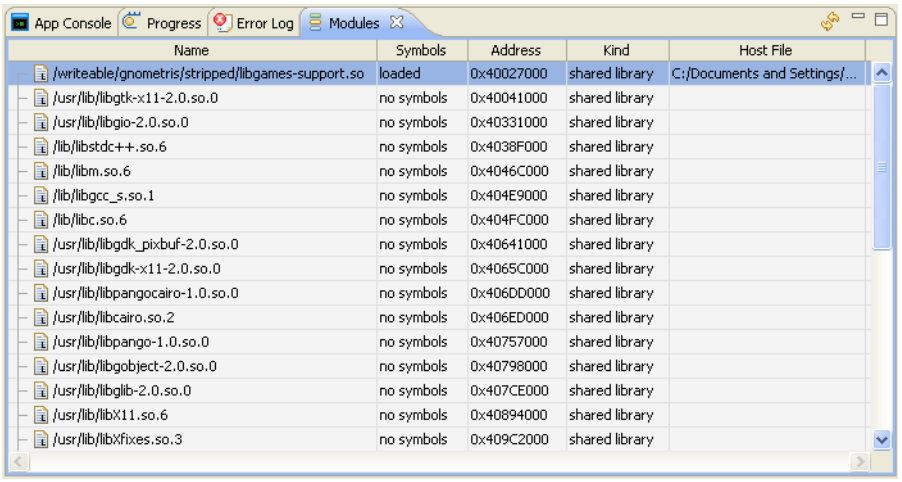

Figure 11-13 Modules view showing shared libraries

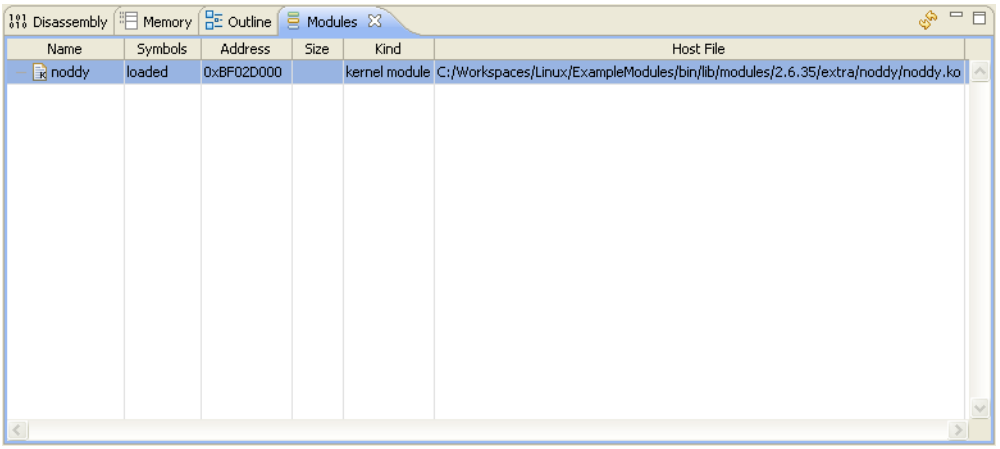

Figure 11-14 Modules view showing an operating system module

Note

A connection must be established and operating system support must be activated within the debugger before a loadable module can be detected. You can use the `set os` command to control operating system support in the debugger.

Right-click on the column headers to select the columns that you want displayed:

Name Displays the name and location of the component on the target.

Symbols Displays whether the symbols are currently loaded for each object.

Address Displays the load address of the object.

Size Displays the size of the object.

Kind Displays the component type. For example, shared library or OS module.

Host File Displays the name and location of the component on the host workstation.

The `Name`, `Symbols`, `Address`, `Kind`, and `Host File` columns are displayed by default.

The Modules view is not visible by default. To add this view:
1. Ensure that you are in the DS-5 Debug perspective.
2. Select **Window → Show View** to open the Show View dialog box.
3. Select **Modules** view.

11.12.1 Toolbar and context menu options

The following options are available from the toolbar or context menu:

Refresh

Refresh the view.

Copy Copy the selected data.

Select All Select all the displayed data.

Load Symbols

Loads debug information into the debugger from the source file displayed in the Host File column. This option is disabled if the host file is unknown. For example before the file is loaded.

Add Symbol File...

Opens a dialog box where you can select a file from the host workstation containing the debug information required by the debugger.

Discard Symbols

Discards debug information relating to the selected file.

Show in Memory

Displays the Memory view starting at the load address of the selected object.

Show in Disassembly

Displays the Disassembly view starting at the load address of the selected object.

11.12.2 See also

Tasks

- *Examining the target execution environment* on page 5-2
- *Examining the call stack* on page 5-4
- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Using Eclipse*:
  - *Perspectives and views* on page 3-19.
- *ARM® DS-5™ Debugger Command Reference*:
  - *add-symbol-file* on page 2-22

- *discard-symbol-file* on page 2-46
- *file, symbol-file* on page 2-55.

11.13 Registers view

This view enables you to:

- See the contents of target registers.
- Change the values for registers that have write access. When a register value changes, the register value background changes to yellow.
- Change the display format of register values. The *Program Status Registers* (PSRs) also enable you to set the format using individual bits.
- Freeze the selected view to prevent the values being updated by a running target.
- Drag and drop an address held in a register, such as `R3`, from the Registers view either into the Memory view to see the memory at that address, or into the Disassembly view to disassemble from that address.

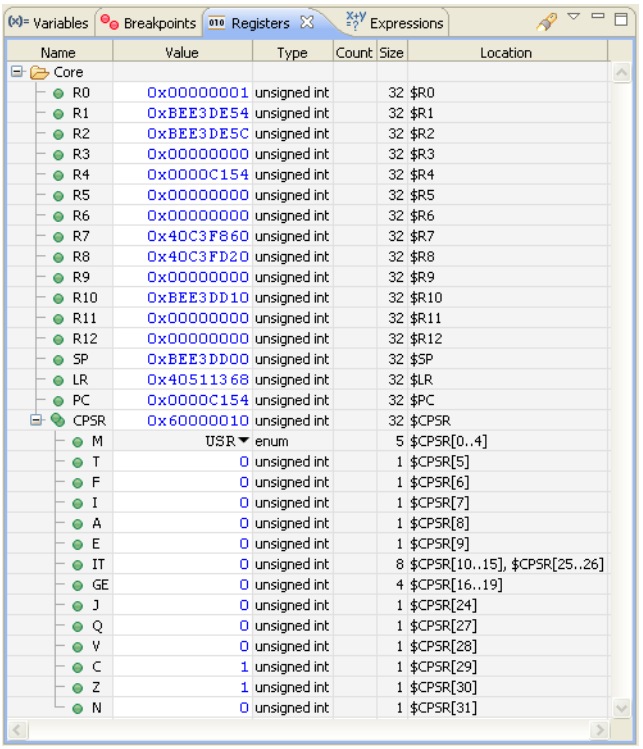

Figure 11-15 Registers view

Right-click on the column headers to select the columns that you want displayed:

Name The name of the register.

Use `$register_name` to reference a register. To refer to a register that has bitfields, such as a PSR, specify `$register_name.bitfield_name`. For example, to print the value of the `M` bitfield of `$CPSR`, enter the following command in the Commands view:

```
print $CPSR.M
$1 = USR
```

Value The value of the register. A shaded background indicates the value has changed.

If you freeze the view, then you cannot change a register value.

Type The type of the register value.

Count The number of array or pointer elements.

Size The size of the register in bits.

Location The name of the register or the bitmap of the bitfield of a PSR. For example, bitfield M of the CSPR is displayed as `$CPSR[0..4]`.

The `Name`, `Value`, and `Size` columns are displayed by default.

11.13.1 Toolbar and context menu options

The following options are available from the toolbar or context menu:

Refresh

Refresh the view.

Search

Search for a register.

Freeze Data

Toggle the freezing of the data in the view. This also disables or enables the **Refresh** option.

New Registers View

Create a new instance of the Registers view.

Copy Copy the selected registers. To copy the bitfields of a PSR, you must first expand the PSR.

This is useful if you want to copy the selected registers to a text editor and compare the values when execution stops at another location.

Select All Select all registers currently expanded in the view.

Show Memory Pointed to By *register_name*

Where enabled, displays the Memory view starting at the address held in the register.

Show Disassemby Pointed to By *register_name*

Where enabled, displays the Disassembly view starting at the address held in the register.

Send to Selection

Enables you to add register filters to an Expression view. Displays a sub menu that enables you to add to a specific Expressions view.

format list A list of formats you can use for the register values. The default is **Hexadecimal**.

Editing context menu options

The following options are available on the context menu when you select a register value for editing:

Cut Copy and delete the selected value.

Copy Copy the selected value.

Paste Paste a value that you have previously cut or copied into the selected register value.

Delete Delete the selected value.

Undo Undo the last change you made to the selected value.

11.13.2 See also

Tasks

- *Examining the target execution environment* on page 5-2
- *Examining the call stack* on page 5-4
- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Using Eclipse*:
  - *Perspectives and views* on page 3-19.

11.14 Screen view

This view enables you to:

- See the contents of the screen buffer on the target. This view only updates when the target stops.
- Set the screen buffer parameters appropriate for the target.
- Freeze the selected view to prevent the screen display being updated by the running target when it next stops.

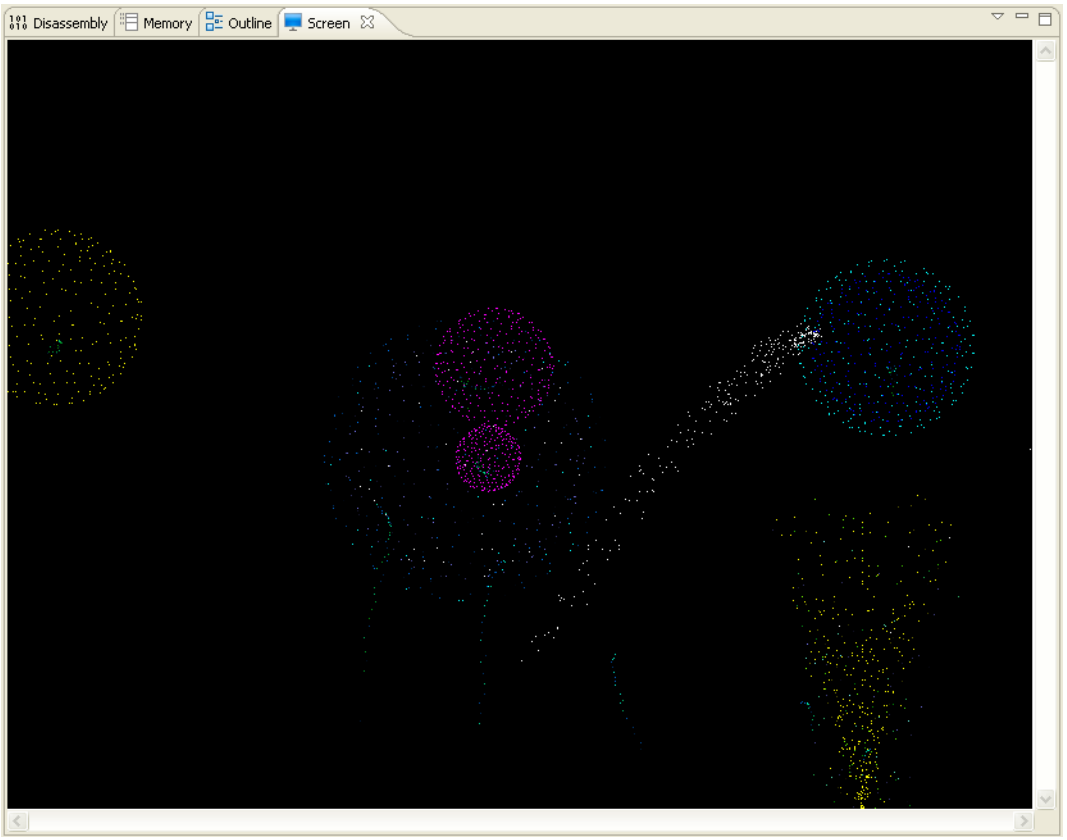

Figure 11-16 Screen view

11.14.1 Toolbar options

The following toolbar options are available:

Refresh

Refresh the view.

Set Screen Buffer Parameters

Display the Screen Buffer Parameters dialog box. The dialog box contains the following parameters:

Base Address

Set the base address of the screen buffer.

Screen Width

Set the width of the screen in pixels.

Screen Height

Set the height of the screen in pixels.

Scan Line Alignment

Set the byte alignment required for each scan line.

Pixel Type

Select the pixel type.

Pixel Byte Order

Select the byte order of the pixels within the data.

Click **Apply to save the settings and close the dialog box**.

Click **Cancel** to close the dialog box without saving.

Freeze Data

Toggle the freezing of the data in the view. This also disables or enables the **Refresh** option.

New Screen Buffer View

Create a new instance of the Screen view.

The Screen view is not visible by default. To add this view:

1. Ensure that you are in the DS-5 Debug perspective.
2. Select **Window → Show View** to open the Show View dialog box.
3. Select **Screen** view.

11.14.2 See also

Tasks

- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Using Eclipse*:
  - *Perspectives and views* on page 3-19.

11.15 Scripts view

This view contains your favorite scripts. You can run, edit, or remove one or more of your favorite scripts. Scripts can be added to this view when you save commands in the History view.

Multiple selections are executed in the order listed in the view. To change the order, remove the scripts from the view and import them in the required order.

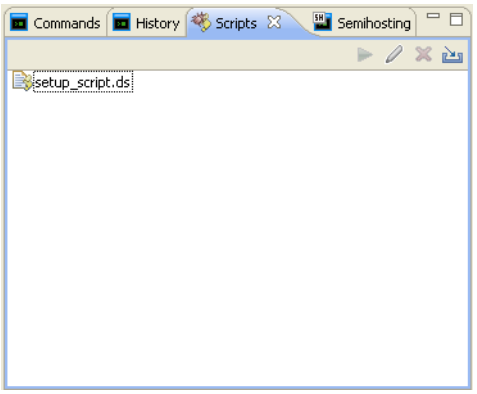

Figure 11-17 Scripts view

Note

Default settings for this view are controlled by a DS-5 Debugger setting in the Preferences dialog box. For example, default locations for script files. You can access these settings by selecting **Preferences...** from the **Window** menu.

11.15.1 Toolbar and context menu options

The following options are available from the toolbar or context menu:

Creates a new script

Create a new empty script. To specify the script contents after it is created, select the script and click **Edit Selected Script**.

Execute Selected Scripts

Run the selected scripts. If you select multiple scripts, the debugger runs them in the order listed in the Scripts view.

Edit Selected Scripts

Edit the selected scripts. The scripts are opened in the C/C++ editor.

Delete Selected Scripts

Delete the selected scripts from the favorites list. You are also prompted to delete the script from the file system.

Import Script...

Import a script file and add it to the favorites list.

Cut Copy and remove the selected script filename. You are also prompted to delete the script from file system.

Copy Copy the selected script.

Paste Paste a script filename that you have previously cut or copied.

If you deleted the file from the file system as part of a Cut operation, the file contents are not restored. You must edit the file to add new commands.

If you did not delete the file as part of a Cut operation, the debugger links the filename to the file in the file system.

Delete Delete the selected script from the favorites list. You are also prompted to delete the script from file system.

Select All Select all script files.

11.15.2 See also

Tasks

- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *C/C++ editor* on page 11-12
- *Commands view* on page 11-15
- *History view* on page 11-28
- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Using Eclipse*:
  - *Perspectives and views* on page 3-19.

11.16 Target view

This view enables you to examine the debug capabilities supported by the target, such as:

- breakpoint types supported
- reset types supported
- memory access types supported.

All capabilities are read-only.

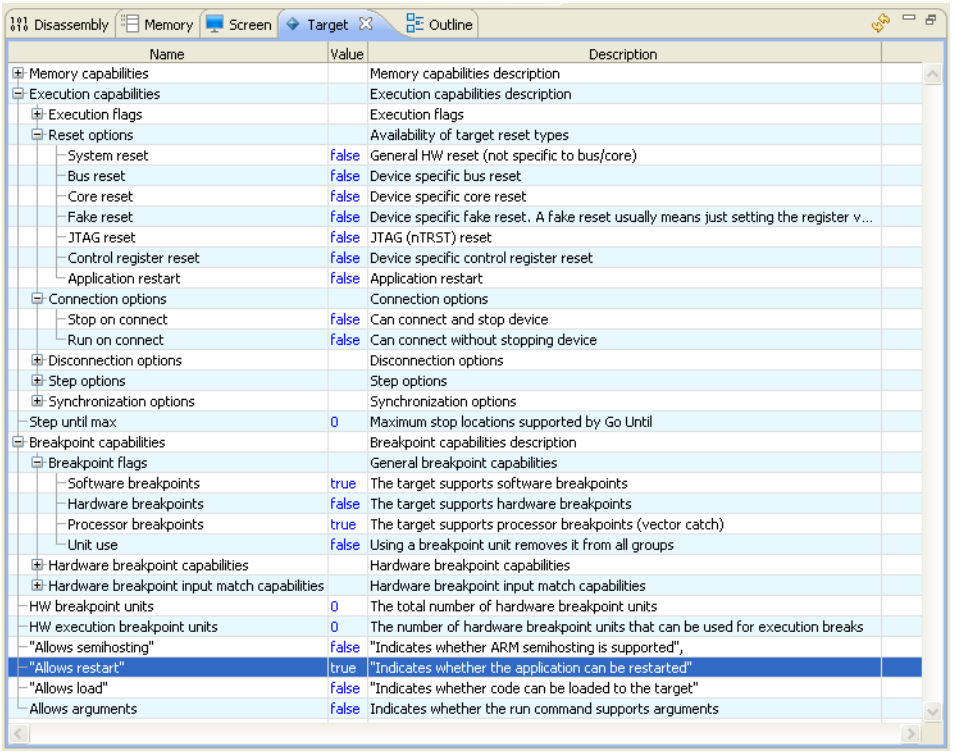

Figure 11-18 Target view

Right-click on the column headers to select the columns that you want displayed:

Name The name of the target capability.

Value The value of the target capability.

Key The name of the target capability. This is used by some commands in the Commands view.

Description A brief description of the target capability.

The `Name`, `Value`, and `Description` columns are displayed by default.

The Target view is not visible by default. To add this view:

1. Ensure that you are in the DS-5 Debug perspective.
2. Select **Window → Show View** to open the Show View dialog box.
3. Select **Target** view.

11.16.1 Toolbar and context menu options

The following options are available from the toolbar or context menu:

Refresh the Target Capabilities

Refresh the view.

Copy Copy the selected capabilities. To copy the capabilities in a group such as `Memory capabilities`, you must first expand that group.

This is useful if you want to copy the selected capabilities to a text editor and save them for future reference.

Select All Select all capabilities currently expanded in the view.

11.16.2 See also

Tasks

- *Examining the target execution environment* on page 5-2
- *Examining the call stack* on page 5-4
- *ARM DS-5 Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *Breakpoints view* on page 11-8
- *Memory view* on page 11-30
- *App Console view* on page 11-3
- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Using Eclipse*:
  - *Perspectives and views* on page 3-19.

11.17 Trace view

When the trace has been captured the debugger extracts the information from the trace stream and decompresses it to provide a full disassembly, with symbols, of the executed code.

This view shows a graphical navigation chart that displays function executions with a navigational timeline. In addition, the disassembly trace shows function calls with associated addresses and if selected, instructions. Clicking on a specific time in the chart synchronizes the disassembly view.

In the left-hand column of the chart, percentages are shown for each function of the total trace. For example, if a total of 1000 instructions are executed and 300 of these instructions are associated with `myFunction()` then this function is displayed with 30%.

In the navigational timeline, the color coding is a “heat” map showing the executed instructions and the amount of instructions each function executes in each timeline. The darker red color showing more instructions and the lighter yellow color showing less instructions. At a scale of 1:1 however, the color scheme changes to display memory access instructions as a darker red color, branch instructions as a medium orange color, and all the other instructions as a lighter green color.

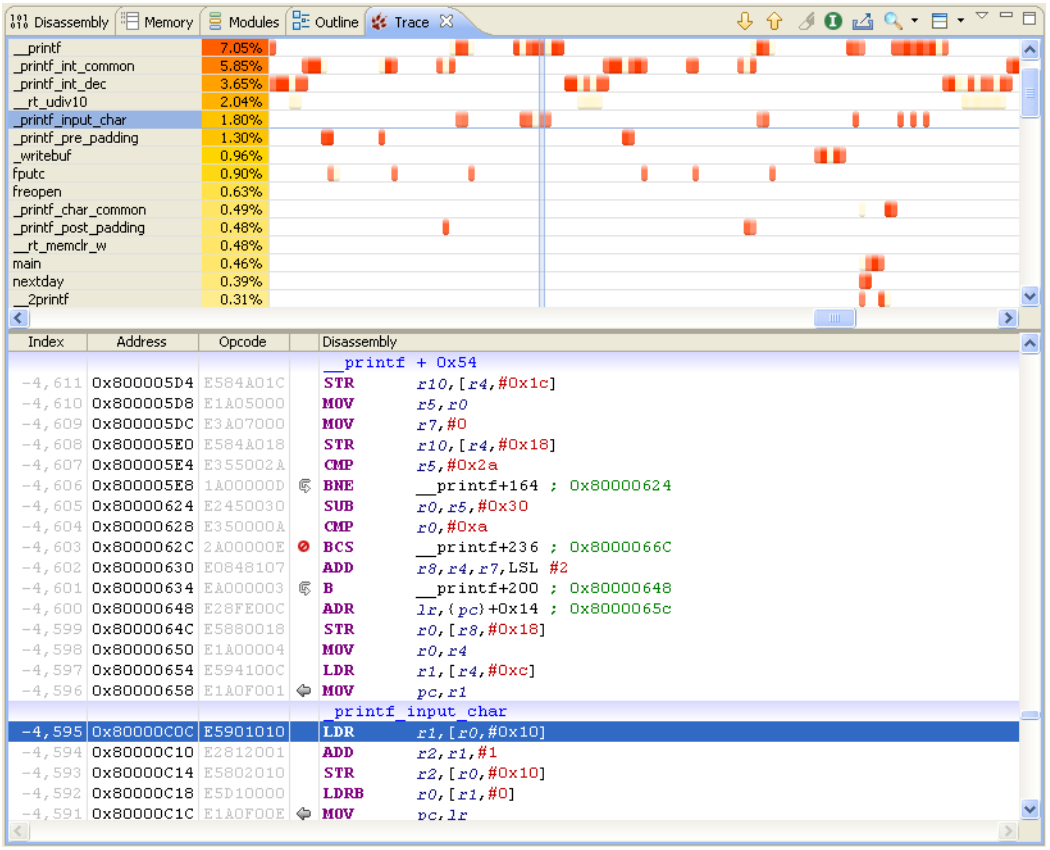

Figure 11-19 Trace view

The Trace view might not be visible by default. To add this view:

1. Ensure that you are in the DS-5 Debug perspective.
2. Select **Window** → **Show View** to open the Show View dialog box.
3. Select **Trace** view.

11.17.1 Toolbar and context menu options

The following options are available from the toolbar or context menu:

Refresh

Refresh the view.

Show Next Match

Move the focus of the navigation chart and disassembly trace to the next matching occurence for the selected function or instruction.

Show Previous Match

Move the focus of the navigation chart and disassembly trace to the previous matching occurence for the selected function or instruction.

Don't mark other occurences - click to start marking, Mark other occurences - click to stop marking

When function trace is selected, mark all occurrences of the selected function with a shaded highlight. This is disabled when instruction trace is selected.

Showing instruction trace - click to switch to functions, Showing function trace - click to switch to instructions

Toggle the disassembly trace between instruction trace or function trace.

Export Trace Report

Display the Export Trace Report dialog box to save the trace data to a file.

Switch between navigation resolutions

Change the timeline resolution in the navigation chart.

Switch between alternate views

Change the view to display the navigation chart, disassembly trace or both.

Freeze Data

Toggle the freezing of the data in the view.

New Trace View

Displays a new instance of the Trace view.

Focus Here At the top of the list, displays the function being executed in the selected time slot. The remaining functions are listed in the order that they are executed after the selected point in time. Any functions that do not appear after that point in time are placed at the bottom and ordered by total time.

Order By Total Time

Displays the functions ordered by the total time spent within the function. This is the default ordering.

View Menu The following **View Menu** options are available:

Copy Copy the selected instruction trace.

Set Maximum Instruction Depth

Displays a dialog box where you can enter the maximum number of instructions to display in the disassembly trace.

Index From Start

Numbers each decoded instruction in the buffer from the start.

Index From Middle

Numbers each decoded instruction in the buffer from the middle.

Index From End

Numbers each decoded instruction in the buffer from the end.

Show in Disassembly

Displays the Disassembly view starting at the address of the selected instruction.

11.17.2 See also

Tasks

- *Examining the target execution environment* on page 5-2
- *Examining the call stack* on page 5-4
- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Concepts

- *About trace support in DS-5* on page 5-6

Reference

- *Export trace report dialog box* on page 11-51
- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Using Eclipse*:
  - *Perspectives and views* on page 3-19.

11.18 Variables view

This view enables you to:

- see the contents of variables that are currently in scope
- change the values for variables that are currently in scope
- freeze the selected view to prevent the values being updated by a running target.

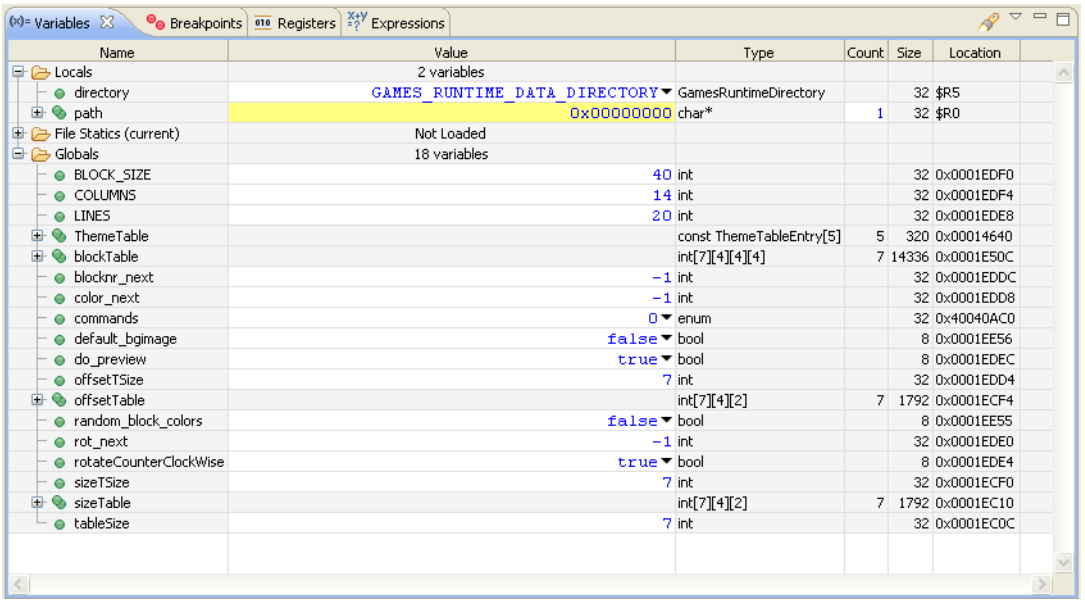

Figure 11-20 Variables view

Right-click on the column headers to select the columns that you want displayed:

Name The name of the variable.

Value The value of the variable.

Read-only values are displayed with a grey background. Any other color means that you can edit the value.

A value that you can edit is initially shown with a white background. If the value changes, either by performing a debug action such as stepping or by you editing the value directly, the background changes to yellow.

If you freeze the view, then you cannot change a value.

Type The type of the variable.

Count The number of array or pointer elements.

Size The size of the variable in bits.

Location The address of the variable.

All columns are displayed by default.

11.18.1 Toolbar and context menu options

The following options are available from the toolbar or context menu:

Refresh

Refresh the view.

Freeze Data

Toggle the freezing of the data in the view. This also disables or enables the **Refresh Variable View** option. Also, you cannot modify the value of a variable if the data is frozen.

If you freeze the data before you expand an item, such as an array, for the first time, the view might show `Pending...` items. Unfreeze the data to see the items.

Search

Search for a variable.

New Variables View

Displays a new instance of the Variables view.

Copy Copy the selected variables. To copy the contents of an item such as a structure or an array, you must first expand that item.

This is useful if you want to copy the selected variables to a text editor and compare the values when execution stops at another location.

Select All Select all capabilities currently expanded in the view.

Show in Memory

Where enabled, displays the Memory view with the address set to either:

- the value of the selected variable, if the variable translates to an address, for example the address of an array, `&name`
- the location of the variable, for example the name of an array, `name`.

The memory size is set to the size of the variable, using the **`sizeof`** keyword.

Show in Registers

If the selected variable is currently held in a register, then displays the Registers view with that register selected. For example, the variable `t` might be held in register R5.

Show Derefence in Memory

If the selected variable is a pointer, then displays the Memory view with the address where the variable is pointing to in memory, selected.

Send to Selection

Enables you to add variable filters to an Expression view. Displays a sub menu that enables you to add to a specific Expressions view.

format list A list of formats you can use for the variable value. The default is **Unsigned Decimal**.

Editing context menu options

The following options are available on the context menu when you select a variable value for editing:

Cut Copy and delete the selected value.

Copy Copy the selected value.

Paste Paste a value that you have previously cut or copied into the selected variable value.

Delete Delete the selected value.

Undo Undo the last change you made to the selected value.

Right to left reading order

Set the reading order for the selected variable value to be left or right justified.

Show unicode control characters

Show any unicode control characters in the selected variable value.

Insert unicode control character

Select the unicode control character to insert into the selected variable value.

11.18.2 See also

Tasks

- *Examining the target execution environment* on page 5-2
- *Examining the call stack* on page 5-4
- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *Expressions view* on page 11-25
- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Using Eclipse*:
  - *Perspectives and views* on page 3-19.

11.19 Export trace report dialog box

This dialog box enables you to generate a trace report.

Select source for trace report

Select the required trace data:

Use trace view as report source

Select the instructions that are currently visible in the trace view.

Use trace buffer as report source

Select the raw trace data that is currently contained in the trace buffer.

Note

When specifying a range, ensure that the range is large enough otherwise you might not get any trace output. This is due to the raw trace packing format used in the buffer.

Report Format

Configure the report:

Output Format

Select the output format.

Include column headers

Enables you to add column headers in the first line of the report.

Select columns to export

Enables you to filter the trace data in the report.

Save as Enter the report location and filename.

Browse Select the report location in the file system.

11.19.1 See also

Tasks

- *Examining the target execution environment* on page 5-2
- *Examining the call stack* on page 5-4
- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Concepts

- *About trace support in DS-5* on page 5-6

Reference

- *Trace view* on page 11-45
- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Using Eclipse*:
  - *Perspectives and views* on page 3-19.

11.20 Breakpoint properties dialog box

This dialog box enables you to:

- display the properties of a selected breakpoint
- set a conditional expression for a specific breakpoint
- set an ignore counter for a specific breakpoint
- specify a script file to run when the selected breakpoint is hit
- enable the debugger to automatically continue running on completion of all the breakpoint actions
- assign a breakpoint action to a specific thread, if available.

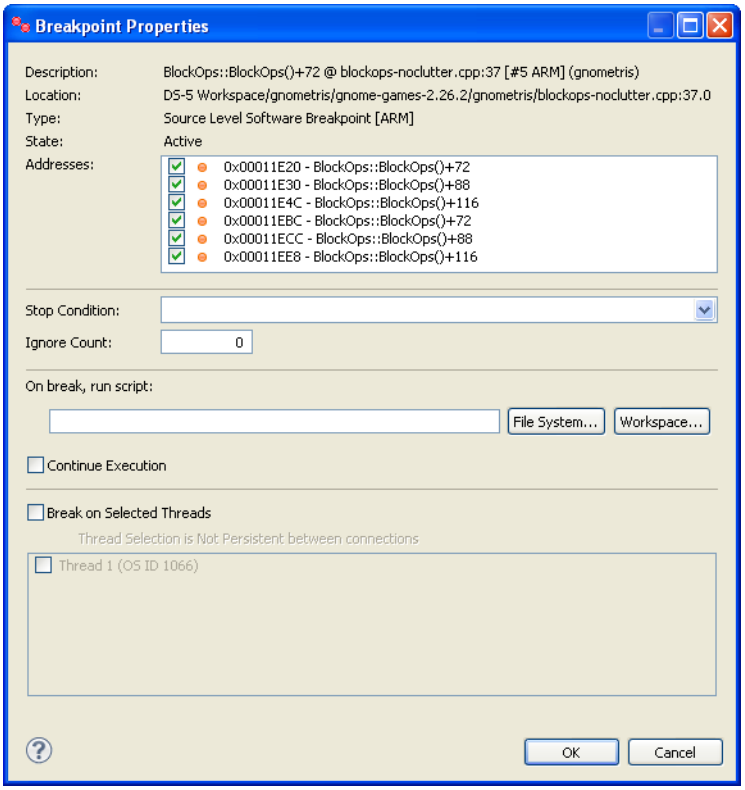

Figure 11-21 Breakpoint properties dialog box

11.20.1 Breakpoint information

The breakpoint information shows the basic properties of a breakpoint:

Description A description of the breakpoint as displayed in the Breakpoints view. This comprises:

- The name of the function in which the breakpoint is set and the number of bytes from the start of the function. For example, `accumulate()+52` shows that the breakpoint is 52 bytes from the start of the `accumulate()` function.
- If the source file is available, the file name and line number in the file where the breakpoint is set, for example `threads.c:115`.
- A breakpoint ID number, *#N*. In some cases, such as in a **`for`** loop, a breakpoint might comprise a number of sub-breakpoints. These are identified as *N.n*, where *N* is the number of the parent. The description of a sub-breakpoint in this dialog box is shown as

```
main()+132sub-breakpoint ofmain()+132 @ threads.c:56 [#14 ARM]
(threads)
```

- The type of instruction at the address of the breakpoint, `ARM` or `Thumb`.
- An `ignore` count, if set. The display format is:

  `ignore = num/count`

  num equals `count` initially, and decrements on each pass until it reaches zero.

  count is the value you have specified for the `ignore` count.
- A `hits` count that increments each time the breakpoint is hit. This is not displayed until the first hit. If you set an `ignore` count, `hits` count does not start incrementing until the `ignore` count reaches zero.
- The stop condition you have specified, for example `(i==3)`.
- The name of the image.

For example:

```
accumulate()+52 @ threads.c:115 [#1 ARM, ignore = 3/3, 3 hits, (i==3)]
(threads)
```

Location The location of the source file containing the address where the breakpoint is set, for example:

```
C:/Myprojects/Eclipse/workspace_ds5/threads/threads.c:115.0
```

If no source file is available, then `Unknown` is displayed.

Type This shows:

- whether or not the source file is available for the code at the breakpoint address, `Source Level` if available or `Address Level` if not available
- if the breakpoint is on code in a shared object, `Auto` indicates that the breakpoint is automatically set when that shared object is loaded
- if the breakpoint is `Active`, the type of breakpoint, either `Software Breakpoint` or `Hardware Breakpoint`
- the type of instruction at the address of the breakpoint, `ARM` or `Thumb`.

For example:

```
Source Level Software Breakpoint [ARM]
```

State Indicates one of the following:

`Active` The image or shared object containing the address of the breakpoint is loaded, and the breakpoint is set.

`No Connection`
The breakpoint is in an application on a target that is not connected.

`Pending` The image or shared object containing the address of the breakpoint has not yet been loaded. The breakpoint becomes active when the image or shared object is loaded.

Address A dialog box that displays one or more breakpoint or sub-breakpoint addresses with check boxes where you can enable or disable them.

Temporary Shows `true` if this is a temporary breakpoint. Otherwise, shows `false`.

11.20.2 Breakpoint options

The following options are available for you to set:

Stop Condition

Specify a C-style conditional expression for the selected breakpoint. For example, to activate the breakpoint when the value of `x` equals `10`, specify `x==10`.

Ignore Count

Specifies the number of times the selected breakpoint is ignored before it is activated.

The debugger decrements the counter on each pass until it reaches zero, for example:

```
main()+140 @ threads.c:51 [#1 ARM, ignore = 2/3] (threads)
```

When the value reaches zero the breakpoint activates. Each subsequent pass causes the breakpoint to activate.

Select the **Reset Ignore Count** option from the context menu to reset the counter to the value you have set and delay activation again.

On break, run script

Specify a script file to run when the selected breakpoint is activated.

Note

Take care with the commands you use in a script that is attached to a breakpoint. For example, if you use the `quit` command in a script, the debugger disconnects from the target when the breakpoint is hit.

Continue Execution

Select this option if you want to continue running the target after the breakpoint is activated.

Break on Selected Threads

Select this option if you want to set a breakpoint for a specific thread. This option is disabled if no threads are available.

When a breakpoint activates, the debugger does the following:

- displays a message in the Commands view, for example:

```
Execution stopped at breakpoint 1: 0x00008850
In thread 1 (OS thread id 1078)
0x00008850   51,0   thread_app_data[t].thread = t;
```

- increments a hit count for the breakpoint, for example:

```
main()+140 @ threads.c:51 [#1 ARM, ignore = 0/3, 2 hits] (threads)
```

11.20.3 See also

Tasks

- *Setting an execution breakpoint* on page 4-9
- *Setting a conditional breakpoint* on page 4-13
- *Pending breakpoints and watchpoints* on page 4-18
- *Examining the target execution environment* on page 5-2
- *Examining the call stack* on page 5-4

- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *Breakpoints view* on page 11-8
- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Debugger Command Reference*:
  - *clear* on page 2-34
  - *delete breakpoints* on page 2-40
  - *disable breakpoints* on page 2-43
  - *enable breakpoints* on page 2-51
  - *ignore* on page 2-67
  - *set breakpoint* on page 2-133.

11.21 Watchpoint properties dialog box

This dialog box enables you to:

- display the properties of a selected watchpoint
- change the watchpoint type.

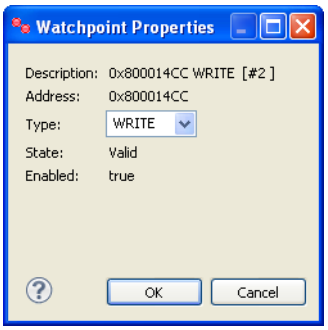

Figure 11-22 Watchpoint properties dialog box

The following types are available:

READ The debugger stops the target when the memory is read

WRITE The debugger stops the target when the memory is written

ACCESS The debugger stops the target when the memory is read or written.

11.21.1 See also

Tasks

- *Setting a data watchpoint* on page 4-11
- *Examining the target execution environment* on page 5-2
- *Examining the call stack* on page 5-4
- *ARM DS-5 Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *Breakpoints view* on page 11-8
- *DS-5 Debugger menu and toolbar icons* on page 11-75
- *ARM® DS-5™ Debugger Command Reference*:
  - *awatch* on page 2-26
  - *delete breakpoints* on page 2-40
  - *disable breakpoints* on page 2-43
  - *enable breakpoints* on page 2-51
  - *ignore* on page 2-67
  - *rwatch* on page 2-126
  - *watch* on page 2-197.

11.22 Manage Signals dialog box

This dialog box enables you to control the handler (vector catch) settings for one or more signals or processor exceptions. When a signal or processor exception occurs you can choose to stop execution, print a message, or both. **Stop** and **Print** are selected for all signals by default.

Note

When connected to an application running on a remote target using gdbserver, the debugger handles Unix signals but on bare-metal targets with no operating system it handles processor exceptions.

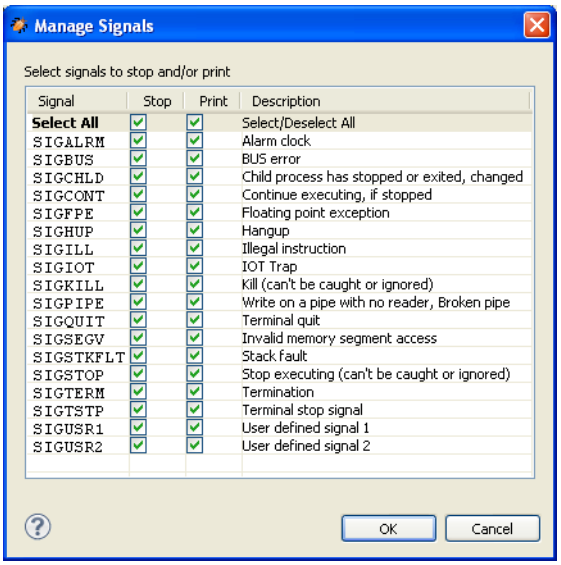

Figure 11-23 Managing signal handler settings

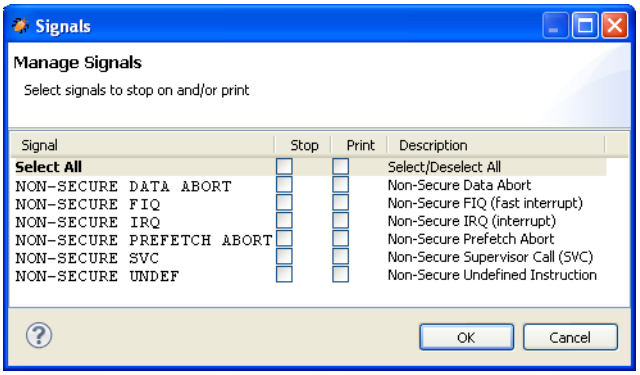

Figure 11-24 Manage exception handler settings

11.22.1 See also

Tasks

- *Handling Unix signals* on page 4-22
- *Handling processor exceptions* on page 4-24
- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *Target view* on page 11-43
- *DS-5 Debugger menu and toolbar icons* on page 11-75.

11.23 Debug Configurations - Connection tab

The **Connection** tab in the Debug Configurations dialog box enables you to configure DS-5 Debugger target connections. Each configuration you create is associated with a single target processor.

If the development platform has multiple processors, then you must create a separate configuration for each processor. Be aware that when connecting to multiple targets you cannot perform synchronization or cross-triggering operations.

Note

Options in the **Connection** tab are dependent on the type of platform that you select.

Select target

These options enable you to select the target:

Platform Select the target platform.

Project type

Select the project type:

Linux Application Debug

Select to debug a Linux application.

Linux Kernel and/or Device Driver Debug

Select to debug a Linux kernel or device driver.

Bare Metal Debug

Select to debug an application running on a bare-metal target.

Debug operation

These options enable you to specify the target environment. The options available depend on the selected platform and project type:

Download and debug application

Select this option when the application image and libraries do not exist on the target. When you connect to the target, the debugger loads the application onto the target.

Start gdbserver and debug target resident application

Select this option when the application already exists on the target.

Connect to already running gdbserver

Select this option when the application already exists on the target and `gdbserver` is already running.

Debug and ETB Trace via DSTREAM/RVI

Select to debug and trace an application running on a bare-metal target with *Embedded Trace Buffer* (ETB) support.

Debug and Trace via DSTREAM

Select to debug and trace an application running on a bare-metal target with *Trace Port Interface Unit* (TPIU) support.

Debug and Kernel-only ETB Trace via DSTREAM/RVI

Select to debug and trace a kernel running on a bare-metal target with ETB support.

Debug via DSTREAM/RVI

Select to debug an application running on a bare-metal target.

Connections

These options enable you to configure the connection between the debugger and the target:

RSE configuration

A list of *Remote System Explorer* (RSE) configurations that you have previously set up. Select the required RSE configuration that you want to use for this debug configuration.

gdbserver (TCP)

Specify the target IP address or name and the associated port number for the connection between the debugger and `gdbserver`.

The following options might also be available, depending on the debug operation you selected:

- Select the **Use Extended Mode** checkbox if you want to restart an application under debug. Be aware that this might not be fully implemented by `gdbserver` on all targets.
- Select the **Terminate gdbserver on disconnect** checkbox to terminate `gdbserver` when you disconnect from the target.
- Select the **Use RSE Host** checkbox to connect to `gdbserver` using the RSE configured host.

gdbserver (serial)

Specify the local serial port and connection speed for the serial connection between the debugger and `gdbserver`.

For the RTSM connection, details for `gdbserver` are obtained automatically from the target.

Select the **Use Extended Mode** checkbox if you want to restart an application under debug. Be aware that this might not be fully implemented by `gdbserver` on all targets.

Bare Metal Debug

Specify the target IP address or name of the debug hardware agent. You can also click on **Browse...** to display all the available debug hardware agents on your local subnet or USB connections.

Model parameters

These options are only enabled for RTSM platforms.

You can configure a *Virtual File System* (VFS) that enables a model to run an application and related shared library files from a directory on the local host. Alternatively you can disable VFS and manually transfer the files to a directory on the model.

Enable Virtual File System support

Enable or disable the use of VFS.

Host mount point

Specify the location of the file system on the local host:

- enter the location in the field provided
- click on **File System...** to locate the directory in an external location from the workspace

- click on **Workspace...** to locate the directory within the workspace.

For example, you can select the workspace root directory.

Remote target mount point

Displays the default location of the file system on the model. The default is the `/writeable` directory.

Apply Save the current configuration. This does not connect to the target.

Revert Undo any changes and revert to the last saved configuration.

Debug Connect to the target and close the Debug Configurations dialog box.

Close Close the Debug Configurations dialog box.

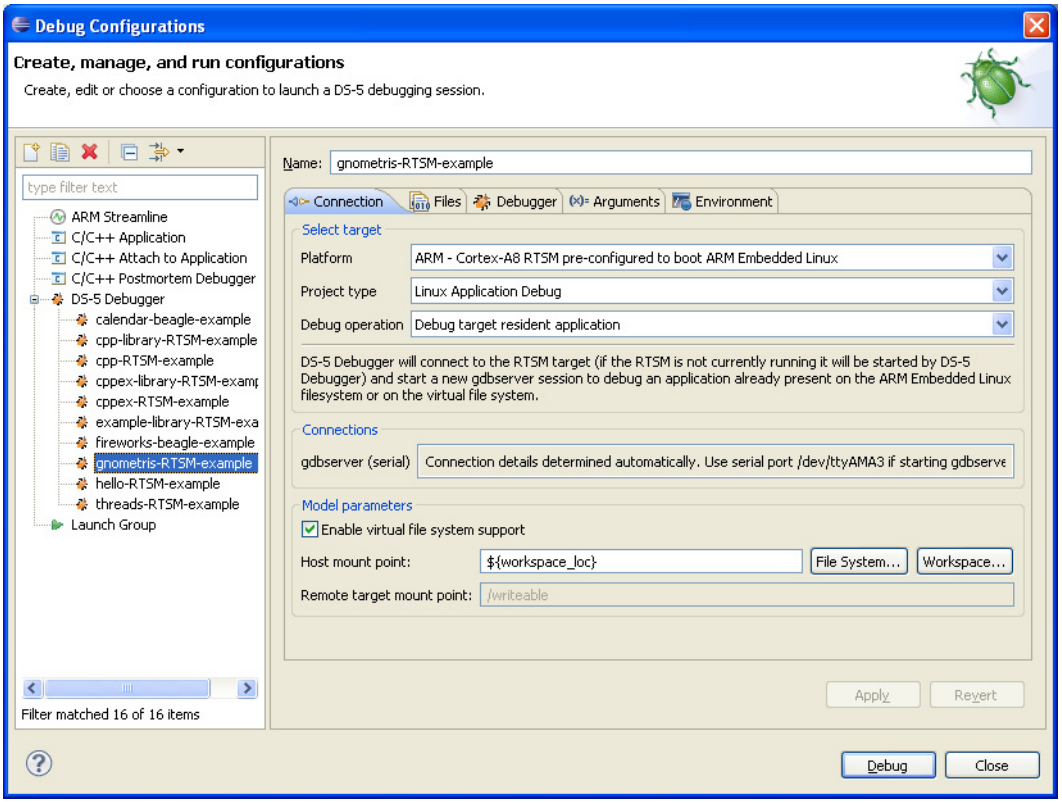

Figure 11-25 Connection configuration for a model using VFS

11.23.1 See also

Tasks

- *Configuring a connection to an RTSM model* on page 3-3
- *Configuring a connection to a Linux target using gdbserver* on page 3-5
- *ARM® DS-5™ Getting Started with DS-5*:
  - *Loading the Gnometris application on a Real-Time System Model* on page 3-6
  - *Loading the Gnometris application on to an ARM Linux target* on page 3-7.
- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *Debug Configurations - Files tab* on page 11-63
- *Debug Configurations - Debugger tab* on page 11-67
- *Debug Configurations - Arguments tab* on page 11-71
- *Debug Configurations - Environment tab* on page 11-73
- *ARM DSTREAM Setting up the Hardware*, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0481-.

11.24 Debug Configurations - Files tab

The **Files** tab in the Debug Configurations dialog box enables you to select debug versions of the application file and libraries on the host that you want the debugger to use. You can also specify the target file system folder to which files can be transferred if required.

Note

Options in the **Files** tab are dependent on the type of platform and debug operation that you select.

Files These options enable you to configure the target file system and select files on the host that you want to download to the target or use by the debugger. The Files tab options available for each **Debug operation** are:

Table 11-1 Files tab options available for each Debug operation

	Download and debug application	Debug target resident application	Connect to already running gdbserver	Debug via DSTREAM/RVI	Debug and ETB Trace via DSTREAM/RVI
Application on host to download	Yes	-	-	Yes	Yes
Application on target	-	Yes	-	-	-
Target download directory	Yes	-	-	-	-
Target working directory	Yes	Yes	-	-	-
Load symbols from file	Yes	Yes	Yes	Yes	Yes
Other file on host to download	Yes	-	-	-	-

Apply Save the current configuration. This does not connect to the target.

Revert Undo any changes and revert to the last saved configuration.

Debug Connect to the target and close the Debug Configurations dialog box.

Close Close the Debug Configurations dialog box.

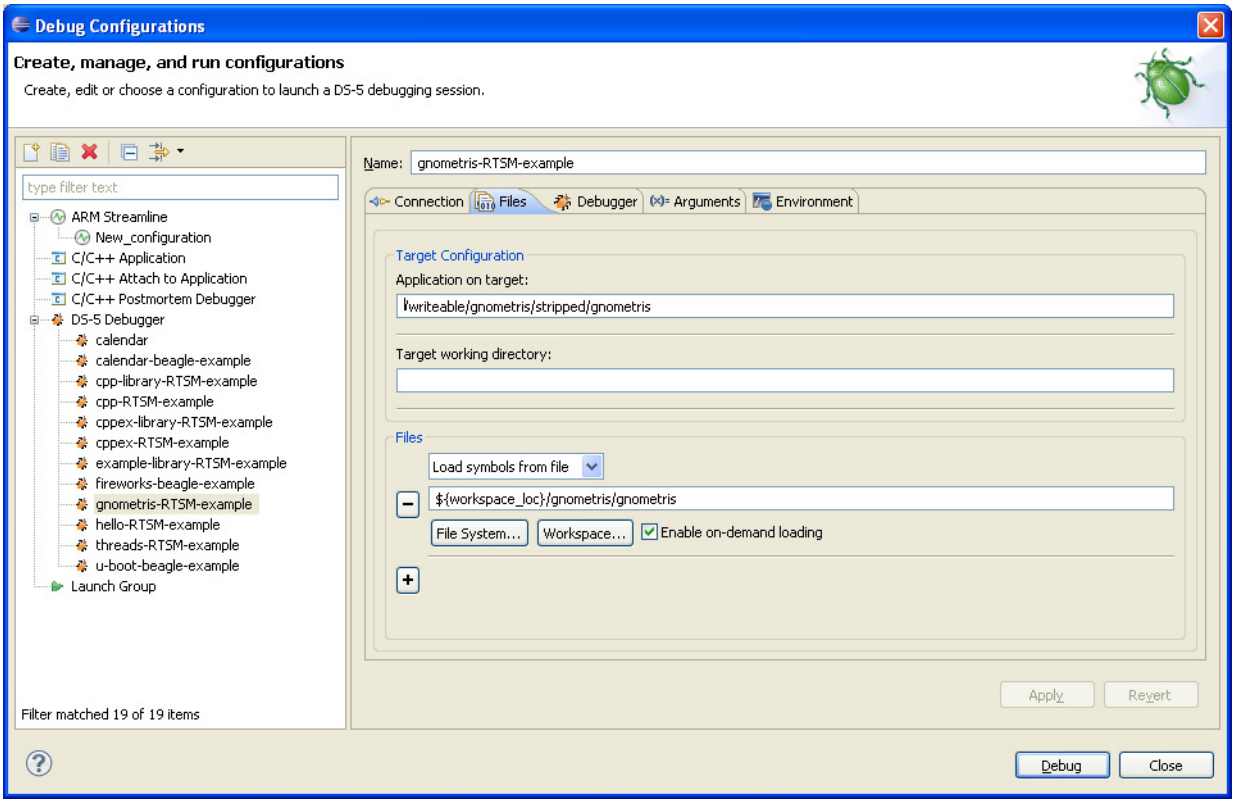

Figure 11-26 File system configuration for an application on a model

11.24.1 Files options summary

The Files options available depend on the debug operation you selected on the **Connection** tab. The possible Files options are:

Application on host to download

Specify the application image file on the host that you want to download to the target:

- enter the host location and file name in the field provided
- click on **File System...** to locate the file in an external directory from the Eclipse workspace
- click on **Workspace...** to locate the file in an project directory or sub-directory within the Eclipse workspace.

  For example, to download the stripped (no debug) Gnometris application image select the `gnometris/stripped/gnometris` file.

Select **Load symbols** to load the debug symbols from the specified image.

Select **Enable on-demand loading** to specify how you want the debugger to load debug information. Enabling this option can provide a faster load and use less memory but debugging might be slower.

Application on target

Specify the location of the application on the target. `gdbserver` uses this to launch the application.

For example, to use the stripped (no debug) Gnometris application image when using a model and VFS is configured to mount the host workspace as `/writeable` on the target, specify the following in the field provided:

`/writeable/gnometris/stripped/gnometris`.

Target download directory

If the target has a preloaded image, then you might have to specify the location of the corresponding image on your host.

The debugger uses the location of the application image on the target as the default current working directory. To change the default setting for the application that you are debugging, enter the location in the field provided. The current working directory is used whenever the application references a file using a relative path.

Load symbols from file

Specify the application image containing the debug information to load:

- enter the host location and file name in the field provided
- click on **File System...** to locate the file in an external directory from the workspace
- click on **Workspace...** to locate the file in a project directory or sub-directory within the workspace.

For example, to load the debug version of Gnometris you must select the `gnometris` application image that is available in the `gnometris` project root directory.

Select **Enable on-demand loading** to specify how you want the debugger to load debug information. Enabling this option can provide a faster load and use less memory but debugging might be slower.

Although you can specify shared library files here, the usual method is to specify a path to your shared libraries with the **Shared library search directory** option on the **Debugger** tab.

Other file on host to download

Specify other files that you want to download to the target:

- Enter the host location and file name in the field provided.
- Click on **File System...** to locate the file in an external directory from the workspace.
- Click on **Workspace...** to locate the file in a project directory or sub-directory within the workspace.

For example, to download the stripped (no debug) Gnometris shared library to the target you can select the `gnometris/stripped/libgames-support.so` file.

Target working directory

If this field is not specified, the debugger uses the location of the application image on the target as the default current working directory. To change the default setting for the application that you are debugging, enter the location in the field provided. The current working directory is used whenever the application refers to a file using a relative path.

Remove this resource from the list

 To remove a resource from the configuration settings, click this button next to the resource that you want to remove.

Add a new resource to the list

 To add a new resource to the file settings, click this button and then configure the options as required.

11.24.2 See also

Tasks

- *Configuring a connection to an RTSM model* on page 3-3
- *Configuring a connection to a Linux target using gdbserver* on page 3-5
- *ARM® DS-5™ Getting Started with DS-5*:
  - *Loading the Gnometris application on a Real-Time System Model* on page 3-6
  - *Loading the Gnometris application on to an ARM Linux target* on page 3-7.
- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *Debug Configurations - Connection tab* on page 11-59
- *Debug Configurations - Debugger tab* on page 11-67
- *Debug Configurations - Arguments tab* on page 11-71
- *Debug Configurations - Environment tab* on page 11-73
- *ARM DSTREAM Setting up the Hardware*, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0481-.

11.25 Debug Configurations - Debugger tab

The **Debugger** tab in the Debug Configurations dialog box enables you to specify the actions that you want the debugger to do after connection to the target.

Run Control

These options enable you to define the running state of the target when you connect:

Connect only

Connect to the target, but do not run the application.

Note

The PC register is not set and pending breakpoints or watchpoints are subsequently disabled when a connection is established.

Debug from entry point

Run the application when a connection is established, then stop at the image entry point.

Debug from symbol

Run the application when a connection is established, then stop at the address of the specified symbol. The debugger must be able to resolve the symbol. If you specify a C or C++ function name, then do not use the `()` suffix.

If the symbol can be resolved, execution stops at the address of that symbol.

If the symbol cannot be resolved, a message is displayed in the Commands view warning that the symbol cannot be found. The debugger then attempts to stop at the image entry point.

Run target initialization debugger script (.ds)

Select this option to execute a target initialization script file containing debugger commands after connection. To select a file:

- enter the location and file name in the field provided
- click on **File System...** to locate the file in an external directory from the workspace
- click on **Workspace...** to locate the file in a project directory or sub-directory within the workspace.

Run debug initialization debugger script (.ds)

Select this option to execute a script file containing debugger commands after connection and execution of any initialization script files. To select a file:

- enter the location and file name in the field provided
- click on **File System...** to locate the file in an external directory from the workspace
- click on **Workspace...** to locate the file in a project directory or sub-directory within the workspace.

Note

You might need to insert a `wait` command before a `run` or `continue` command to enable the debugger to connect and run the application to the specified function.

Execute debugger commands

Enter debugger commands in the field provided if you want to automatically execute specific debugger commands after connection and after all of the previous debugger script files have completed. Each line must contain only one debugger command.

Host working directory

The debugger uses the Eclipse workspace as the default working directory on the host. To change the default setting for the application that you are debugging, deselect the **Use default** check box and then:

- enter the location in the field provided
- click on **File System...** to locate the external directory
- click on **Workspace...** to locate the project directory.

Paths You can modify the search paths on the host used by the debugger when it displays source code.

Source search directory

Specify a directory to search for source files:

- enter the location and file name in the field provided
- click on **File System...** to locate the directory in an external location from the workspace
- click on **Workspace...** to locate the directory within the workspace.

Shared library search directory

Specify a directory to search for shared libraries:

- enter the location in the field provided
- click on **File System...** to locate the directory in an external location from the workspace
- click on **Workspace...** to locate the directory within the workspace.

Remove this resource from the list

 To remove a search path from the configuration settings, click this button next to the resource that you want to remove.

Add a new resource to the list

 To add a new search path to the configuration settings, click this button and then configure the options as required.

Apply Save the current configuration. This does not connect to the target.

Revert Undo any changes and revert to the last saved configuration.

Debug Connect to the target and close the Debug Configurations dialog box.

Close Close the Debug Configurations dialog box.

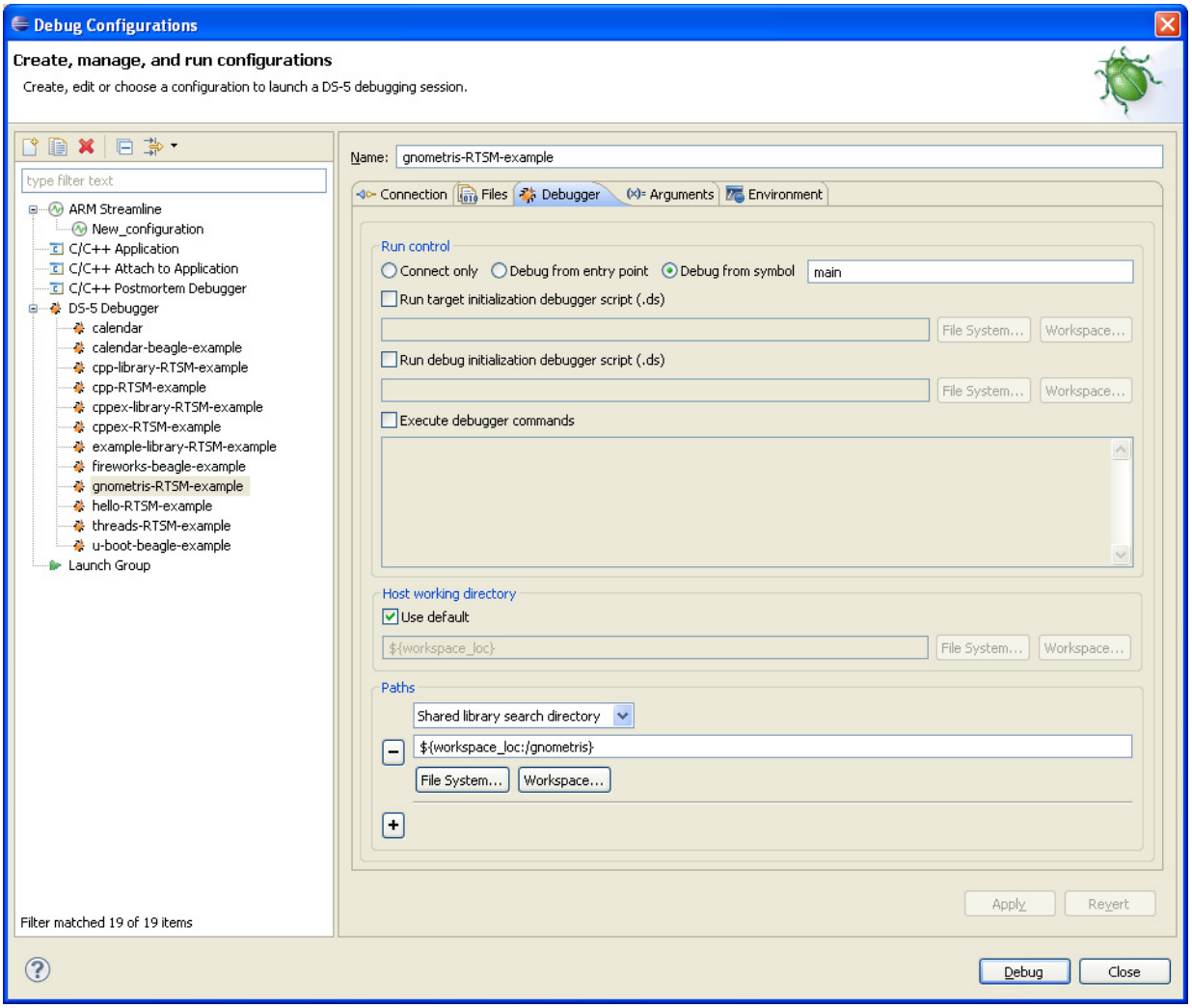

Figure 11-27 Debugger configuration to set starting point and search paths

11.25.1 See also

Tasks

- *Configuring a connection to an RTSM model* on page 3-3
- *Configuring a connection to a Linux target using gdbserver* on page 3-5
- *ARM® DS-5™ Getting Started with DS-5*:
  - *Loading the Gnometris application on a Real-Time System Model* on page 3-6
  - *Loading the Gnometris application on to an ARM Linux target* on page 3-7.
- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *Debug Configurations - Connection tab* on page 11-59
- *Debug Configurations - Files tab* on page 11-63
- *Debug Configurations - Arguments tab* on page 11-71
- *Debug Configurations - Environment tab* on page 11-73

- *DS-5 Debugger menu and toolbar icons* on page 11-75.

11.26 Debug Configurations - Arguments tab

If your application accepts command-line arguments to `main()`, you can specify the values to pass to the application when execution starts.

The **Arguments** tab in the Debug Configurations dialog box enables you to enter arguments that are passed to the application.

Note

This settings in this tab are not used for connections that use the **Connect to already running gdbserver** debug operation.

The **Arguments** tab contains the following elements:

Program Arguments

This panel enables you to enter the arguments. Arguments are passed to the target application unmodified except when the text is an eclipse argument variable of the form `${var_name}` where Eclipse replaces it with the related value.

For a Linux target you might have to escape some characters using a backslash (\)character. For example, the `@`, `(`, `)`, `"`, and `#` characters must be escaped.

Variables... This button opens the Select Variable dialog box where you can select variables that are passed to the application when the debug session starts.

Apply Save the current configuration. This does not connect to the target.

Revert Undo any changes and revert to the last saved configuration.

Debug Connect to the target and close the Debug Configurations dialog box.

Close Close the Debug Configurations dialog box.

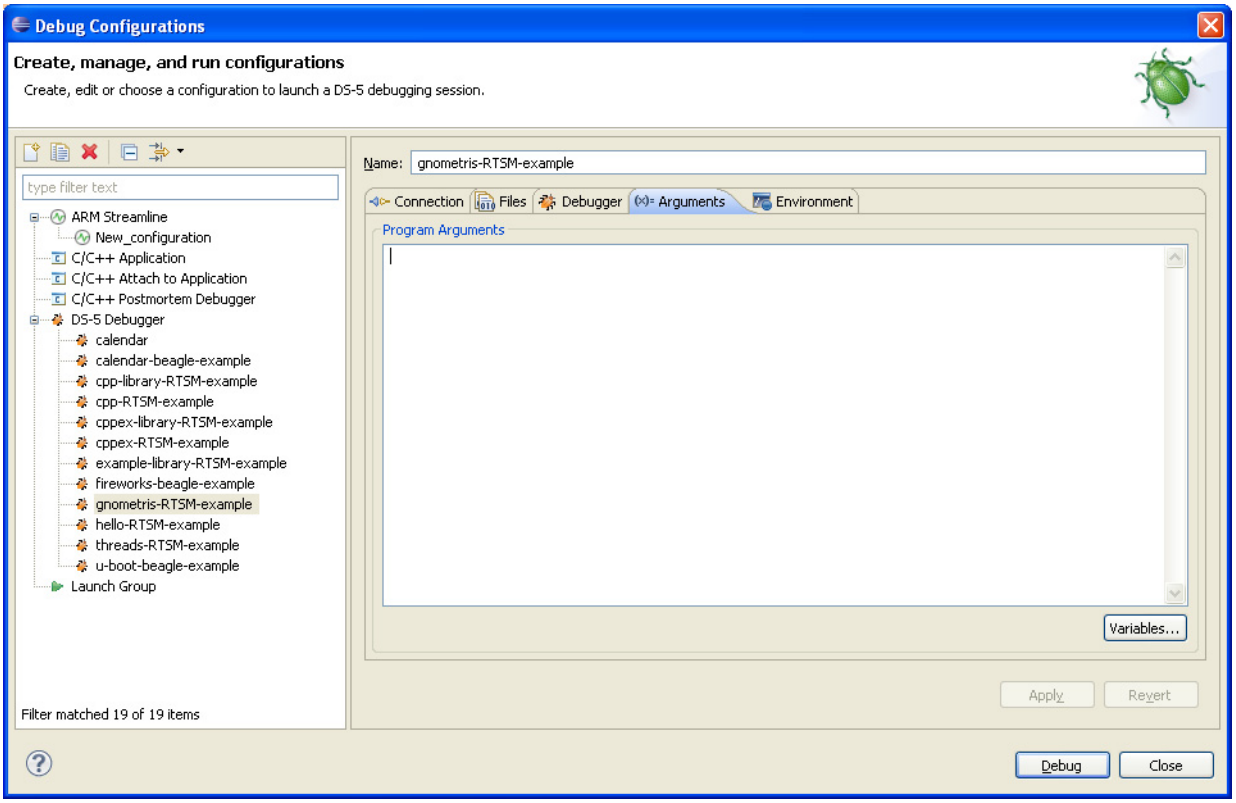

Figure 11-28 Application arguments configuration

11.26.1 See also

Tasks

- *Configuring a connection to an RTSM model* on page 3-3
- *Configuring a connection to a Linux target using gdbserver* on page 3-5
- *ARM® DS-5™ Getting Started with DS-5*:
  - *Loading the Gnometris application on a Real-Time System Model* on page 3-6
  - *Loading the Gnometris application on to an ARM Linux target* on page 3-7.
- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *Debug Configurations - Connection tab* on page 11-59
- *Debug Configurations - Files tab* on page 11-63
- *Debug Configurations - Debugger tab* on page 11-67
- *Debug Configurations - Environment tab* on page 11-73
- *DS-5 Debugger menu and toolbar icons* on page 11-75.

11.27 Debug Configurations - Environment tab

The **Environment** tab in the Debug Configurations dialog box enables you to create and configure the target environment variables that are passed to the application when the debug session starts.

Note

This settings in this tab are not used for connections that use the **Connect to already running gdbserver** debug operation.

The **Environment** tab contains the following elements:

Target environment variables to set

This panel displays the current target environment variables in use by the debugger.

New... This button opens the New Environment Variable dialog box where you can create a new target environment variable.

For example, to debug the Gnometris application on a model you must create a target environment variable for the `DISPLAY` setting.

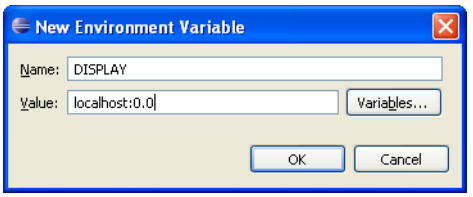

Figure 11-29 Setting up target environment variables

Edit... This button opens the Edit Environment Variable dialog box where you can edit the properties for the selected target environment variable.

Remove This button removes the select target environment variables from the list.

Apply Save the current configuration. This does not connect to the target.

Revert Undo any changes and revert to the last saved configuration.

Debug Connect to the target and close the Debug Configurations dialog box.

Close Close the Debug Configurations dialog box.

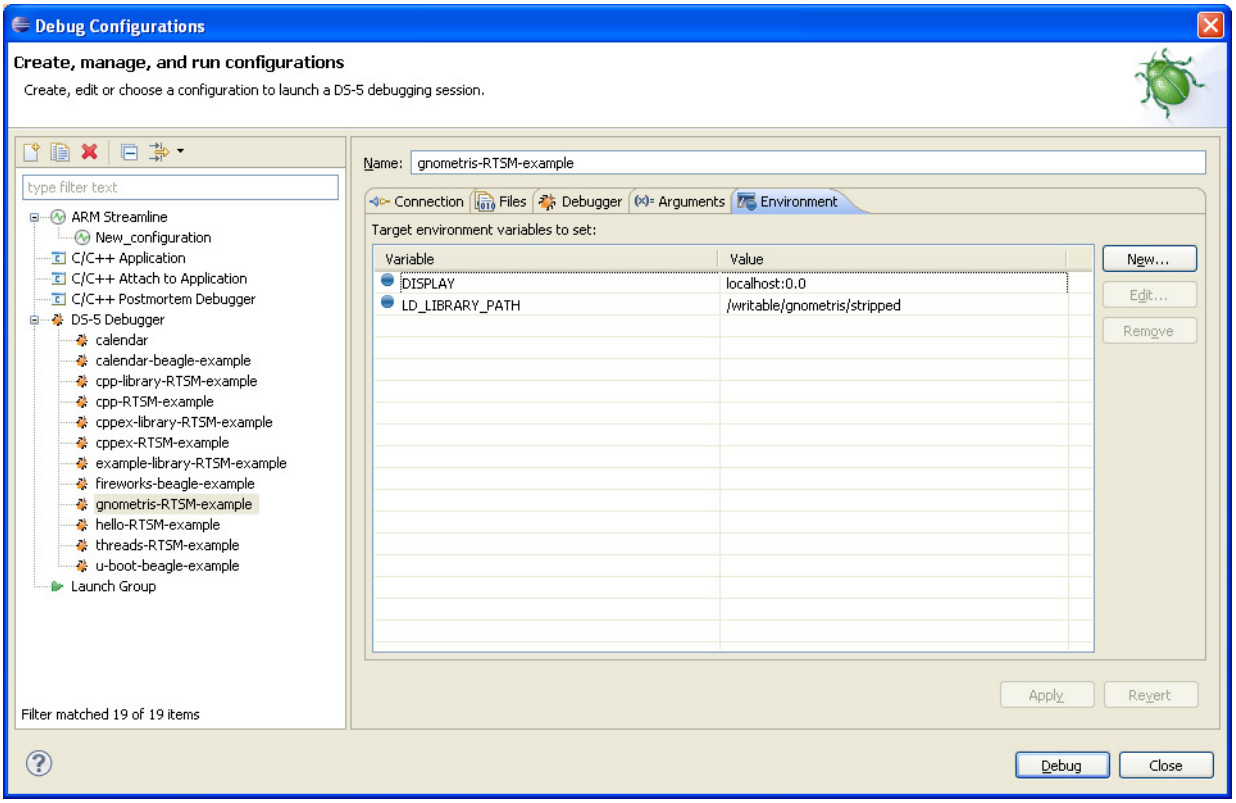

Figure 11-30 Environment configuration for a model

11.27.1 See also

Tasks

- *Configuring a connection to an RTSM model* on page 3-3
- *Configuring a connection to a Linux target using gdbserver* on page 3-5
- *ARM® DS-5™ Getting Started with DS-5*:
  - *Loading the Gnometris application on a Real-Time System Model* on page 3-6
  - *Loading the Gnometris application on to an ARM Linux target* on page 3-7.
- *ARM® DS-5™ Using Eclipse*:
  - *Accessing the dynamic help* on page 3-35.

Reference

- *Debug Configurations - Connection tab* on page 11-59
- *Debug Configurations - Files tab* on page 11-63
- *Debug Configurations - Debugger tab* on page 11-67
- *Debug Configurations - Arguments tab* on page 11-71
- *DS-5 Debugger menu and toolbar icons* on page 11-75.

11.28 DS-5 Debugger menu and toolbar icons

These tables list the most common menu and toolbar icons available for use with DS-5 Debugger. For information on icons, markers, and buttons not listed in the following tables, see the standard *Workbench User Guide* or the *C/C++ Development User Guide* in the Help Contents.

If you leave the mouse pointer positioned on a toolbar icon for a few seconds without clicking, a tooltip appears informing you of the purpose of the icon.

11.28.1 DS-5 Debugger icons

Table 11-2 DS-5 Debugger icons

Icon	Description	Icon	Description
	Connect to target		Connected to target
	Disconnect from target		Delete connection
	Start application and run to main		Start application and run to entry point
	Run application from entry point		Restart the application
	Continue running application		Stop application
	Step into		Step over
	Step out		Toggle stepping mode
	Collapse all configurations in stack trace		Call stack
	Thread		Process
	Kernel module		Define a new RSE connection
	Refresh the RSE resource tree		Save view contents to a file
	Clear view contents		Switch to History view
	Synchronize view contents		Toggle scroll lock
	Run commands from a script file		Export commands to a script file
	Remove selected breakpoint, watchpoints, or expression (view dependent)		Remove all breakpoints, watchpoints, or expressions (view dependent)
	Display breakpoint location in source file		Deactivate all breakpoints and watchpoints
	Import from a file		Export to a file
	Create new script file or add new expression (view dependent)		Run select script file
	Open selected file for editing		Delete the selected files
	Set display width		Set display format

Table 11-2 DS-5 Debugger icons (continued)

Icon	Description	Icon	Description
	Toggle the display of ASCII characters		Toggle freeze mode
	Edit Screen view parameters		Add new Screen view
	Add new Disassembly view		Add new Variables view
	Add new Registers view		Add new Memory view
	Add new Expression view		Add new Trace view
	View update in progress		Toggle trace marker
	Show next match		Show previous match
	Show function trace		Show instruction trace
	Toggle the views		Toggle navigation resolution

11.28.2 Perspective icons

Table 11-3 Perspective icons

Icon	Description	Icon	Description
	Open new perspective		C/C++
	DS-5 Debug		Fast view bar

11.28.3 View icons

Table 11-4 View icons

Button	Description	Button	Description
	Display drop-down menu		Synchronize view contents
	Minimize		Maximize
	Restore		Close

11.28.4 View markers

Table 11-5 View markers

Icon	Description	Icon	Description
	Software breakpoint enabled		Hardware breakpoint enabled
	Access watchpoint enabled		Read watchpoint enabled
	Write watchpoint enabled		Software breakpoint disabled
	Hardware breakpoint disabled		Access watchpoint disabled
	Read watchpoint disabled		Write watchpoint disabled
	Software breakpoint pending		Hardware breakpoint pending
	Access watchpoint pending		Read watchpoint pending
	Write watchpoint pending		Software breakpoint disconnected
	Hardware breakpoint disconnected		Access watchpoint disconnected
	Read watchpoint disconnected		Write watchpoint disconnected

Table 11-5 View markers (continued)

Icon	Description	Icon	Description
	Multiple-statement software breakpoint enabled		Multiple-statement software breakpoint disabled
	Error		Current location
	Warning		Bookmark
	Information		Task
	Search result		

11.28.5 Miscellaneous icons

Table 11-6 Miscellaneous icons

Icon	Description	Icon	Description
	Open a new resource wizard		Open new project wizard
	Open new folder wizard		Open new file wizard
	Open search dialog box		Display context-sensitive help
	Open import wizard		Open export wizard

Chapter 12
Troubleshooting

The following topics describe how to diagnose problems when debugging applications using DS-5 Debugger.

Concepts

- *ARM Linux problems and solutions* on page 12-2
- *Enabling internal logging from the debugger* on page 12-3
- *Target connection problems and solutions* on page 12-4.

Other information

- *ARM Technical Support Knowledge Articles*, `http://infocenter.arm.com/help/topic/com.arm.doc.faqs/index.html`.

12.1 ARM Linux problems and solutions

You might encounter the following problems when debugging a Linux application.

12.1.1 ARM Linux permission problem

If you receive a permission denied error message when starting an application on the target then you might need to change the execute permissions on the application. For example:

```
chmod +x myImage
```

12.1.2 A breakpoint is not being hit

You must ensure that the application and shared libraries on your target are the same as those on your host. The code layout must be identical, but the application and shared libraries on your target do not need to contain debug symbols.

12.1.3 See also

Tasks

- *About debugging shared libraries* on page 6-4.

12.2 Enabling internal logging from the debugger

On rare occasions an internal error might occur causing the debugger to generate an error message suggesting that you report it to your local support representatives. You can help to improve the debugger by giving feedback with an internal log that captures the stacktrace and shows where in the debugger the error occurs. To obtain the current version of DS-5, you can select **About ARM DS-5** from the **Help** menu in Eclipse or open the product release notes.

12.2.1 Procedure

To enable internal logging within Eclipse, enter the following in the Commands view of the **DS-5 Debug** perspective:

1. To enable the output of logging messages from the debugger using the predefined DEBUG level configuration:

   ```
   log config debug
   ```

2. To redirect all logging messages from the debugger to a file:

   ```
   log file debug.log
   ```

Note

Enabling internal logging can produce very large files and slow down the debugger significantly. Only enable internal logging when there is a problem.

12.2.2 See also

Tasks

- Chapter 1 *Conventions and feedback*.

Reference

- *Commands view* on page 11-15.
- *ARM® DS-5™ Debugger Command Reference*:
  - *log config* on page 2-104
  - *log file* on page 2-105.
- *ARM® DS-5™ Using Eclipse:*
  - *Perspectives and views* on page 3-19.

12.3 Target connection problems and solutions

You might encounter the following problems when connecting to a target.

12.3.1 Failing to make a connection

The debugger might fail to connect to the selected debug target because of the following reasons:

- you do not have a valid license to use the debug target
- the debug target is not installed or the connection is disabled
- the target hardware is in use by another user
- the connection has been left open by software that exited incorrectly
- the target has not been configured, or a configuration file cannot be located
- the target hardware is not powered up ready for use
- the target is on a scan chain that has been claimed for use by something else
- the target hardware is not connected
- you want to connect through gdbserver but the target is not running gdbserver
- there is no ethernet connection from the host to the target
- the port number in use by the host and the target are incorrect

Check the target connections and power up state, then try and reconnect to the target.

12.3.2 Debugger connection settings

When debugging a bare-metal target the debugger might fail to connect because of the following reasons:

- **Heap Base** address is incorrect
- **Stack Bas**e (top of memory) address is incorrect
- **Heap Limit** address is incorrect
- Incorrect vector catch settings.

Check that the memory map settings are correct for the selected target. If set incorrectly, the application might crash because of stack corruption or because the application overwrites its own code.

12.3.3 See also

Tasks

- *Configuring a connection to an RTSM model* on page 3-3
- *Configuring a connection to a Linux target using gdbserver* on page 3-5
- *Configuring a connection to a bare-metal target* on page 3-9
- *Disconnecting from a target* on page 3-11
- *About semihosting and top of memory* on page 8-2.

Other Information

- The documentation supplied with the board.