ARM® DS-5™
Version 5.9

Using the Debugger
ARM DS-5
Using the Debugger

Copyright © 2010-2012 ARM. All rights reserved.

Release Information

The following changes have been made to this book.

<table>
<thead>
<tr>
<th>Date</th>
<th>Issue</th>
<th>Confidentiality</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>June 2010</td>
<td>A</td>
<td>Non-Confidential</td>
<td>First release for DS-5</td>
</tr>
<tr>
<td>September 2010</td>
<td>B</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.2</td>
</tr>
<tr>
<td>November 2010</td>
<td>C</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.3</td>
</tr>
<tr>
<td>January 2011</td>
<td>D</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.4</td>
</tr>
<tr>
<td>May 2011</td>
<td>E</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.5</td>
</tr>
<tr>
<td>July 2011</td>
<td>F</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.6</td>
</tr>
<tr>
<td>September 2011</td>
<td>G</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.7</td>
</tr>
<tr>
<td>November 2011</td>
<td>H</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.8</td>
</tr>
<tr>
<td>February 2012</td>
<td>I</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.9</td>
</tr>
</tbody>
</table>

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Where the term ARM is used it means "ARM or any of its subsidiaries as appropriate".

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

Chapter 2  Getting started with the debugger
2.1 About the debugger ................................................................. 2-2
2.2 Debugger concepts ................................................................. 2-4
2.3 Launching the debugger from Eclipse ....................................... 2-6
2.4 Launching the debugger from the command-line console ............ 2-7
2.5 DS-5 Debug perspective keyboard shortcuts .............................. 2-10
2.6 DS-5 Debugger command-line console keyboard shortcuts .......... 2-11

Chapter 3  Configuring and connecting to a target
3.1 Types of target connections ....................................................... 3-2
3.2 Configuring a connection to an RTSM model ............................ 3-3
3.3 Configuring a connection to a Linux target using gdbserver .......... 3-5
3.4 Configuring a connection to a Linux Kernel .............................. 3-8
3.5 Configuring a connection to a bare-metal target ....................... 3-10
3.6 Configuring an Event Viewer connection to a bare-metal target .... 3-12
3.7 About the configuration database import utility ......................... 3-14
3.8 Launching the configuration database import utility .................... 3-17
3.9 Creating a new platform configuration in DS-5 from a model ........ 3-18
3.10 Creating a new platform configuration in DS-5 from an RVC file .... 3-19
3.11 Exporting an existing launch configuration .............................. 3-20
3.12 Importing an existing launch configuration .............................. 3-22
3.13 Disconnecting from a target .................................................... 3-24

Chapter 4  Controlling execution
4.1 About loading an image on to the target .................................... 4-2
4.2 About loading debug information into the debugger .................... 4-4
4.3 Running an image .................................................................... 4-6
Chapter 5  
Examining the target  
5.1 Examining the target execution environment ................................................. 5-2  
5.2 Examining the call stack ..................................................................................... 5-4  
5.3 About trace support ............................................................................................ 5-6  

Chapter 6  
Debugging embedded systems  
6.1 About endianness ............................................................................................... 6-2  
6.2 About accessing AHB and APB buses ............................................................... 6-3  
6.3 About virtual and physical memory .................................................................... 6-4  
6.4 About debugging hypervisors ............................................................................. 6-5  
6.5 About debugging bare-metal symmetric multiprocessing systems ..................... 6-6  
6.6 About debugging multi-threaded applications ................................................... 6-9  
6.7 About debugging shared libraries ....................................................................... 6-11  
6.8 About debugging a Linux kernel ......................................................................... 6-14  
6.9 About debugging Linux kernel modules ............................................................ 6-16  
6.10 About debugging TrustZone enabled targets .................................................. 6-18  

Chapter 7  
Debugging with scripts  
7.1 Exporting DS-5 Debugger commands generated during a debug session ......... 7-2  
7.2 Creating a DS-5 Debugger script ....................................................................... 7-3  
7.3 Creating a Jython script ..................................................................................... 7-4  
7.4 Creating a CMM-style script ............................................................................. 7-7  
7.5 Running a script ................................................................................................. 7-8  
7.6 Configuring a Jython project in Eclipse ............................................................ 7-9  

Chapter 8  
Controlling runtime messages  
8.1 About semihosting and top of memory .............................................................. 8-2  
8.2 Working with semihosting .................................................................................. 8-4  
8.3 Enabling automatic semihosting support in the debugger .................................. 8-5  
8.4 Controlling semihosting messages using the command-line console ............... 8-6  
8.5 Controlling the output of logging messages ...................................................... 8-7  
8.6 About Log4j configuration files ......................................................................... 8-8  
8.7 Customizing the output of logging messages from the debugger .................... 8-9  

Chapter 9  
Working with the Snapshot Viewer  
9.1 Creating a Snapshot Viewer initialization file .................................................... 9-2  
9.2 About the Snapshot Viewer ............................................................................... 9-5  
9.3 Connecting to the Snapshot Viewer ................................................................... 9-7  
9.4 Considerations when creating debugger scripts for the Snapshot Viewer ........ 9-8  

Chapter 10  
DS-5 Debug perspective and views  
10.1 App Console view ............................................................................................ 10-3  
10.2 ARM Asm Info view ......................................................................................... 10-5  
10.3 ARM assembler editor ...................................................................................... 10-6  
10.4 Breakpoints view ............................................................................................... 10-8
Contents

10.5 C/C++ editor ................................................................. 10-12
10.6 Channel editor for the Event Viewer .................................. 10-15
10.7 Commands view ......................................................... 10-17
10.8 Debug Control view .................................................... 10-20
10.9 Disassembly view ....................................................... 10-25
10.10 Expressions view ....................................................... 10-29
10.11 Functions view ......................................................... 10-32
10.12 History view ............................................................. 10-35
10.13 Memory view ............................................................ 10-37
10.14 Modules view ........................................................... 10-41
10.15 Registers view ......................................................... 10-44
10.16 Screen view ............................................................. 10-47
10.17 Scripts view ............................................................. 10-50
10.18 Target view ............................................................... 10-52
10.19 Trace view ............................................................... 10-54
10.20 Variables view .......................................................... 10-58
10.21 Export memory dialog box ............................................ 10-61
10.22 Import memory dialog box ......................................... 10-63
10.23 Export trace report dialog box ...................................... 10-65
10.24 Breakpoint properties dialog box .................................. 10-67
10.25 Watchpoint properties dialog box .................................. 10-71
10.26 Tracepoint properties dialog box .................................... 10-72
10.27 Manage Signals dialog box .......................................... 10-73
10.28 Functions Filter dialog box .......................................... 10-75
10.29 Debug Configurations - Connection tab ....................... 10-76
10.30 Debug Configurations - Files tab ................................. 10-81
10.31 Debug Configurations - Debugger tab ......................... 10-86
10.32 Debug Configurations - Arguments tab ....................... 10-90
10.33 Debug Configurations - Environment tab ...................... 10-93
10.34 Debug Configurations - Event Viewer tab .................... 10-96
10.35 DS-5 Debugger menu and toolbar icons ....................... 10-99

Chapter 11 Troubleshooting
11.1 ARM Linux problems and solutions ................................ 11-2
11.2 Enabling internal logging from the debugger ................... 11-3
11.3 Target connection problems and solutions ..................... 11-4
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.
- **monospace** 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
Conventions and feedback

• 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 0446I
• 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
• 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
- accessing variables, and register values
- navigating the call stack
- support for handling exceptions and Linux signals
- debugging multi-threaded Linux applications
- debugging Linux kernel modules, boot code and kernel porting
- debugging bare-metal Symmetric MultiProcessing (SMP) systems.

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 ARM® DS-5™ Getting Started with DS-5:
  - Importing the example projects into Eclipse on page 3-4
  - Building the Gnometris project from Eclipse on page 3-5
  - Loading the Gnometris application on a Real-Time System Model on page 3-7
  - Loading the Gnometris application on to an ARM Linux target on page 3-8
  - Using an RSE connection to work with an ARM Linux target on page 3-9
  - Connecting to the Gnometris application that is already running on a ARM Linux target on page 3-15
  - Debugging Gnometris on page 3-18
  - Debugging a loadable kernel module on page 3-19
  - Performance analysis of threads application running on ARM Linux on page 3-24
  - Loading the hello-neon application on to an Android target on page 3-28.

- 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:
- Installing DS-5 into a custom Eclipse environment on page 3-2.

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 2 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, nor the debugger itself. Examples include:
- debug hardware agents:
  - ARM DSTREAM™ unit
  - ARM RV6™ 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 3-8.
• Configuring a connection to a bare-metal target on page 3-10.

Concepts
• Debugger concepts on page 2-4.

Reference
• ARM® DS-5™ Setting up the ARM® DSTREAM™ Hardware, http://infocenter.arm.com/help/topic/com.arm.doc.dui0481-/index.html
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 3-8
- Configuring a connection to a bare-metal target on page 3-10.

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

Reference
- Debug Control view on page 10-20
- Commands view on page 10-17
- Breakpoints view on page 10-8
- Disassembly view on page 10-25
- Variables view on page 10-58
- Registers view on page 10-44
- Memory view on page 10-37
- Debug Configurations - Connection tab on page 10-76.
- ARM® DS-5™ Using Eclipse:
  — Perspectives and views on page 2-22.
2.4 Launching the debugger from the command-line console

To launch the debugger:

1. Launch a 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
   Specifies either the host:port for the connection between the debugger and gdbserver or a target configuration file such as an XML or RVC file.
   A serial connection requires an XML file similar to the following example:

   ```xml
   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:

   --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.

   --disable_semihosting
   Disables semihosting operations.

   --disable_semihosting_console
   Disables all semihosting operations to the debugger console.

   --enable_semihosting
   Enables semihosting operations.

   --help
   Displays a summary of the main command-line options.
--image=filename
    Specifies the image file for the debugger to load when it connects to the target.

--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.

--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.

--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.

--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.

--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.

--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. Available options are:
    linux    OS support for debug of Linux applications.
    linux-kernel
    OS support for debug of a Linux kernel.
--top_mem=address

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

Note

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

---

Example 2-2 Example of bare-metal command-line connection

```
debugger --target=beagleboard.rvc --target_device=Cortex-A8
```

---

When connected, use the DS-5 debugger commands to access the target and start debugging. For example, `info registers` displays all application level registers.

2.4.1 See also

Task

- Exporting DS-5 Debugger commands generated during a debug session 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

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 for example, **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. You can also use **ALT+F5** to step in the opposite mode. For example, if you are in source level stepping mode then using **ALT+F5** performs an instruction level step.
- **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** Start running the application image from the current location.

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

\[\text{Note}\]  
A Connect only connection might require setting the PC register to the start of the image before running it.

- **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 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
• 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 3-8
• Configuring a connection to a bare-metal target on page 3-10
• Configuring an Event Viewer connection to a bare-metal target on page 3-12
• About the configuration database import utility on page 3-14
• Launching the configuration database import utility on page 3-17
• Creating a new platform configuration in DS-5 from a model on page 3-18
• Creating a new platform configuration in DS-5 from an RVC file on page 3-19
• Exporting an existing launch configuration on page 3-20
• Importing an existing launch configuration on page 3-22
• Disconnecting from a target on page 3-24.

Concepts
• About the configuration database import utility on page 3-14
• Types of target connections on page 3-2.
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

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

- to gdbserver running on a 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.

Bare-metal and Linux kernel

To debug an application running on a bare-metal target, a Linux kernel, or a kernel device driver, 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 3-8
- Configuring a connection to a bare-metal target on page 3-10
- Exporting an existing launch configuration on page 3-20
- Importing an existing launch configuration on page 3-22.

Concepts

- Debugger concepts on page 2-4
- About the Snapshot Viewer on page 9-5.
3.2 Configuring a connection to an RTSM model

DS-5 supports serial connections between a Real-Time System Model (RTSM) and the host machine on both Windows and Linux platforms.

To connect to an 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.
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
- Exporting an existing launch configuration on page 3-20
- Importing an existing launch configuration on page 3-22
- ARM® DS-5™ Using Eclipse:
  - Accessing the dynamic help on page 2-38.

Concepts

- Types of target connections on page 3-2.

Reference

- Debug Configurations - Connection tab on page 10-76
- Debug Configurations - Files tab on page 10-81
- Debug Configurations - Debugger tab on page 10-86
- Debug Configurations - Arguments tab on page 10-90
- Debug Configurations - Environment tab on page 10-93
- ARM® DS-5™ Getting Started with DS-5:
  - About Real-Time System Models on page 2-5.
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:
     ```
     stty -F /dev/ttyS2 115200 -brkint -icrnl -imaxbel -opost -onlcr -isig -icanon -iexten -echo -echoe -echok -echoctl -echoke
     ```
3. If required, set up a *Remote System Explorer* (RSE) connection to the target.

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. :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.
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
- Exporting an existing launch configuration on page 3-20
- Importing an existing launch configuration on page 3-22
- ARM® DS-5™ Getting started with DS-5:
  — Using an RSE connection to work with an ARM Linux target on page 3-9.
- ARM® DS-5™ Using Eclipse:
  — Accessing the dynamic help on page 2-38.

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

**Reference**
- Debug Configurations - Connection tab on page 10-76
- Debug Configurations - Files tab on page 10-81
- Debug Configurations - Debugger tab on page 10-86
- Debug Configurations - Arguments tab on page 10-90
- Debug Configurations - Environment tab on page 10-93.
• *ARM*® DS-5™ Using Eclipse:
  — Chapter 5 *Working with Remote System Explorer*. 
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.
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 (\*vmlinux\*) compiled against exactly the same kernel version as the target.
   b. Module image compiled against exactly the same kernel version as the target, if required.
   Alternatively after connection, you can manually load debug information into the debugger by using the **Load...** menu option from the Debug Control view.
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.

--- Note ---
*Operating System* (OS) support is automatically enabled when a Linux kernel image is loaded into the debugger from the DS-5 Debugger launch configuration. However, you can manually control this by using the **set os** command.
For example, if you want to delay the activation of OS support until after the kernel has booted and the *Memory Management Unit* (MMU) is initialized then you can configure a connection that uses a target initialization script to disable OS support. To debug the kernel, OS support must be enabled in the debugger.
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

- *Launching the debugger from Eclipse* on page 2-6
- *Exporting an existing launch configuration* on page 3-20
- *Importing an existing launch configuration* on page 3-22
- *ARM® DS-5™ Using Eclipse:*
  - *Accessing the dynamic help* on page 2-38.
- *ARM® DS-5™ Getting Started with DS-5:*
  - *Debugging a loadable kernel module* on page 3-19.

#### Concepts

- *Types of target connections* on page 3-2
- *About debugging a Linux kernel* on page 6-14
- *About debugging Linux kernel modules* on page 6-16
- *ARM Linux problems and solutions* on page 11-2
- *Target connection problems and solutions* on page 11-4.

#### Reference

- *set os* on page 2-153
- *Debug Configurations - Connection tab* on page 10-76
- *Debug Configurations - Files tab* on page 10-81
- *Debug Configurations - Debugger tab* on page 10-86.
### 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.
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, using semihosting, to the application when the debug session starts.
9. Click on **Apply** to save the configuration settings.
10. Click on **Debug** to connect to the target.
11. 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
- Exporting an existing launch configuration on page 3-20
- Importing an existing launch configuration on page 3-22
- ARM® DS-5™ Using Eclipse:
  - Accessing the dynamic help on page 2-38.

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

Reference
- Debug Configurations - Connection tab on page 10-76
- Debug Configurations - Files tab on page 10-81
- Debug Configurations - Debugger tab on page 10-86
- Debug Configurations - Arguments tab on page 10-90
- Debug Configurations - Environment tab on page 10-93.
3.6 Configuring an Event Viewer connection to a bare-metal target

The Event Viewer allows you to capture and view textual logging information from bare-metal applications. Logging is captured from your application using annotations that you must add to the source code.

--- Note ---
The Event Viewer tab in the Debug Configurations dialog box is only enabled for targets where Instrumentation Trace Macrocell (ITM) capture is supported.

3.6.1 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.
- Annotate your application source code with logging points and recompile it. See the ITM and Event Viewer Example for Versatile Express A9x4 provided with DS-5 for more information.

3.6.2 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.
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. For example, ARM-Versatile Express A9x4.
   b. Select the Bare Metal Debug project type.
   c. Select the required debug operation. For example, Debug and Trace Cortex-A9x4 SMP via DSTREAM.
   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, using semihosting, to the application when the debug session starts.
9. Click on the **Event Viewer** tab to configure the ITM capture settings.
   a. Select **Enable ITM capture**.
   b. Enter the maximum size of the trace buffer. For example, you can enter 100MB for a DSTREAM connection. Be aware that larger buffers have a performance impact by taking longer to process but collect more trace data.
   c. Add a configuration for each ITM channel that you want to receive trace on. This is dependent on how your program configures and uses the ITM.

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 Channel editors for the Event Viewer.

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

### 3.6.3 See also

**Tasks**
- *Launching the debugger from Eclipse* on page 2-6
- *Exporting an existing launch configuration* on page 3-20
- *Importing an existing launch configuration* on page 3-22
- *ARM® DS-5™ Using Eclipse:*
  — *Accessing the dynamic help* on page 2-38.

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

**Reference**
- *Channel editor for the Event Viewer* on page 10-15
- *Debug Configurations - Connection tab* on page 10-76
- *Debug Configurations - Files tab* on page 10-81
- *Debug Configurations - Debugger tab* on page 10-86
- *Debug Configurations - Arguments tab* on page 10-90
- *Debug Configurations - Environment tab* on page 10-93
- *Debug Configurations - Event Viewer tab* on page 10-96.
3.7 About the configuration database import utility

The import utility, cdbimporter, aims to provide an easy method to import platform information into DS-5, and so provide limited debug and trace support for the platform through RVI/DSTREAM or model connections.

A target database holds the platform information for the DS-5. The import utility creates a new platform entry in a target database using information from:

- A configuration file created and saved using the Debug Hardware Configuration utility, dbghwconfig or rviconfig.
- A running model with a CADI server. The model can be already running or you can specify the launch path and filename in the command-line options.

ARM® recommends that you build new platform entries in a fresh directory to produce a new target database. This is because the DS-5 target database might install in a read-only location and when DS-5 updates, any changes to the DS-5 target database are lost. You can specify multiple target databases in DS-5 using the Preferences dialog. This enables platforms in the new database to use existing processor and register definitions.

Note

DS-5 is not yet capable of creating RVI/DSTREAM configuration files from within Eclipse, the Debug Hardware Configuration utility is currently the only method of doing this.

The import utility creates the following debug operations:

- Single processor bare-metal and Linux kernel debug for hardware and models.
- Symmetrical MultiProcessing (SMP) bare-metal and Linux kernel debug operations for targets with two or more identical processors.
- Embedded Trace Buffer on-chip (ETB) trace and Trace Port Interface Unit off-chip (TPIU) trace configuration for hardware targets containing ETB/TPIU devices. If a Cross Trigger Interface (CTI) is present, the import utility configures it to carry trace triggers.

Note

ETMv1.x is not supported.

- Linux applications debug configurations for hardware and models. ARM7TDMI® or Cortex™-M series processors are not supported.

The import utility does not create:

- debug operations that configure non-instruction trace macrocells
- big.LITTLE configurations.

For SMP, duplicate debug operations are produced for synchronisation with or without using CTI devices. Using CTIs produces a much tighter synchronization with a very low latency in order of cycles but the CTIs must be fully implemented and connected in line with the ARM reference designs, and must not be used for any other purpose. Synchronization without using CTIs has a much higher latency, but makes no assumptions about implementation or usage.

You might have to manually configure off-chip TPIU trace for multiplexed pins and also perform calibrations to cope with signal timing issues.

If you experience any problems or need to produce other configurations, contact your support representative.
3.7.1 Assumptions

The import utility makes the following assumptions when creating debug operations:

- There is a linear mapping between trace macrocells and CoreSight™ trace funnel ports.
- The Embedded Trace Macrocell (ETM)/Program Trace Macrocell (PTM) versions are fixed for each type of processor.

<table>
<thead>
<tr>
<th>Processor Type</th>
<th>ETM/PTM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cortex-A15</td>
<td>PTM</td>
</tr>
<tr>
<td>Cortex-A7</td>
<td>ETMv3.5</td>
</tr>
<tr>
<td>Cortex-A8</td>
<td>ETMv3.3</td>
</tr>
<tr>
<td>Cortex-A9</td>
<td>PTM</td>
</tr>
<tr>
<td>Cortex-R4</td>
<td>ETMv3.3</td>
</tr>
<tr>
<td>Cortex-R5</td>
<td>ETMv3.3</td>
</tr>
<tr>
<td>Cortex-R7</td>
<td>ETMv3.3</td>
</tr>
<tr>
<td>Cortex-M3</td>
<td>ETMv3.4</td>
</tr>
<tr>
<td>Cortex-M4</td>
<td>ETMv3.4</td>
</tr>
<tr>
<td>ARM9 series</td>
<td>ETMv1.x is not supported.</td>
</tr>
<tr>
<td>ARM11 series</td>
<td>ETMv3.1</td>
</tr>
</tbody>
</table>

- The CTI devices are not used for other operations.
- In a target containing multiple CoreSight ETBs, TPIUs or trace funnels, the import utility produces configuration for the first example of each trace funnel, ETB, and TPIU with the lowest base address.

3.7.2 Limitations

It is only possible to import platforms that can be auto-configured using the Debug Hardware Configuration utility or from a model.

DS-5 supports only a certain number of processors. To see a list of the supported processors, run the import utility using the --list-cores option (-l).

The import utility produces a basic configuration with appropriate processor and CP15 register sets but this might not always be perfect. For example, showing TrustZone® registers on all Cortex-A9 processors is not always appropriate, and NEON registers are never shown for an imported platform.

3.7.3 See also

Tasks

- Creating a new platform configuration in DS-5 from a model on page 3-18
- Creating a new platform configuration in DS-5 from an RVC file on page 3-19.
Reference

• Launching the configuration database import utility on page 3-17.
3.8 Launching the configuration database import utility

Launch the import utility using the following command-line syntax:

```
cdbimporter -cdb config_db -target-cdb destination_db {file.rvc|-model model_exec}
```

where:

- `-c=path,--cdb=path`
  Specifies a path to the main database containing processor and registered definitions.

- `-t=path,--target-cdb=path`
  Directory in which the destination database resides and receives the generated platform.

- `-m=path,--model=path`
  Imports from a model that provides a CADI server.
  - If you supply the path to the model executable, the utility launches the model and then interrogates it.
  - If you do not supply the path to the model executable, you can:
    - Omit the `path` value. This forces the utility to search for a running model to interrogate
    - Manually enter the data for the connection to the model. For example, processors names, IDs, and processor definitions.

- `RVC_file=path`
  Imports from a configuration file (.rvc) generated by the Debug Hardware Configuration utility, `rviconfig`.

- `-l,--list-cores`
  lists all the processors defined by the database supplied in the `--cdb` option.

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

3.8.1 See also

Tasks
- Creating a new platform configuration in DS-5 from a model on page 3-18
- Creating a new platform configuration in DS-5 from an RVC file on page 3-19.

Concepts
- About the configuration database import utility on page 3-14.
3.9 Creating a new platform configuration in DS-5 from a model

To import from a model with a CADI server:

1. Launch a command-line console:
   - On Windows, select **Start → All Programs → DS-5 Command Prompt**.
   - On Linux:
     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.

2. Ensure that the model is running or locate the model if it is not running.

3. Launch the configuration database import utility, `cdbimporter` from the command-line using the required options.

4. Follow the command prompts until successfully completed.

5. Launch Eclipse.
   a. Select **Preferences** from **Windows** menu.
   b. Expand the **DS-5** configuration group.
   c. Select **Target Database**.
   d. Click **Add...** to locate the new target database.
   e. Select the entire directory.
   f. Click **OK** to close the dialog box.
   g. Select the new target database in the Settings for the configuration database panel.
   h. Click down continuously to move the new database to the end of the list.
   i. Click **Rebuild database...**
   j. Click **OK** to close the dialog box and save the settings.

3.9.1 See also

**Tasks**
- *Creating a new platform configuration in DS-5 from an RVC file* on page 3-19.

**Concepts**
- *About the configuration database import utility* on page 3-14.

**Reference**
- *Launching the configuration database import utility* on page 3-17.
3.10 Creating a new platform configuration in DS-5 from an RVC file

You can use the Debug Hardware Configuration utility to connect to the target and save the information in a configuration file (.rvc). The resultant file contains limited debug and trace support for the platform that can be used to populate the DS-5 configuration database.

To import from an RVC file:

1. Launch a command-line console:
   - On Windows, select **Start → All Programs → DS-5 Command Prompt**.
   - On Linux:
     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.

2. Create a configuration file for the target connection.

3. Launch the configuration database import utility, `cdbimporter` from the command-line using the required options.

4. Follow the command prompts until successfully completed.

5. Launch Eclipse.
   a. Select **Preferences** from **Windows** menu.
   b. Expand the **DS-5** configuration group.
   c. Select **Target Database**.
   d. Click **Add...** to locate the new target database.
   e. Select the entire directory.
   f. Click **OK** to close the dialog box.
   g. Select the new target database in the Settings for the configuration database panel.
   h. Click down continuously to move the new database to the end of the list.
   i. Click **Rebuild database...**
   j. Click **OK** to close the dialog box and save the settings.

3.10.1 See also

Tasks
- **Creating a new platform configuration in DS-5 from a model** on page 3-18.

Concepts
- **About the configuration database import utility** on page 3-14.

Reference
- **Launching the configuration database import utility** on page 3-17.
3.11 Exporting an existing launch configuration

To export an existing launch configuration:

1. Select Export... from the File menu.
2. In the Export dialog box, expand the Run/Debug group and select Launch Configurations.

![Figure 3-1 Export launch configuration dialog box](image)

3. Click on Next.
4. In the Export Launch Configurations dialog box:
   a. Expand the DS-5 Debugger group and then select one or more launch configurations.
   b. Click on Browse... to select the required location in the local file system.
   c. Select the folder and then click OK.
5. If required, select Overwrite existing file(s) without warning.
6. Click on Finish.

3.11.1 See also

Tasks
- Importing an existing launch configuration on page 3-22.

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

Reference
- Debug Configurations - Connection tab on page 10-76
- Debug Configurations - Files tab on page 10-81
- Debug Configurations - Debugger tab on page 10-86
- Debug Configurations - Arguments tab on page 10-90
- Debug Configurations - Environment tab on page 10-93.
3.12 Importing an existing launch configuration

To import an existing launch configuration into Eclipse:

1. Select Import... from the File menu.
2. In the Import dialog box, expand the Run/Debug group and select Launch Configurations.

![Figure 3-3 Import launch configuration dialog box](image)

3. Click on Next.
4. Click on Browse... to select the required location in the local file system.
5. Select the folder containing the launch files and then click OK.
6. Select the checkboxes for the required folder and launch file.
7. If you are replacing an existing configuration with the same name then select **Overwrite existing launch configurations without warning**.

8. Click on **Finish**.

### 3.12.1 See also

**Tasks**

- *Exporting an existing launch configuration* on page 3-20.

**Concepts**

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

**Reference**

- *Debug Configurations - Connection tab* on page 10-76
- *Debug Configurations - Files tab* on page 10-81
- *Debug Configurations - Debugger tab* on page 10-86
- *Debug Configurations - Arguments tab* on page 10-90
- *Debug Configurations - Environment tab* on page 10-93.
3.13 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**.

3.13.1 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 3-8
- Configuring a connection to a bare-metal target on page 3-10.

Reference
- Debug Control view on page 10-20
- Commands view on page 10-17
- DS-5 Debugger menu and toolbar icons on page 10-99.
- ARM® DS-5™ Debugger Command Reference:
  — quit, exit on page 2-132.
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

• Running an image on page 4-6
• Setting an execution breakpoint on page 4-10
• Setting a data watchpoint on page 4-12
• Setting a tracepoint on page 4-14
• Setting a conditional breakpoint on page 4-15
• Setting a breakpoint on a specific thread on page 4-18
• Pending breakpoints and watchpoints on page 4-20
• Exporting DS-5 breakpoint settings to a file on page 4-22
• Importing DS-5 breakpoint settings from a file on page 4-24
• Stepping through an application on page 4-26
• Handling Unix signals on page 4-28
• Handling processor exceptions on page 4-30
• Configuring the debugger path substitution rules on page 4-32.

Concepts

• About loading an image on to the target on page 4-2
• About loading debug information into the debugger on page 4-4
• About breakpoints and watchpoints on page 4-7.
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 require 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.

**Load Offset**

Specifies a decimal or hexadecimal offset that is added to all addresses within the image. A hexadecimal offset must be prefixed with 0x.

**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 3-10
- Disconnecting from a target on page 3-24
- 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 10-20
• Commands view on page 10-17
• Debug Configurations - Connection tab on page 10-76.
• ARM® DS-5™ Debugger Command Reference:
  — load on page 2-111
  — loadfile on page 2-112.
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 can 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.
- **Load Offset** Specifies a decimal or hexadecimal offset that is added to all addresses within the image. A hexadecimal offset must be prefixed with 0x.
- **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.
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 3-10
- Disconnecting from a target on page 3-24
- Running an image on page 4-6
- Configuring the debugger path substitution rules on page 4-32.

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

Reference
- Debug Control view on page 10-20
- Commands view on page 10-17
- Debug Configurations - Connection tab on page 10-76
- ARM® DS-5™ Debugger Command Reference:
  - add-symbol-file on page 2-30
  - discard-symbol-file on page 2-59
  - file, symbol-file on page 2-68
  - set substitute-path on page 2-162.
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 10-20
- Commands view on page 10-17
- Debug Configurations - Connection tab on page 10-76
- Debug Configurations - Files tab on page 10-81
- Debug Configurations - Debugger tab on page 10-86
- ARM® DS-5™ Debugger Command Reference:
  — continue on page 2-50
  — run on page 2-138
  — start on page 2-194.
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 re-evaluated when the shared object is unloaded. Those with addresses that can be resolved are set and the others remain pending.

• If a breakpoint is set by function name then only inline instances that have been already demand loaded are found. To find all the inline instances of a function you must disable on-demand loading.

4.4.2 See also

Tasks
• Setting an execution breakpoint on page 4-10
• Setting a data watchpoint on page 4-12
• Setting a conditional breakpoint on page 4-15
• Setting a breakpoint on a specific thread on page 4-18
• Pending breakpoints and watchpoints on page 4-20
• Exporting DS-5 breakpoint settings to a file on page 4-22
• Importing DS-5 breakpoint settings from a file on page 4-24.

Concepts
• Debugger concepts on page 2-4.

Reference
• ARM® DS-5™ Debugger Command Reference:
  — advance on page 2-32
  — awatch on page 2-34
  — break on page 2-37
  — break-stop-on-threads, break-stop-on-cores on page 2-42
  — break-stop-on-vmid on page 2-44
  — clear on page 2-46
  — condition on page 2-49
  — delete breakpoints on page 2-53
  — disable breakpoints on page 2-56
  — enable breakpoints on page 2-64
  — hbreak on page 2-73
  — ignore on page 2-77
  — info breakpoints, info watchpoints on page 2-80
  — resolve on page 2-136
  — rwatch on page 2-139
— set breakpoint on page 2-147
— tbreak on page 2-201
— thbreak on page 2-203
— watch on page 2-212.
### 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 marker bar 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](image)

### 4.5.1 See also

**Tasks**

- *Setting a conditional breakpoint on page 4-15*
- *Setting a breakpoint on a specific thread on page 4-18*
Controlling execution

• Pending breakpoints and watchpoints on page 4-20
• Exporting DS-5 breakpoint settings to a file on page 4-22
• Importing DS-5 breakpoint settings from a file on page 4-24.

Concept
• About breakpoints and watchpoints on page 4-7.

Reference
• ARM assembler editor on page 10-6
• Breakpoints view on page 10-8
• C/C++ editor on page 10-12
• Commands view on page 10-17
• Debug Control view on page 10-20
• Disassembly view on page 10-25
• Registers view on page 10-44
• Memory view on page 10-37
• Variables view on page 10-58.
• ARM® DS-5™ Debugger Command Reference:
  — advance on page 2-32
  — break on page 2-37
  — break-stop-on-threads, break-stop-on-cores on page 2-42
  — break-stop-on-vmid on page 2-44
  — clear on page 2-46
  — condition on page 2-49
  — delete breakpoints on page 2-53
  — disable breakpoints on page 2-56
  — enable breakpoints on page 2-64
  — hbreak on page 2-73
  — ignore on page 2-77
  — info breakpoints, info watchpoints on page 2-80
  — resolve on page 2-136
  — set breakpoint on page 2-147
  — tbreak on page 2-201
  — thbreak on page 2-203.
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, select **Toggle Watchpoint**, select the required access type, and then click **OK**. The watchpoint is visible in the Variables view and also in the Breakpoints view.

![Figure 4-4 Setting a data watchpoint](image)

---

**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 correspond with the instruction at 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 4-20
- *Exporting DS-5 breakpoint settings to a file* on page 4-22

**Concept**

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

**Reference**

- *ARM assembler editor* on page 10-6
- *Breakpoints view* on page 10-8
- *C/C++ editor* on page 10-12
- *Commands view* on page 10-17
- *Debug Control view* on page 10-20
- *Disassembly view* on page 10-25
- *Expressions view* on page 10-29
- *Memory view* on page 10-37
- *Registers view* on page 10-44
• Variables view on page 10-58.
• ARM® DS-5™ Debugger Command Reference:
  — awatch on page 2-34
  — clearwatch on page 2-48
  — break-stop-on-threads, break-stop-on-cores on page 2-42
  — break-stop-on-vmid on page 2-44
  — delete breakpoints on page 2-53
  — disable breakpoints on page 2-56
  — enable breakpoints on page 2-64
  — info breakpoints, info watchpoints on page 2-80
  — rwatch on page 2-139
  — set breakpoint on page 2-147
  — watch on page 2-212.
4.7 Setting a tracepoint

Tracepoints are memory locations that are used to trigger behavior in a trace capture device when running an application. A tracepoint is hit when the processor executes an instruction at a specific address. Depending on the type, trace capture is either enabled or disabled.

Tracepoints can be set from any of the following views:
- ARM Assembler editor
- C/C++ editor
- Disassembly view
- Functions view
- Memory view
- Disassembly panel of the Trace view.

To set a tracepoint, right-click in the left-hand marker bar at the position where you want to set the tracepoint and select either Toggle Trace Start Point, Toggle Trace Stop Point, or Toggle Trace Trigger Point from the context menu. To remove a tracepoint, repeat this procedure on the same tracepoint or delete it from the Breakpoints view.

Tracepoints are stored on a per connection basis. If the active connection is disconnected then tracepoints can only be created from the source editor.

All tracepoints are visible in the Breakpoints view.

4.7.1 See also

Tasks
- Pending breakpoints and watchpoints on page 4-20.

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

Reference
- ARM assembler editor on page 10-6
- Breakpoints view on page 10-8
- C/C++ editor on page 10-12
- Disassembly view on page 10-25
- Memory view on page 10-37
- Functions view on page 10-32
- Trace view on page 10-54
- Variables view on page 10-58
- Tracepoint properties dialog box on page 10-72.
4.8 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 or processor.

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 or processors:
   a. Select Break on Selected Threads or Cores to enable the selection panel.
b. Select the checkbox for each thread or processor that you want to assign the breakpoint to.

8. Click OK to save the changes.

**Figure 4-5 Setting a conditional breakpoint**

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.

### 4.8.1 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 not activate until it has been hit with `c==5` for the fourth time. It subsequently activates every time it is hit with `c==5`.

- If you choose to break on selected thread or processor, then the Stop Condition and Ignore Count are checked only for the selected thread or processor.

- Conditions are evaluated in the following order:
  1. thread or processor
  2. condition
3. **ignore count.**

### 4.8.2 **See also**

**Tasks**
- Setting an execution breakpoint on page 4-10
- Setting a breakpoint on a specific thread on page 4-18
- Pending breakpoints and watchpoints on page 4-20
- Exporting DS-5 breakpoint settings to a file on page 4-22
- Importing DS-5 breakpoint settings from a file on page 4-24.

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

**Reference**
- ARM assembler editor on page 10-6
- Breakpoints view on page 10-8
- C/C++ editor on page 10-12
- Breakpoint properties dialog box on page 10-67
- Commands view on page 10-17
- Debug Control view on page 10-20
- Disassembly view on page 10-25
- Memory view on page 10-37
- Registers view on page 10-44
- Variables view on page 10-58.
- ARM® DS-5™ Debugger Command Reference:
  - advance on page 2-32
  - break on page 2-37
  - break-stop-on-threads, break-stop-on-cores on page 2-42
  - break-stop-on-vmid on page 2-44
  - clear on page 2-46
  - condition on page 2-49
  - delete breakpoints on page 2-53
  - disable breakpoints on page 2-56
  - enable breakpoints on page 2-64
  - hbreak on page 2-73
  - ignore on page 2-77
  - info breakpoints, info watchpoints on page 2-80
  - resolve on page 2-136
  - set breakpoint on page 2-147
  - silence on page 2-192
  - tbreak on page 2-201
  - thbreak on page 2-203.
### 4.9 Setting a 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 to 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](image-url)
4.9.1 See also

Tasks
- Setting an execution breakpoint on page 4-10
- Setting a conditional breakpoint on page 4-15
- Pending breakpoints and watchpoints on page 4-20
- Exporting DS-5 breakpoint settings to a file on page 4-22
- Importing DS-5 breakpoint settings from a file on page 4-24.

Reference
- ARM assembler editor on page 10-6
- Breakpoints view on page 10-8
- C/C++ editor on page 10-12
- Commands view on page 10-17
- Debug Control view on page 10-20
- Disassembly view on page 10-25
- Memory view on page 10-37
- Registers view on page 10-44
- Variables view on page 10-58.
- ARM® DS-5™ Debugger Command Reference:
  - break on page 2-37
  - break-stop-on-threads, break-stop-on-cores on page 2-42
  - break-stop-on-vmid on page 2-44
  - info threads on page 2-104
  - thread, core on page 2-205.
4.10  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, tbreak, 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

4.10.1  See also

**Tasks**
- Setting an execution breakpoint on page 4-10
- Setting a data watchpoint on page 4-12
- Setting a conditional breakpoint on page 4-15
- Setting a breakpoint on a specific thread on page 4-18
- Exporting DS-5 breakpoint settings to a file on page 4-22
- Importing DS-5 breakpoint settings from a file on page 4-24.

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

**Reference**
- ARM assembler editor on page 10-6
- Breakpoints view on page 10-8
- C/C++ editor on page 10-12
- Commands view on page 10-17
- Debug Control view on page 10-20
- Disassembly view on page 10-25
- Memory view on page 10-37
- Registers view on page 10-44
- Variables view on page 10-58.
- ARM® DS-5™ Debugger Command Reference:
  - advance on page 2-32
  - awatch on page 2-34
  - break on page 2-37
— break-stop-on-threads, break-stop-on-cores on page 2-42
— break-stop-on-vmid on page 2-44
— clear on page 2-46
— condition on page 2-49
— delete breakpoints on page 2-53
— disable breakpoints on page 2-56
— enable breakpoints on page 2-64
— hbreak on page 2-73
— ignore on page 2-77
— info breakpoints, info watchpoints on page 2-80
— resolve on page 2-136
— rwatch on page 2-139
— set breakpoint on page 2-147
— tbreak on page 2-201
— thbreak on page 2-203
— watch on page 2-212.
4.11 Exporting DS-5 breakpoint settings to a file

To export DS-5 breakpoint settings to a file:

1. Ensure that you are in the DS-5 Debug perspective.
2. Select Export Breakpoints from the Breakpoints view menu.
3. Select the required location in the local file system and enter a filename.
4. Click Save.

__Note__

All breakpoints and watchpoints shown in the DS-5 Breakpoints view are saved.

4.11.1 See also

Tasks
- Setting an execution breakpoint on page 4-10
- Setting a data watchpoint on page 4-12
- Setting a conditional breakpoint on page 4-15
- Setting a breakpoint on a specific thread on page 4-18
- Pending breakpoints and watchpoints on page 4-20
- Importing DS-5 breakpoint settings from a file on page 4-24.

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

Reference
- Breakpoints view on page 10-8
- Commands view on page 10-17
- Debug Control view on page 10-20
- Disassembly view on page 10-25
- Memory view on page 10-37
- Registers view on page 10-44
- Variables view on page 10-58.
- ARM® DS-5™ Debugger Command Reference:
  - advance on page 2-32
  - awatch on page 2-34
  - break on page 2-37
  - break-stop-on-threads, break-stop-on-cores on page 2-42
  - break-stop-on-vmid on page 2-44
  - clear on page 2-46
  - condition on page 2-49
  - delete breakpoints on page 2-53
  - disable breakpoints on page 2-56
  - enable breakpoints on page 2-64
  - hbreak on page 2-73
  - ignore on page 2-77
  - info breakpoints, info watchpoints on page 2-80
— resolve on page 2-136
— rwatch on page 2-139
— set breakpoint on page 2-147
— tbreak on page 2-201
— thbreak on page 2-203
— watch on page 2-212.
4.12 Importing DS-5 breakpoint settings from a file

To import DS-5 breakpoint settings from a file:

1. Ensure that you are in the DS-5 Debug perspective.
2. Select **Import Breakpoints** from the **Breakpoints** view menu.
3. Select the file containing the breakpoint settings from the local file system.
4. Click **Open**.

**Note**
Existing settings for the current connection are deleted and replaced by the DS-5 breakpoint and watchpoint settings from the file.

4.12.1 See also

**Tasks**
- Setting an execution breakpoint on page 4-10
- Setting a data watchpoint on page 4-12
- Setting a conditional breakpoint on page 4-15
- Setting a breakpoint on a specific thread on page 4-18
- Pending breakpoints and watchpoints on page 4-20
- Exporting DS-5 breakpoint settings to a file on page 4-22.

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

**Reference**
- Breakpoints view on page 10-8
- Commands view on page 10-17
- Debug Control view on page 10-20
- Disassembly view on page 10-25
- Memory view on page 10-37
- Registers view on page 10-44
- Variables view on page 10-58.
- **ARM® DS-5™ Debugger Command Reference**:
  - advance on page 2-32
  - awatch on page 2-34
  - break on page 2-37
  - break-stop-on-threads, break-stop-on-cores on page 2-42
  - break-stop-on-vmid on page 2-44
  - clear on page 2-46
  - condition on page 2-49
  - delete breakpoints on page 2-53
  - disable breakpoints on page 2-56
  - enable breakpoints on page 2-64
  - hbreak on page 2-73
  - ignore on page 2-77
— info breakpoints, info watchpoints on page 2-80
— resolve on page 2-136
— rwatch on page 2-139
— set breakpoint on page 2-147
— tbreak on page 2-201
— thbreak on page 2-203
— watch on page 2-212.
4.13 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.

![Debug Control view](image)

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
```

4.13.1 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 6-9
- About debugging shared libraries on page 6-11
- Handling Unix signals on page 4-28
- Handling processor exceptions on page 4-30.
Concepts

• About breakpoints and watchpoints on page 4-7.

Reference

• Breakpoints view on page 10-8
• Commands view on page 10-17
• Debug Control view on page 10-20
• Disassembly view on page 10-25
• Memory view on page 10-37
• Registers view on page 10-44
• Variables view on page 10-58.
• ARM® DS-5™ Debugger Command Reference:
  — next on page 2-123
  — nexti on page 2-124
  — nexts on page 2-125
  — finish on page 2-70
  — set step-mode on page 2-160
  — show step-mode on page 2-186
  — step on page 2-196
  — stepi on page 2-197
  — steps on page 2-198.
4.14 Handling Unix signals

For Linux applications, ARM processors have the facility to trap Unix signals. These are managed in the debugger by selecting **Manage Signals** from the Breakpoints view menu or you can use the `handle` command. You can also use the `info signals` command to display the current 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](Image)

**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.

### 4.14.1 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
```

4.14.2 See also

Tasks

- Stepping through an application on page 4-26
- Examining the target execution environment on page 5-2
- Examining the call stack on page 5-4
- About debugging multi-threaded applications on page 6-9
- About debugging shared libraries on page 6-11
- Handling processor exceptions on page 4-30.

Concepts

- About breakpoints and watchpoints on page 4-7.

Reference

- Commands view on page 10-17
- Breakpoints view on page 10-8
- Manage Signals dialog box on page 10-73.
- ARM® DS-5™ Debugger Command Reference:
  - handle on page 2-72
  - info signals, info handle on page 2-99.
4.15 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, except that vector catches use dedicated hardware in the processor and do not use up valuable breakpoint resources. This is called vector catch. Vector catch is managed in the debugger by selecting Manage Signals from the Breakpoints view menu or you can use the handle command. You can also use the info signals command to display the current handler settings.

The vector catch events that are available are dependent on the exact processor that you are connected to.

![Manage Signals](image)

**Figure 4-9 Manage exception handler settings**

4.15.1 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, a NON-SECURE_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 NON-SECURE_FIQ stop          # Enable stop and print on a NON-SECURE_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 NON-SECURE_FIQ nostop        # Disable stop on a NON-SECURE_FIQ exception
```

4.15.2 See also

Tasks

- Stepping through an application on page 4-26
- Examining the target execution environment on page 5-2
- Examining the call stack on page 5-4
- About debugging multi-threaded applications on page 6-9.
• About debugging shared libraries on page 6-11
• Handling Unix signals on page 4-28.

Concepts
• About breakpoints and watchpoints on page 4-7.

Reference
• Commands view on page 10-17
• Breakpoints view on page 10-8
• Manage Signals dialog box on page 10-73.
• ARM® DS-5™ Debugger Command Reference:
  — handle on page 2-72
  — info signals, info handle on page 2-99.
4.16 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, a warning is displayed in the C/C++ editor. Click on Set Path Substitution.
   - In the Debug Control view, select Path Substitution from the view menu.

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.
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.

### 4.16.1 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 2-38.

**Reference**
- *C/C++ editor* on page 10-12
- *Debug Control view* on page 10-20
- *ARM® DS-5™ Debugger Command Reference:*
  - *set substitute-path* on page 2-162.
Chapter 5
Examining the target

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

Tasks
• Examining the target execution environment on page 5-2
• Examining the call stack on page 5-4.

Concepts
• About trace support on page 5-6.
5.1 Examining the target execution environment

During a debug session you might want to display, 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 current 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.

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 4-10
- Setting a data watchpoint on page 4-12
- Setting a tracepoint on page 4-14
- Setting a conditional breakpoint on page 4-15
• Pending breakpoints and watchpoints on page 4-20
• Stepping through an application on page 4-26
• Examining the call stack on page 5-4
• Handling Unix signals on page 4-28
• Handling processor exceptions on page 4-30.

Concepts
• About debugging bare-metal symmetric multiprocessing systems on page 6-6
• About debugging multi-threaded applications on page 6-9
• About debugging shared libraries on page 6-11
• About debugging a Linux kernel on page 6-14
• About debugging Linux kernel modules on page 6-16.

Reference
• App Console view on page 10-3
• ARM Asm Info view on page 10-5
• ARM assembler editor on page 10-6
• Breakpoints view on page 10-8
• C/C++ editor on page 10-12
• Commands view on page 10-17
• Debug Control view on page 10-20
• Disassembly view on page 10-25
• Expressions view on page 10-29
• Functions view on page 10-32
• History view on page 10-35
• Memory view on page 10-37
• Registers view on page 10-44
• Screen view on page 10-47
• Scripts view on page 10-50
• Target view on page 10-52
• Trace view on page 10-54
• Variables view on page 10-58.

• ARM® DS-5™ Debugger Command Reference:
  — disassemble on page 2-58
  — info address on page 2-78
  — info functions on page 2-86
  — info locals on page 2-89
  — info registers on page 2-96
  — info symbol on page 2-102
  — info variables on page 2-105
  — print, inspect on page 2-129
  — whatis on page 2-214
  — x on page 2-217.
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 4-10
• Setting a data watchpoint on page 4-12
• Setting a tracepoint on page 4-14
• Setting a conditional breakpoint on page 4-15
• Pending breakpoints and watchpoints on page 4-20
• Stepping through an application on page 4-26
• Examining the target execution environment on page 5-2
• Handling Unix signals on page 4-28
• Handling processor exceptions on page 4-30.

Concepts
• About debugging bare-metal symmetric multiprocessing systems on page 6-6
• About debugging multi-threaded applications on page 6-9
• About debugging shared libraries on page 6-11
• About debugging a Linux kernel on page 6-14
• About debugging Linux kernel modules on page 6-16.

Reference

• App Console view on page 10-3
• ARM Asm Info view on page 10-5
• ARM assembler editor on page 10-6
• Breakpoints view on page 10-8
• C/C++ editor on page 10-12
• Commands view on page 10-17
• Debug Control view on page 10-20
• Disassembly view on page 10-25
• Expressions view on page 10-29
• Functions view on page 10-32
• History view on page 10-35
• Memory view on page 10-37
• Registers view on page 10-44
• Screen view on page 10-47
• Scripts view on page 10-50
• Target view on page 10-52
• Trace view on page 10-54
• Variables view on page 10-58.

• ARM® DS-5™ Debugger Command Reference:
  — down on page 2-60
  — down-silently on page 2-61
  — frame on page 2-71
  — info frame on page 2-85
  — info registers on page 2-96
  — info stack, backtrace, where on page 2-101
  — select-frame on page 2-141
  — set backtrace on page 2-145
  — up on page 2-209
  — up-silently on page 2-210.
5.3 About trace support

ARM® DS-5™ enables you to perform tracing on your application or system. Tracing is the ability to capture in real-time a historical, non-invasive debug of instructions. It 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, such as an ARM DSTREAM™ unit with a connection to a trace stream
• the debugger is connected to the debug hardware agent.

5.3.1 Trace hardware

Trace is typically provided by an external hardware block connected to the processor. This is known as an Embedded Trace Macrocell™ (ETM™) or Program Trace Macrocell (PTM) and is an optional part of an ARM architecture-based system. System-on-chip designers might omit this block from their silicon to reduce costs. These blocks observe (but do not affect) the processor behavior and are able to monitor instruction execution and data accesses.

There are two main problems with capturing trace. The first is that with very high processor clock speeds, even a few seconds of operation can mean billions of cycles of execution. Clearly, to look at this volume of information would be extremely difficult. The second, related problem is that modern processors could potentially perform one or more 64-bit cache accesses per cycle and to record both the data address and data values might require large bandwidth. This presents a problem in that typically, only a few pins are provided on the chip and these outputs might be able to be switched at significantly lower rates than the processor can be clocked at. It is very easy to exceed the capacity of the trace port. To solve this latter problem, the trace macrocell tries to compress information to reduce the bandwidth required. However, the main method to deal with these issues is to control the trace block so that only selected trace information is gathered. For example, trace only execution, without recording data values, or trace only data accesses to a particular peripheral or during execution of a particular function.

In addition, it is common to store trace information in an on-chip memory buffer, the Embedded Trace Buffer™ (ETB™). This alleviates the problem of getting information off-chip at speed, but has an additional cost in terms of silicon area and also provides a fixed limit on the amount of trace that can be captured.

The ETB stores the compressed trace information in a circular fashion, continuously capturing trace information until stopped. The size of the ETB varies between chip implementations, but a buffer of 8 or 16kB is typically enough to hold a few thousand lines of program trace.

When a program fails, and the trace buffer is enabled, you can see a portion of program history. With this program history, it is easier to walk back through your program to see what happened just before the point of failure. This is particularly useful for investigating intermittent and real-time failures, which can be difficult to identify through traditional debug methods that require stopping and starting the processor. The use of hardware tracing can significantly reduce the amount of time required to find these failures, because the trace shows exactly what was executed.
5.3.2 Trace Ranges

Trace ranges enable you to restrict the capture of trace to a linear range of memory. A trace range has a start and end address in virtual memory, and any execution within this address range is captured. In contrast to trace start and end points, any function calls made within a trace range are only captured if the target of the function call is also within the specified address range. The number of trace ranges that can be enabled is determined by the debug hardware in your processor.

Trace capture is enabled by default when no trace ranges are set. Trace capture is disabled by default when any trace ranges are set, and is only enabled when within the defined ranges.

You can configure trace ranges using the Ranges tab in the Trace view. The start and end address for each range can either be an absolute address or an expression, such as the name of a function. Be aware that optimizing compilers might rearrange or minimize code in memory from that in the associated source code. This can lead to code being unexpectedly included or excluded from the trace capture.

5.3.3 Trace Points

Trace points enable you to control precisely where in your program trace is captured. Trace points are non-intrusive and do not require stopping the system to process. The maximum number of trace points that can be set is determined by the debug hardware in your processor. The following types of trace points are available:

- **Trace Start Point**  Enables trace capture when execution reaches the selected address.
- **Trace Stop Point**  Disables trace capture when execution reaches the selected address.
- **Trace Trigger Point**  Marks this point in the trace so that you can more easily locate it in the Trace view.

To set trace points in the source view, right-click in the margin and select from the DS-5 Breakpoints context menu. To set trace points in the Disassembly view, right-click on an instruction and select from the context menu. Trace points are listed in the Breakpoints view.

Trace Start Points and Trace Stop Points enable and disable capture of trace respectively. Trace points do not take account of nesting. For example, if you hit two Trace Start Points in a row, followed by two Trace Stop Points, then the trace is disabled immediately when the first Trace Stop Point is reached, not the second. With no Trace Start Points set then trace is enabled all the time by default. If you have any Trace Start Points set, then trace is disabled by default and is only enabled when the first Trace Start Point is hit.

Trace trigger points enable you to mark interesting locations in the trace so that you can easily find them later in the Trace view. Select the Find Trigger Packet option in the view menu to locate trace trigger points. To configure the debugger so that it stops when a trace trigger point is hit, use the Stop On Trigger checkbox in the in the Properties tab of the Trace view. When this is set you can configure the amount of trace that is captured before and after a trace trigger point using the Post-Trigger Capture Size field.

**Note**

Some target platforms do not support the detection of trace trigger points by the trace capture device (ETB/DSTREAM).
5.3.4 See also

Reference

- Trace view on page 10-54
- Export trace report dialog box on page 10-65
- ARM® DS-5™ Setting up the ARM® DSTREAM™ Hardware:
  — Connecting the DSTREAM unit,

Concepts

- About debugging bare-metal symmetric multiprocessing systems on page 6-6.
Chapter 6
Debugging embedded systems

The following topics give an introduction to debugging embedded systems.

Concepts

• About endianness on page 6-2
• About accessing AHB and APB buses on page 6-3
• About virtual and physical memory on page 6-4
• About debugging hypervisors on page 6-5
• About debugging bare-metal symmetric multiprocessing systems on page 6-6
• About debugging multi-threaded applications on page 6-9
• About debugging shared libraries on page 6-11
• About debugging a Linux kernel on page 6-14
• About debugging Linux kernel modules on page 6-16
• About debugging TrustZone enabled targets on page 6-18.
6.1 About endianness

The term endianness is used to describe the ordering of individually addressable quantities, which means bytes and halfwords in the ARM architecture. The term byte-ordering can also be used rather than endian.

If an image is loaded to the target on connection, the debugger automatically selects the endianness of the image otherwise it selects the current endianness of the target. If the debugger detects a conflict then a warning message is generated.

You can use the `set endian` command to modify the default debugger setting.

6.1.1 See also

Reference

- ARM® DS-5™ Debugger Command Reference:
  - `set endian` on page 2-150
  - `show endian` on page 2-177.
6.2 About accessing AHB and APB buses

ARM-based systems connect the processors, memories and peripherals using buses. Examples of common bus types include AMBA High-performance Bus (AHB) and Advanced Peripheral Bus (APB).

In some systems these buses are accessible from the debug interface. Where this is the case then DS-5 Debugger provides access to these buses when performing bare-metal or kernel debugging. Buses are exposed within the debugger as additional address spaces. Accesses to these buses are available when the processor is running.

Within a debug session in DS-5 Debugger you can discover which buses are available using the info memory command. The address and description columns in the output of this command explain what each address space represents and how the debugger accesses it.

You can use AHB: and APB: address prefixes for these buses anywhere in the debugger where you normally enter an address or expression. For example, assuming that the debugger provides an APB address space, then you can print the contents of address zero using the following command:

\[\text{x/1 APB:0x0}\]

The exact topology of the buses and their connection to the debug interface is dependent on your system. See the CoreSight specifications for your hardware for more information. Typically, the debug access to these buses bypass the processor, and so does not take into account memory mappings or caches within the processor itself. It is implementation dependent on whether access to the buses occur before or after any other caches in the system, such as L2 or L3 caches. The debugger does not attempt to achieve coherency between caches in your system when accessing these buses and it is your responsibility to take this into account and manually perform any clean or flush operations as required.

For example, to achieve cache coherency when debugging an image with the processors level 1 cache enabled, you must clean and invalidate portions of the L1 cache prior to modifying any of your application code or data using the AHB address space. This ensures that any existing changes in the cache are written out to memory before writing to that address space, and that the processor correctly reads your modification when execution resumes.

The behavior when accessing unallocated addresses is undefined, and depending on your system can lead to locking up the buses. It is recommended that you only access those specific addresses that are defined in your system. You can use the memory command to redefine the memory regions within the debugger and modifying access rights to control the addresses. You can use the x command with the count option to limit the amount of memory that is read.
6.3 About virtual and physical memory

Processors that contain a Memory Management Unit (MMU) provide two views of memory, virtual and physical. The virtual address is the address prior to address translation in the MMU and the physical address is the address after translation. Normally when the debugger accesses memory, it uses virtual addresses. However, if the MMU is disabled then the mapping is flat and the virtual address is the same as the physical address. To force the debugger to use physical addresses prefix an addresses with P:. For example:

P:0x8000
P:0+main will create a physical address with the address offset of main.

If your processor additionally contains TrustZone technology, then you have access to Secure and Normal worlds, each with their own separate virtual and physical address mappings. In this case, the address prefix P: is not available, and instead you must use NP: for normal physical and SP: for secure physical.

Note

Physical address access is not enabled for all operations. For example, the ARM hardware does not provide support for setting breakpoints via a physical address.

When memory is accessed via a physical address the caches are not flushed. Hence, results might differ depending on whether you view memory via the physical or virtual addresses (assuming they are addressing the same memory addresses).
6.4 About debugging hypervisors

ARM processors that support virtualization extensions have the ability to run multiple guest operating systems beneath a hypervisor. The hypervisor is the software that arbitrates amongst the guest operating systems and controls access to the hardware.

DS-5 Debugger provides basic support for bare-metal hypervisor debugging. When connected to a processor that supports virtualization extensions, the debugger enables you to distinguish between hypervisor and guest memory, and to set breakpoints that only apply when in hypervisor mode or within a specific guest operating system.

A hypervisor typically provides separate address spaces for itself as well as for each guest operating system. Unless informed otherwise, all memory accesses by the debugger occur in the current context. If you are stopped in hypervisor mode then memory accesses use the hypervisor memory space, and if stopped in a guest operating system then memory accesses use the address space of the guest operating system. To force access to a particular address space you must prefix the address with either $H: for hypervisor or $N: for guest operating system. Note that it is only possible to access the address space of the guest operating system that is currently scheduled to run within the hypervisor. It is not possible to specify a different guest operating system.

Similarly, hardware and software breakpoints can be configured to match on hypervisor or guest operating systems using the same address prefixes. If no address prefix is used then the breakpoint applies to the address space that is current when the breakpoint is first set. For example, if a software breakpoint is set in memory that is shared between hypervisor and a guest operating system, then the possibility exists for the breakpoint to be hit from the wrong mode, and in this case the debugger may not recognise your breakpoint as the reason for stopping.

For hardware breakpoints only, not software breakpoints, you can additionally configure them to match only within a specific guest operating system. This feature uses the architecturally defined Virtual Machine ID (VMID) register to spot when a specific guest operating system is executing. The hypervisor is responsible for assigning unique VMIDs to each guest operating system setting this in the VMID register when that guest operating system executes. In using this feature, it is your responsibility to understand which VMID is associated with each guest operating system that you want to debug. Assuming a VMID is known, you can apply a breakpoint to it within the Breakpoints view or by using the break-stop-on-vmid command.

When debugging a system that is running multiple guest operating systems, you can optionally enable the set print current-vmid setting to receive notifications in the console when the debugger stops and the current VMID changes. You can also obtain the VMID within DS-5 scripts using the $vmid debugger variable.

6.4.1 See also

Reference

• ARM® DS-5™ Debugger Command Reference:
  — Using expressions on page 2-4
  — break-stop-on-vmid on page 2-44
  — set print on page 2-154
  — show print on page 2-181.
6.5  About debugging bare-metal symmetric multiprocessing systems

DS-5 Debugger supports debugging bare-metal Symmetric MultiProcessing (SMP) systems. The debugger expects an SMP system to meet the following requirements:

- The same ELF image running on all processors.
- All processors must have identical debug hardware. For example, the number of hardware breakpoint and watchpoint resources must be identical.
- Breakpoints and watchpoints must only be set in regions where all processors have identical memory maps, both physical and virtual. Processors with different instance of identical peripherals mapped at the same address are considered to meet this requirement, as in the case of the private peripherals of ARM multicore processors.

6.5.1  Configuring and connecting

To enable SMP support in the debugger you must first configure a debug session in the Debug Configurations dialog. Targets that support SMP debugging are identified by having SMP mentioned in the Debug operation drop-down list.

Configuring a single SMP connection is all you require to enable SMP support in the debugger. On connection, you can then debug all of the SMP processors in your system by selecting them in the Debug Control view.

6.5.2  Image and symbol loading

When debugging an SMP system, image and symbol loading operations apply to all the SMP processors. For image loading, this means that the image code and data are written to memory once through one of the processors, and are assumed to be accessible through the other processors at the same address because they share the same memory. For symbol loading, this means that debug information is loaded once and is available when debugging any of the processors.

6.5.3  Running, stopping and stepping

When debugging an SMP system, attempting to run one processor automatically starts running all the other processors in the system. Similarly, when one processor stops (either because you requested it or because of an event such as a breakpoint being hit), then all processors in the system stop.

For instruction level single-stepping (\texttt{stepi} and \texttt{nexti} commands), then the currently selected processor steps one instruction. The exception to this is when a \texttt{nexti} operation is required to step over a function call in which case the debugger sets a breakpoint and then runs all processors. All other stepping commands affect all processors.

Depending on your system, there might be a delay between one processor running or stopping and another processor running or stopping. This delay can be very large because the debugger must manually run and stop all the processors individually.

In rare cases, one processor might stop and one or more of the others fails to stop in response. This can occur, for example, when a processor running code in secure mode has temporarily disabled debug ability. When this occurs, the Debug Control view displays the individual state of each processor (running or stopped), so that you can see which ones have failed to stop. Subsequent run and step operations might not operate correctly until all the processors stop.
6.5.4 Breakpoints, watchpoints, and signals

By default, when debugging an SMP system, breakpoint, watchpoint, and signal (vector catch) operations apply to all processors. This means that you can set one breakpoint to trigger when any of the processors execute code that meets the criteria. When the debugger stops due to a breakpoint, watchpoint, or signal, then the processor that causes the event is listed in the Commands view.

Breakpoints or watchpoints can be configured for one or more processors by selecting the required processor in the relevant Properties dialog box. Alternatively, you can use the break-stop-on-cores command. This feature is not available for signals.

6.5.5 Examining target state

Views of the target state, including registers, call stack, memory, disassembly, expressions, and variables contain content that is specific to a processor.

Views such as breakpoints, signals and commands are shared by all the processors in the SMP system, and display the same contents regardless of which processor is currently selected.

6.5.6 Trace

When you are using a connection that enables trace support then you are able to view trace for each of the processors in your system. By default, the Trace view shows trace for the processor that is currently selected in the Debug Control view. Alternatively, you can choose to link a Trace view to a specific processor by using the Linked: context toolbar option for that Trace view. Creating multiple Trace views linked to specific processors enables you to view the trace from multiple processors at the same time. The indexes in the Trace views do not necessarily represent the same point in time for different processors.

6.5.7 See also

Tasks
- Setting an execution breakpoint on page 4-10
- Setting a data watchpoint on page 4-12
- Setting a conditional breakpoint on page 4-15
- Setting a breakpoint on a specific thread on page 4-18
- Pending breakpoints and watchpoints on page 4-20
- Stepping through an application on page 4-26
- Handling Unix signals on page 4-28
- Handling processor exceptions on page 4-30
- Examining the target execution environment on page 5-2
- Examining the call stack on page 5-4.

Concepts
- About breakpoints and watchpoints on page 4-7
- About trace support on page 5-6
- About debugging multi-threaded applications on page 6-9.

Reference
- Breakpoints view on page 10-8
- Commands view on page 10-17
- Debug Control view on page 10-20
• Disassembly view on page 10-25
• Memory view on page 10-37
• Modules view on page 10-41
• Registers view on page 10-44
• Variables view on page 10-58.

• ARM® DS-5™ Debugger Command Reference:
  — break on page 2-37
  — break-stop-on-threads, break-stop-on-cores on page 2-42
  — info cores on page 2-83
  — info threads on page 2-104
  — thread, core on page 2-205.
6.6 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).

Thread 1086 #1 stopped (PID 1086)

where #1 is the unique ID used by the debugger and PID 1086 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

6.6.1 See also

Tasks
- Setting a breakpoint on a specific thread on page 4-18
- Setting an execution breakpoint on page 4-10
- Setting a data watchpoint on page 4-12
- Setting a conditional breakpoint on page 4-15
- Pending breakpoints and watchpoints on page 4-20
- Stepping through an application on page 4-26
- Handling Unix signals on page 4-28
- Handling processor exceptions on page 4-30
- Examining the target execution environment on page 5-2
- Examining the call stack on page 5-4.

Concepts
- About debugging bare-metal symmetric multiprocessing systems on page 6-6.

Reference
- Breakpoints view on page 10-8
- Commands view on page 10-17
- Debug Control view on page 10-20
- Disassembly view on page 10-25
- Memory view on page 10-37
- Modules view on page 10-41
• Registers view on page 10-44
• Variables view on page 10-58.
• ARM® DS-5™ Debugger Command Reference:
  — break on page 2-37
  — break-stop-on-threads, break-stop-on-cores on page 2-42
  — info threads on page 2-104
  — thread, core on page 2-205.
6.7 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 require 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, and 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 in the **Files** tab.

![Debug Configurations dialog box](image)

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

### See also

#### Tasks

- Configuring a connection to a Linux target using gdbserver on page 3-5
- Setting an execution breakpoint on page 4-10
- Setting a data watchpoint on page 4-12
- Setting a conditional breakpoint on page 4-15
- Pending breakpoints and watchpoints on page 4-20
- Stepping through an application on page 4-26
- Handling Unix signals on page 4-28
- Examining the target execution environment on page 5-2
- Examining the call stack on page 5-4
- ARM® DS-5™ Getting Started with DS-5:
  — Debugging Gnometris on page 3-18.
- ARM® DS-5™ Using Eclipse:
  — Accessing the dynamic help on page 2-38.
Concepts

• About breakpoints and watchpoints on page 4-7.

Reference

• Breakpoints view on page 10-8
• Commands view on page 10-17
• Debug Control view on page 10-20
• Disassembly view on page 10-25
• Memory view on page 10-37
• Modules view on page 10-41
• Registers view on page 10-44
• Variables view on page 10-58
• Debug Configurations - Files tab on page 10-81
• Debug Configurations - Debugger tab on page 10-86.

• ARM® DS-5™ Debugger Command Reference:
  — add-symbol-file on page 2-30
  — info sharedlibrary on page 2-98
  — nosharedlibrary on page 2-126
  — set auto-solib-add on page 2-144
  — set solib-search-path on page 2-159
  — set stop-on-solib-events on page 2-161
  — set sysroot, set solib-absolute-prefix on page 2-163
  — sharedlibrary on page 2-166.
6.8 About debugging a Linux kernel

DS-5 supports source level debugging of a Linux kernel. The Linux kernel (and associated device drivers) can be debugged in the same way as a standard ELF format executable. For example, you can set breakpoints in the kernel code, step through the source, inspect the call stack, and watch variables.

Note
User space parameters (marked __user) that are not currently mapped in cannot be read by the debugger.

To debug the kernel:

1. Compile the kernel source using the following options:
   - CONFIG_DEBUG_KERNEL=y
   - CONFIG_DEBUG_INFO=y
   - Other options might be required depending on the type of debugging you want to perform.

   Compiling the kernel source generates a Linux kernel image and symbol files containing debug information.

   Note
   Be aware that a Linux kernel is always compiled with full optimizations and inlining enabled, therefore:
   - stepping through code might not work as expected due to the possible reordering of some instructions
   - some variables might be optimized out by the compiler and therefore not be available for the debugger.

2. Load the Linux kernel on to the target
3. Load kernel debug information into the debugger
4. Debug the kernel as required.

6.8.1 See also

Tasks
- Configuring a connection to a Linux Kernel on page 3-8
- Setting a breakpoint on a specific thread on page 4-18
- Setting an execution breakpoint on page 4-10
- Setting a data watchpoint on page 4-12
- Setting a conditional breakpoint on page 4-15
- Pending breakpoints and watchpoints on page 4-20
- Stepping through an application on page 4-26
- Handling Unix signals on page 4-28
- Handling processor exceptions on page 4-30
- Examining the target execution environment on page 5-2
- Examining the call stack on page 5-4
- ARM® DS-5™ Getting Started with DS-5:
  — Debugging a loadable kernel module on page 3-19.
Concepts

- About debugging Linux kernel modules on page 6-16
- ARM Linux problems and solutions on page 11-2
- Target connection problems and solutions on page 11-4.

Reference

- Breakpoints view on page 10-8
- Commands view on page 10-17
- Debug Control view on page 10-20
- Disassembly view on page 10-25
- Memory view on page 10-37
- Modules view on page 10-41
- Registers view on page 10-44
- Variables view on page 10-58.

- ARM® DS-5™ Debugger Command Reference:
  - break on page 2-37
  - break-stop-on-threads, break-stop-on-cores on page 2-42
  - info threads on page 2-104
  - thread, core on page 2-205.
6.9 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 having 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 require debug information.

6.9.1 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 (statically linked) 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.

6.9.2 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 (OS) support enabled within the debugger before a loadable module can be detected. OS support is automatically enabled when a Linux kernel image is loaded into the debugger. However, you can manually control this by using the set os command.

6.9.3 See also

Tasks
• Configuring a connection to a Linux Kernel on page 3-8
• ARM® DS-5™ Getting Started with DS-5:
  — Debugging a loadable kernel module on page 3-19.
Concepts

• About debugging a Linux kernel on page 6-14.

Reference

• Breakpoints view on page 10-8
• Commands view on page 10-17
• Debug Control view on page 10-20
• Disassembly view on page 10-25
• Memory view on page 10-37
• Modules view on page 10-41
• Registers view on page 10-44
• Variables view on page 10-58
• Debug Configurations - Files tab on page 10-81
• Debug Configurations - Debugger tab on page 10-86
• ARM® DS-5™ Debugger Command Reference:
  — info os-log on page 2-92
  — info os-modules on page 2-93
  — info os-version on page 2-94
  — info processes on page 2-95
  — set os on page 2-153
  — show os on page 2-180.
6.10 About debugging TrustZone enabled targets

ARM TrustZone® is a security technology designed into some ARM processors. For example, the Cortex™-A class processors. It segments execution and resources such as memory and peripherals into secure and normal worlds.

When connected to a target that supports TrustZone and where access to the secure world is permitted, then the debugger provides access to both secure and normal worlds. In this case, all addresses and address-related operations are specific to a single world. This means that any commands you use that require an address or expression must also specify the world that they apply to. 

\[ S:0x1000 \]

Where:

\( N: \) For an address in Normal World memory

\( S: \) For an address in Secure World memory.

If you want to specify an address in the current world, then you can omit the prefix.

When loading images and debug information it is important that you load them into the correct world. The debug launcher panel does not provide a way to directly specify an address world for images and debug information, so to achieve this you must use scripting commands instead. The Debugger tab in the debugger launcher panel provides an option to run a debug initialization script or a set of arbitrary debugger commands on connection. Here are some example commands:

- Load image only to normal world (applying zero offset to addresses in the image)
  
  \[
  \text{load myimage.axf N:0}
  \]

- Load debug information only to secure world (applying zero offset to addresses in the debug information)
  
  \[
  \text{file myimage.axf S:0}
  \]

- Load image and debug information to secure world (applying zero offset to addresses)
  
  \[
  \text{loadfile myimage.axf S:0}
  \]

When an operation such as loading debug symbols or setting a breakpoint needs to apply to both normal and secure worlds then you must perform the operation twice, once for the normal world and once for the secure world.

Registers such as \$PC have no world. To access the content of memory from an address in a register that is not in the current world, you can use an expression, \( N:0+\$PC \). This is generally not necessary for expressions involving debug information, because these are associated with a world when they are loaded.

6.10.1 See also

Reference

- Breakpoints view on page 10-8
- ARM® DS-5™ Debugger Command Reference:
  
  — add-symbol-file on page 2-30
  — file, symbol-file on page 2-68
  — load on page 2-111
  — loadfile on page 2-112.
• Technical Reference Manual,
• Architecture Reference Manual,
Chapter 7
Debugging with scripts

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

Tasks
- Exporting DS-5 Debugger commands generated during a debug session on page 7-2
- Creating a DS-5 Debugger script on page 7-3
- Creating a Jython script on page 7-4
- Creating a CMM-style script on page 7-7
- Running a script on page 7-8
- Configuring a Jython project in Eclipse on page 7-9.
7.1 Exporting DS-5 Debugger commands generated during a debug session

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 Commands generated during a debug session](image)

7.1.1 See also

Tasks
- Launching the debugger from Eclipse on page 2-6
- Running a script on page 7-8.

Reference
- Commands view on page 10-17
- History view on page 10-35
- Scripts view on page 10-50
- ARM® DS-5™ Debugger Command Reference:
  — Chapter 2 DS-5 Debugger commands
  — Chapter 3 CMM-style commands supported by the debugger.
7.2 Creating a DS-5 Debugger script

The script file must contain only one command on each line. Each command can be identified with comments if required. The .ds file extension must be used to identify this type of script.

Example 7-1 DS-5 Debugger script

```plaintext
# Filename: myScript.ds

# 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
```

7.2.1 See also

Tasks
- Launching the debugger from Eclipse on page 2-6
- Running a script on page 7-8.

Reference
- Scripts view on page 10-50
- ARM® DS-5™ Debugger Command Reference:
  — Chapter 2 DS-5 Debugger commands.
7.3 Creating a Jython script

Jython is a Java implementation of the Python scripting language. It provides extensive support for data types, conditional execution, loops and organisation of code into functions, classes and modules, as well as access to the standard Jython libraries. Jython is an ideal choice for larger or more complex scripts.

These are important concepts that are required in order to write a debugger Jython script.

**Imports**

The debugger module provides a Debugger class for initial access to the DS-5 Debugger, with further classes, known as services, to access registers and memory. Here is an example showing the full set of module imports that are typically placed at the top of the jython script:

```python
from arm_ds.debugger_v1 import Debugger
from arm_ds.debugger_v1 import DebugException
```

**Execution Contexts**

Most operations on DS-5 Debugger Jython interfaces require an execution context. The execution context represents the state of the target system. Separate execution contexts exist for each process, thread, or processor that is accessible in the debugger. You can obtain an execution context from the Debugger class instance, for example:

```python
# Obtain the first execution context
debugger = Debugger()
ec = debugger.getExecutionContext(0)
```

**Registers**

You can access processor registers, coprocessor registers and peripheral registers using the debugger Jython interface. To access a register you must know its name. The name can be obtained from the Registers view in the graphical debugger. The RegisterService enables you to read and write register values, for a given execution context, for example:

```python
# Print the Program Counter (PC) from execution context ec
value = ec.getRegisterService().getValue('PC')
print 'The PC is %s' %value
```

**Memory**

You can access memory using the debugger Jython interface. You must specify an address and the number of bytes to access. The address and size can be an absolute numeric value or a string containing an expression to be evaluated within the specified execution context. Here is an example:

```python
# Print 16 bytes at address 0x0 from execution context ec
print ec.getMemoryService().read(0x0, 16)
```

**DS Commands**

The debugger jython interface enables you to execute arbitrary DS-5 commands. This is useful when the required functionality is not directly provided in the Jython interface. You must specify the execution context, the command and any arguments that you want to execute. The return value includes the textual output from the command and any errors. Here is an example:

```python
# Execute the DS-5 command 'print $ENTRYPOINT' and print the result
print ec.executeDSCommand('print $ENTRYPOINT')
```

**Error Handling**

The methods on the debugger Jython interfaces throw DebugException whenever an error occurs. You can catch exceptions to handle errors in order to provide more information. Here is an example:
# Catch a DebugException and print the error message
try:
    ec.getRegisterService().getValue('ThisRegisterDoesNotExist')
except DebugException, de:
    print "Caught DebugException: %s" % (de.getMessage())

The .py file extension must be used to identify this type of script.

Example 7-2 Jython script

```python
# File name: myScript.py
import sys

from arm_ds.debugger_v1 import Debugger
from arm_ds.debugger_v1 import DebugException

# Debugger object for accessing the debugger
debugger = Debugger()

# Initialisation commands
debugger = debugger.getExecutionContext(0)
debugger.getExecutionService().stop()
debugger.getExecutionService().waitForStop()
# in case the execution context reference is out of date
debugger = debugger.getExecutionContext(0)
# load image if provided in script arguments
if len(sys.argv) == 2:
    image = sys.argv[1]
    debugger.getImageService().loadImage(image)
    debugger.getExecutionService().setExecutionAddressToEntryPoint()
    debugger.getImageService().loadSymbols(image)

    # we can use all the DS commands available
    print "Entry point: ",
    print debug.debugger.executeDSCommand("print $ENTRYPOINT")
    # Sample output:
    # Entry point: $8 = 32768
else:
    pass # assuming image and symbols are loaded

# sets a temporary breakpoint at main and resumes
debugger.getExecutionService().resumeTo("main") # this method is non-blocking
try:
    debugger.getExecutionService().waitForStop(500) # wait for 500ms
except DebugException, e:
    if e.getErrorCode() == "JYI31": # code of "Wait for stop timed out" message
        print "Waiting timed out!"
        sys.exit()
    else:
        raise # re-raise the exception if it is a different error
debugger = debugger.getExecutionContext(0)

def getRegisterValue(executionContext, name):
    """Get register value and return string with unsigned hex and signed integer, possibly string "error" if there was a problem reading the register."
    """
    try:
        value = executionContext.getRegisterService().getValue(name)
        # the returned value behaves like a numeric type,
```
# and even can be accessed like an array of bytes, e.g. 'print value[:]' 
return "%s (%d)" % (str(value), int(value))

except DebugException, e:
    return "error"

# print Core registers on all execution contexts 
for i in range(debugger.getExecutionContextCount()): 
    ec = debugger.getExecutionContext(i) 
    # filter register names starting with "Core:" 
    coreRegisterNames = filter(lambda name: name.startswith("Core:"), 
                                ec.getRegisterService().getRegisterNames()) 
    # using Jython list comprehension get values of all these registers 
    registerInfo = ["%s = %s" % (name, getRegisterValue(ec, name)) 
                    for name in coreRegisterNames] 
    registerInfo = ", ".join(registerInfo[:3]) 
    print "Identifier: %s, Registers: %s" % (ec.getIdentifier(), registers) 
# Output: 
# Identifier: 1, Registers: Core::R0 = 0x00000010 (16), Core::R1 = 0x00000000 (0), Core::R2 = 0x0000A4A4 (42148) 
# ... 

7.3.1  See also

Tasks
•  Launching the debugger from Eclipse on page 2-6 
•  Running a script on page 7-8 
•  Configuring a Jython project in Eclipse on page 7-9.

Reference
•  Scripts view on page 10-50
•  ARM® DS-5™ Debugger Command Reference: 
    —  Chapter 2 DS-5 Debugger commands.
### 7.4 Creating a CMM-style script

The script file must contain only one command on each line. Each command can be identified with comments if required. The `.cmm` or `.t32` file extension must be used to identify this type of script.

**Example 7-3 CMM-style script**

```cmm
// Filename: myScript.cmm
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
```

### 7.4.1 See also

**Tasks**
- *Launching the debugger from Eclipse* on page 2-6
- *Running a script* on page 7-8.

**Reference**
- *Scripts view* on page 10-50
- *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*.
7.5 **Running a script**

To run a script from Eclipse:

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](image)

Alternatively you can use the DS-5 Debugger source command.

### 7.5.1 See also

**Tasks**
- *Launching the debugger from Eclipse* on page 2-6
- *Exporting DS-5 Debugger commands generated during a debug session* on page 7-2
- *Creating a Jython script* on page 7-4
- *Creating a DS-5 Debugger script* on page 7-3
- *Creating a CMM-style script* on page 7-7
- *Configuring a Jython project in Eclipse* on page 7-9.

**Reference**
- *Scripts view* on page 10-50
- Chapter 3 *Configuring and connecting to a target*
- *Debug Configurations - Debugger tab* on page 10-86
- *ARM® DS-5™ Debugger Command Reference:*
  — *source* on page 2-193.
7.6 Configuring a Jython project in Eclipse

You can use Eclipse to create or edit Jython scripts. Eclipse for DS-5 provides auto-completion features and built-in documentation for Jython.

To configure a Jython project:

1. Create a new project:
   a. Select **File → New → Project...** from the main menu.
   b. Expand the **PyDev** group.
   c. Select **PyDev Project**.
   d. Click **Next**.

   ![Figure 7-3 PyDev project wizard](image)

   e. Enter a suitable name for the project.
   f. Select **Jython** as the project type.
   g. Select **DS-5 Jython** as the interpreter.
   h. Click **Finish** to create the project.
If this is the first time you have created a PyDev project then a dialog box appears informing you that the Python interpreter is not configured. Jython, however, is configured so you can ignore this message. Click Don’t ask again to proceed.

Alternatively you can configure an existing project for Jython:

a. Right-click on the project in the Project Explorer view and select PyDev → Set as PyDev Project from the context menu.

b. Click Don’t ask again if the Python not configured dialog box appears.

c. Select Properties from the Project menu.

d. Select PyDev - Interpreter/Grammar from the Properties dialog box.

e. Select Jython as the project type.

f. Select DS-5 Jython as the interpreter.

g. Click OK to apply these settings and close the dialog box.

Add a Python source file to the project. The .py file extension must be used to identify this type of script.

7.6.1 See also

Tasks

• Launching the debugger from Eclipse on page 2-6
• Creating a Jython script on page 7-4
• Running a script on page 7-8
• ARM® DS-5™ Using Eclipse:
  — Creating a new C or C++ project on page 3-4
  — Importing an existing Eclipse project on page 3-8
  — Adding a new source file to your project on page 3-16.

Reference
• Scripts view on page 10-50
• ARM® DS-5™ Debugger Command Reference:
  — Chapter 2 DS-5 Debugger commands.
The following topics describe semihosting and how to control runtime messages.

Tasks

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

Concepts

- About semihosting and top of memory on page 8-2
- About Log4j configuration files on page 8-8.
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.

![Diagram of memory layout](image)

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 10-3
- *Debug Configurations - Connection tab* on page 10-76
- *ARM® DS-5™ Debugger Command Reference:*
  - *set semihosting* on page 2-156
show semihosting on page 2-182.
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 10-3
• ARM® DS-5™ Debugger Command Reference:
  — set semihosting on page 2-156
  — show semihosting on page 2-182.
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.

Note

Creating a special semihosting function is not required if you build your application image using ARM Compiler 5.0 and later. The linker automatically adds a semihosting symbol if required.

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

```c
#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 10-3
- ARM® DS-5™ Debugger Command Reference:
  - set semihosting on page 2-156
  - show semihosting on page 2-182.
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:

\texttt{log config=\textit{option}}

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

Where:

\texttt{option} Specifies a predefined logging configuration or a user-defined logging configuration file:

\texttt{info} Output messages using the predefined INFO level configuration. This is the default.

\texttt{debug} Output messages using the predefined DEBUG level configuration.

\texttt{filename} Specifies a user-defined logging configuration file to customize the output of messages. The debugger supports log4j configuration files.

\texttt{log file=\textit{filename}}

Output messages to a file in addition to the console.

8.5.1 See also

\textbf{Tasks}

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

\textbf{Reference}

- Commands view on page 10-17
- ARM® DS-5™ Debugger Command Reference:
  - \texttt{log config} on page 2-113
  - \texttt{log file} on page 2-114.

\textbf{Other information}

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>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logger</td>
<td>Defines the level of logging.</td>
</tr>
<tr>
<td>Appender</td>
<td>Defines the output destination.</td>
</tr>
<tr>
<td>Layout</td>
<td>Defines the message format.</td>
</tr>
</tbody>
</table>

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
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, 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

```java
# 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

```java
# 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

```java
# 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
```
Controlling runtime messages

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_access.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 10-17
•  ARM® DS-5™ Debugger Command Reference:
  —  log config on page 2-113
  —  log file on page 2-114.

Other information
Chapter 9
Working with the Snapshot Viewer

The following topics describe how to use the Snapshot Viewer.

Tasks
• Creating a Snapshot Viewer initialization file on page 9-2
• Connecting to the Snapshot Viewer on page 9-7.

Concepts
• About the Snapshot Viewer on page 9-5
• Considerations when creating debugger scripts for the Snapshot Viewer on page 9-8.
9.1 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.

9.1.1 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.

9.1.2 Procedure

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

[globals] 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),
    enabling 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 are only required 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.

9.1.3 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.

9.1.4 Example

Example 9-1 Snapshot Viewer Initialization file

; All sections are optional

[global]
core=Cortex-M3 ; Selected processor

[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)

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)

[regs]
R0=0x000080C8
R1=0x0000C000
R2=0x00007C000
R3=0x00007C000
R4=0x00000363
R5=0x00008EEC
R6=0x00000000
R7=0x00000000
R8=0x00000000
R9=0x83532737
R10=0x000008DE8
R11=0x00000000
R12=0x00000000
9.1.5 See also

**Tasks**
- *Launching the debugger from Eclipse* on page 2-6
- *Connecting to the Snapshot Viewer* on page 9-7
- *Considerations when creating debugger scripts for the Snapshot Viewer* on page 9-8.

**Concepts**
- *About the Snapshot Viewer* on page 9-5.

**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
9.2  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.

9.2.1  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 Registers (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, 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 (Sn)
• Double precision registers (Dn)
• Quad precision registers (Qn).

9.2.2  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, you must 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 (\texttt{extern} or file \texttt{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. 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.

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 \texttt{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.

\subsection*{9.2.3 Debug symbols}

The debugger require debug information 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.

\subsection*{9.2.4 See also}

\textbf{Tasks}

- \textit{Connecting to the Snapshot Viewer} on page 9-7
- \textit{Creating a Snapshot Viewer initialization file} on page 9-2
- \textit{Considerations when creating debugger scripts for the Snapshot Viewer} on page 9-8.
9.3 **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.

### 9.3.1 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.

### 9.3.2 Procedure

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

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

### 9.3.3 See also

- **Tasks**
  - *Launching the debugger from Eclipse on page 2-6*
  - *Creating a Snapshot Viewer initialization file on page 9-2*
  - *Considerations when creating debugger scripts for the Snapshot Viewer on page 9-8.*

- **Concepts**
  - *About the Snapshot Viewer on page 9-5.*

- **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.*
9.4 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 9-2 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 "Value of *(d_array)+9] is " *(d_array)+9]
var.print "Value 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
```

9.4.1 See also

Tasks
- Launching the debugger from Eclipse on page 2-6
- Connecting to the Snapshot Viewer on page 9-7
- Creating a Snapshot Viewer initialization file on page 9-2.
Concepts

• About the Snapshot Viewer on page 9-5.

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 10  
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
- App Console view on page 10-3
- ARM Asm Info view on page 10-5
- ARM assembler editor on page 10-6
- Breakpoints view on page 10-8
- C/C++ editor on page 10-12
- Channel editor for the Event Viewer on page 10-15
- Commands view on page 10-17
- Debug Control view on page 10-20
- Disassembly view on page 10-25
- Expressions view on page 10-29
- Functions view on page 10-32
- History view on page 10-35
- Memory view on page 10-37
- Modules view on page 10-41
- Registers view on page 10-44
- Screen view on page 10-47
- Scripts view on page 10-50
- Target view on page 10-52
- Trace view on page 10-54
- Variables view on page 10-58
• Export memory dialog box on page 10-61
• Import memory dialog box on page 10-63
• Export trace report dialog box on page 10-65
• Breakpoint properties dialog box on page 10-67
• Watchpoint properties dialog box on page 10-71
• Tracepoint properties dialog box on page 10-72
• Manage Signals dialog box on page 10-73
• Debug Configurations - Connection tab on page 10-76
• Debug Configurations - Files tab on page 10-81
• Debug Configurations - Debugger tab on page 10-86
• Debug Configurations - Arguments tab on page 10-90
• Debug Configurations - Environment tab on page 10-93
• Debug Configurations - Event Viewer tab on page 10-96
• DS-5 Debugger menu and toolbar icons on page 10-99
• ARM® DS-5™ Using Eclipse:
  — Remote Systems view on page 5-3
  — Remote System Details view on page 5-4
  — Remote Scratchpad view on page 5-5
  — Terminals view on page 5-6.
10.1 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 must be enabled in the debugger.

![App Console view](image)

**Figure 10-1 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.

10.1.1 Toolbar and context menu options

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

**Linked:** **context**

Links this view to the selected connection in the Debug Control view. This is the default. Alternatively you can link the view to a specific connection. If the connection you want is not shown in the drop-down list you might have to select it first in the Debug Control view.

**Save Console Buffer**

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

**Clear Console**

Clears the contents of the Semihosting Console view.

**Scroll Lock** Enables or disables the automatic scrolling of messages in the Semihosting Console view.

**View Menu** This menu contains the following option:

**New App ConsoleView**

Displays a new instance of the App Console view.
Bring to Front for Write
If enabled, the debugger automatically changes the focus to this view when a Semihosting application prompts for input.

Copy
Copies the selected text.

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

Select All
Selects all text.

10.1.2 See also

Tasks
• Working with semihosting on page 8-4
• Enabling automatic semihosting support in the debugger on page 8-5
• ARM DS-5 Using Eclipse:
  — Accessing the dynamic help on page 2-38.

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

Reference
• Target view on page 10-52
• DS-5 Debugger menu and toolbar icons on page 10-99
• ARM® DS-5™ Using Eclipse:
  — Perspectives and views on page 2-22.
10.2  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.

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.

10.2.1 See also

Tasks
• ARM® DS-5™ Using Eclipse:
   — Accessing the dynamic help on page 2-38.

Reference
• ARM assembler editor on page 10-6
• DS-5 Debugger menu and toolbar icons on page 10-99
• ARM® DS-5™ Using Eclipse:
   — Perspectives and views on page 2-22.
10.3 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 10-3 ARM assembler editor](image)

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.

10.3.1 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 ARM assembler editor. The options available include:

**DS-5 Breakpoints menu**

The following breakpoint options are available:

**Toggle Breakpoint**

Adds a new breakpoint, or remove a selected breakpoint.

**Disable Breakpoint, Enable Breakpoint**

Disables or enables the selected breakpoint.
Breakpoint Properties...
Displays the Breakpoint Properties dialog box for the selected breakpoint. This enables you to control breakpoint activation.

Toggle Trace Start Point
Sets or removes a trace start point at the selected address.

Toggle Trace Stop Point
Sets or removes a trace stop point at the selected address.

Toggle Trace Trigger Point
Starts a trace trigger point at the selected address.

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.

10.3.2 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 2-28
  - Accessing the dynamic help on page 2-38.

Reference

- ARM Asm Info view on page 10-5
- DS-5 Debugger menu and toolbar icons on page 10-99
- ARM® DS-5™ Using Eclipse:
  - Perspectives and views on page 2-22.
10.4 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.

![Breakpoints view](image)

10.4.1 Syntax of a breakpoint entry

A breakpoint entry has the following syntax:

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

where:

**source_file:linenum**

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

**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.

**address**

The address where the breakpoint is set.

**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()+132 sub-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, (i=3).

10.4.2 Syntax of a watchpoint entry
A watchpoint entry has the following syntax:
name type[#ID]
where:
name The name of the variable where the watchpoint is set.
type The access type of the watchpoint.
ID The watchpoint ID number.

10.4.3 Syntax of a tracepoint entry
A tracepoint entry has the following syntax:
source_file:linenum address
where:
address The address where the tracepoint is set.
source_file:linenum
If the source file is available, the file name and line number in the file where the tracepoint is set, Fireworks.c:529 0x80000A72.

10.4.4 Toolbar and context menu options
The following options are available from the toolbar or context menu:
Linked: context
Links this view to the selected connection in the Debug Control view. This is the default. Alternatively you can link the view to a specific connection. If the connection you want is not shown in the drop-down list you might have to select it first in the Debug Control view.

Remove Removes the selected breakpoints and watchpoints.
Remove All Removes all breakpoints and watchpoints.
Go to File Displays the source file containing the line of code where the selected breakpoint is set. This option is disabled for a watchpoint.
Go to Disassembly
Displays the disassembly where the selected breakpoint is set. This option is disabled for a watchpoint.

Go to Memory
Displays the memory where the selected watchpoint is set. This option is disabled for a breakpoint.

Skip All Breakpoints
Deactivates 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
Enables the selected breakpoints and watchpoints.

Disable Breakpoints
Disables the selected breakpoints and watchpoints.

Resolve
Re-evaluates 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...
Displays the Properties dialog box for the selected breakpoint, watchpoint or tracepoint. This enables you to control activation or change the access type for the selected watchpoint.

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

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

View Menu
The following View Menu options are available:

New Breakpoints View
Displays a new instance of the Breakpoints view.

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

Import Breakpoints
Imports a list of breakpoints and watchpoints from a file.

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

Ordered Sort
Sorts the list in the order they have been set.

Manage Signals
Displays the Manage Signal dialog box.
10.4.5 See also

Tasks
- Setting an execution breakpoint on page 4-10
- Setting a data watchpoint on page 4-12
- Setting a tracepoint on page 4-14
- Setting a conditional breakpoint on page 4-15
- Pending breakpoints and watchpoints on page 4-20
- Handling Unix signals on page 4-28
- Handling processor exceptions on page 4-30
- 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 2-38.

Reference
- ARM assembler editor on page 10-6
- C/C++ editor on page 10-12
- Disassembly view on page 10-25
- Memory view on page 10-37
- Breakpoint properties dialog box on page 10-67
- Watchpoint properties dialog box on page 10-71
- Tracepoint properties dialog box on page 10-72
- Manage Signals dialog box on page 10-73
- DS-5 Debugger menu and toolbar icons on page 10-99
- ARM® DS-5™ Debugger Command Reference:
  — awatch on page 2-34
  — clear on page 2-46
  — delete breakpoints on page 2-53
  — disable breakpoints on page 2-56
  — enable breakpoints on page 2-64
  — ignore on page 2-77
  — resolve on page 2-136
  — rwatch on page 2-139
  — set breakpoint on page 2-147
  — set substitute-path on page 2-162
  — watch on page 2-212.
- ARM® DS-5™ Using Eclipse:
  — Perspectives and views on page 2-22.
10.5 C/C++ editor

This editor enables you to:

- Edit source code.
- View the syntax highlighting.
- View interactive help when hovering over C library functions. For example, `printf()`.
- Use content assist (`Ctrl+Space`) for auto-completion.
- Create, delete, enable or disable a breakpoint.

---

**Figure 10-5 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.

10.5.1 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**  Sets or removes a breakpoint at the selected address.
- **Toggle Hardware Breakpoint**  Sets or removes a hardware breakpoint at the selected address.
- **Resolve Breakpoint**  Resolves a pending breakpoint at the selected address.
- **Enable Breakpoint**  Enables the breakpoint at the selected address.
- **Disable Breakpoint**  Disables the breakpoint at the selected address.
Breakpoint Properties...
Displays the Breakpoint Properties dialog box for the selected breakpoint. This enables you to control breakpoint activation.

Toggle Trace Start Point
Sets or removes a trace start point at the selected address.

Toggle Trace Stop Point
Sets or removes a trace stop point at the selected address.

Toggle Trace Trigger Point
Starts a trace trigger point at the selected address.

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
Shows or hides line numbers.

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

10.5.2 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
Sets the PC to the address of the selected source line.

Run to Selection
Runs to the selected source line.

Show in Disassembly
This option:
2. Highlights the addresses and instructions associated with the selected source line. A vertical bar and shaded highlight shows the related disassembly.
10.5.3 See also

Tasks
- Setting an execution breakpoint on page 4-10
- Setting a conditional breakpoint on page 4-15
- Setting a breakpoint on a specific thread on page 4-18
- Configuring the debugger path substitution rules on page 4-32
- 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 2-28
  - Accessing the dynamic help on page 2-38.

Concepts
- ARM® DS-5™ Using Eclipse:
  - Overview of the C/C++ editor on page 4-2.

Reference
- Disassembly view on page 10-25
- DS-5 Debugger menu and toolbar icons on page 10-99
- ARM® DS-5™ Debugger Command Reference:
  - set substitute-path on page 2-162.
- ARM® DS-5™ Using Eclipse:
  - Perspectives and views on page 2-22.
10.6 Channel editor for the Event Viewer

This editor enables you to view the output generated by the Event Viewer.

Note

This editor is enabled when you connect to a target where Instrumentation Trace Macrocell (ITM) capture is supported.

On starting a debug session with the Event Viewer enabled, a Channel editor tab opens for each ITM channel that is enabled. The Event Viewer displays textual logs from your application in order of creation (newer entries are added to the bottom of the view). If timestamps are available then this is displayed in an additional column, next to each log entry.

Data is captured from your application only when it runs. However, no data appears in the view until you stop the application. You can stop the target by clicking the Interrupt icon in the Debug Control view, or by entering the stop command in the Commands view. When your application stops then any captured logging information is automatically appended to the open views.

To cease capturing logging information, close the Channel editor tabs, and if you have finished debugging, then disconnect from the target by clicking on the Disconnect from Target icon in the Debug Control view.
Figure 10-7 Channel editor for the Event Viewer

10.6.1 See also

Tasks

- ARM® DS-5™ Using Eclipse:
  - Accessing the dynamic help on page 2-38.

Reference

- Configuring an Event Viewer connection to a bare-metal target on page 3-12
- Debug Configurations - Event Viewer tab on page 10-96
- DS-5 Debugger menu and toolbar icons on page 10-99
- ARM® DS-5™ Using Eclipse:
  - Perspectives and views on page 2-22.
10.7 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.

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 \texttt{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 \texttt{pr} followed by the keyboard combination \texttt{Ctrl+Space} to search for the print command.

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

\begin{figure}
\centering
\includegraphics[width=\textwidth]{commands_view.png}
\caption{Commands view}
\end{figure}

\begin{itemize}
\item \textbf{Linked:} context
\begin{itemize}
\item Links this view to the selected connection in the Debug Control view. This is the default. Alternatively you can link the view to a specific connection. If the connection you want is not shown in the drop-down list you might have to select it first in the Debug Control view.
\end{itemize}
\end{itemize}
Save Console Buffer
Saves the contents of the Commands view to a text file.

Clear Console
Clears the contents of the Commands view.

Scroll Lock
Enables or disables 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...
Displays the Open dialog box to select and run a script file.

Organize Favorites...
Displays the Scripts view, where you can organize your scripts.

Show Command History View
Displays the History view.

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

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

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

Save the selected lines as a script...
Displays 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
Runs the selected commands.

New Commands View
Displays a new instance of the Commands view.

10.7.2 See also

Tasks
• ARM® DS-5™ Using Eclipse:
  — Accessing the dynamic help on page 2-38.

Reference
• History view on page 10-35
• Scripts view on page 10-50
• DS-5 Debugger menu and toolbar icons on page 10-99
• ARM® DS-5™ Debugger Command Reference:
  — DS-5 Debugger commands listed in alphabetical order on page 2-26
  — General syntax and usage of DS-5 Debugger commands on page 2-2.
• ARM® DS-5™ Using Eclipse:
  — Perspectives and views on page 2-22.
10.8 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 current working directory

![Figure 10-9 Set the current working directory](image)

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. Some of the views in the DS-5 Debug perspective are associated with the current stack frame. Other views are associated with editors or target connections. Each associated view is synchronized accordingly.

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.
10.8.1 Toolbar and context menu options

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

**Collapse All**
- Collapses all expanded stack trace configurations.

**Connect to Target**
- Connects to the selected target using the same launch configuration settings as the previous connection.

**Disconnect from Target**
- Disconnects from the selected target.

**Remove Connection**
- Removes 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**
- Starts running the application image from the current location.
  - After a breakpoint is hit or the target is interrupted, continues running the application.

  —— Note ——
  - A **Connect only** connection might require setting the PC register to the start of the image before running it.

**Interrupt**
- Interrupts 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, steps at the source level including stepping into all function calls where there is debug information.
• If instruction mode is selected, steps 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, steps at the source level but stepping over all function calls.
• If instruction mode is selected, steps at the instruction level but stepping over all function calls.

Step Out

Continues 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)

Toggles 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

Continues running to the selected stack frame.

Change Connection Color

Enables you to change the color of the connection icon.

View Menu

The following options are available:

Add Configuration (without connecting)...

Displays 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...

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

Path Substitution...

Displays 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.

**Reset DS-5 Views to ‘Linked’**

Resets DS-5 views to link to the selected connection in the Debug Control view.

**Threads Presentation**

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

**Auto Expand Stack**

Controls whether to automatically display an expanded stack when selecting a connection.

### 10.8.2 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 3-10
- Disconnecting from a target on page 3-24
- Running an image on page 4-6
- Stepping through an application on page 4-26
- Configuring the debugger path substitution rules on page 4-32
- Examining the call stack on page 5-4
- ARM DS-5 Using Eclipse:
  - Accessing the dynamic help on page 2-38.

**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 10-12
- DS-5 Debugger menu and toolbar icons on page 10-99
- ARM® DS-5™ Debugger Command Reference:
  - add-symbol-file on page 2-30
  - continue on page 2-50
  - file, symbol-file on page 2-68
  - interrupt, stop on page 2-108
  - load on page 2-111
  - loadfile on page 2-112
  - next on page 2-123
  - nexti on page 2-124
  - nexts on page 2-125
  - reset on page 2-134
  - run on page 2-138
  - set debug-from on page 2-148
— *start* on page 2-194
— *step* on page 2-196
— *stepi* on page 2-197
— *steps* on page 2-198
— *quit, exit* on page 2-132.

- *ARM® DS-5™ Using Eclipse*:
  — *Perspectives and views* on page 2-22.
10.9 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, $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.

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.
10.9.1 Toolbar and context menu options

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

**Linked: context**
Links this view to the selected connection in the Debug Control view. This is the default. Alternatively you can link the view to a specific connection. If the connection you want is not shown in the drop-down list you might have to select it first in the Debug Control view.

**Back, Forward**
Navigates through the history list.

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

**$LR**
Navigates to the LR register.

**expression**
Navigates to the address specified by an expression. $PC+256.

**address**
Navigates 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.

**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.

**Search**
Searches through debug information for symbols.

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

**New Disassembly View**
Displays a new instance of the Disassembly view.

**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.

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

**Refresh** Refreshes the view.

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

**Copy**
Copies the selected address.

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

**Select All**
Selects 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, Ctrl+C on Windows.

**Run to Selection**
Runs to the selected address

**Set PC to Selection**
Sets the PC register to the selected address.

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

**Show in functions**
If the memory address corresponds to a function, then displays the Functions view with the related function selected.

**Toggle Breakpoint**
Sets or removes a breakpoint at the selected address.

**Toggle Hardware Breakpoint**
Sets or removes a hardware breakpoint at the selected address.

**Resolve Breakpoint**
Resolves a pending breakpoint at the selected address.

**Enable Breakpoint**
Enables the breakpoint at the selected address.

**Disable Breakpoint**
Disables the breakpoint at the selected address.

**Toggle Trace Start Point**
Sets or removes a trace start point at the selected address.

**Toggle Trace Stop Point**
Sets or removes a trace stop point at the selected address.

**Toggle Trace Trigger Point**
Starts a trace trigger point at the selected address.
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** Copies and deletes the selected text.
- **Copy** Copies the selected text.
- **Paste** Pastes text that you previously cut or copied.
- **Delete** Deletes the selected text.
- **Undo** Reverts the last change.
- **Select All** Selects all the text.

### 10.9.2 See also

**Tasks**
- Setting an execution breakpoint on page 4-10
- Setting a conditional breakpoint on page 4-15
- Setting a breakpoint on a specific thread 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 2-38.

**Reference**
- C/C++ editor on page 10-12
- Registers view on page 10-44
- DS-5 Debugger menu and toolbar icons on page 10-99
- ARM® DS-5™ Using Eclipse:
  - Perspectives and views on page 2-22.
10.10 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 10-12 Expressions view](image)

### 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.

, 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 expression 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.
- **Access**: The access type of expression.

All columns are displayed by default.
10.10.1 Toolbar and context menu options

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

**Linked:**

Links this view to the selected connection in the Debug Control view. This is the default. Alternatively you can link the view to a specific connection. If the connection you want is not shown in the drop-down list you might have to select it first in the Debug Control view.

**Add New Expression**

Adds a new expression to the expression list.

**Remove Selected Expression**

Removes the selected expression from the list.

**Remove All Expressions**

Removes all expressions from the list.

**Search**

Searches the data in the current view for an expression.

**Cut**

Copies and removes the selected expression.

**Copy**

Copies 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**

Pastes expressions that you have previously cut or copied.

**Delete**

Deletes the selected expression.

**Select All**

Selects 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, \&name
- the location of the expression, 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, $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.

**View Menu**

The following View Menu options are available:

**New Expression View**

Displays a new instance of the Expressions view.
Refresh

Refreshes the view.

Freeze Data

Toggles the freezing of data in the current view. This also disables and enables the Refresh option.

10.10.2 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 2-38.

Reference

• Functions view on page 10-32
• Memory view on page 10-37
• Registers view on page 10-44
• Variables view on page 10-58
• DS-5 Debugger menu and toolbar icons on page 10-99
• ARM® DS-5™ Using Eclipse:
  — Perspectives and views on page 2-22.
10.11 Functions view

This view enables you to:

- see the ELF data associated with function symbols for all loaded images
- freeze the selected view to prevent the information being updated by a running target.

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

- **Name**: The name of the function.
- **Mangled Name**: The C++ mangled name of the function.
- **Base Address**: The function entry point.
- **Start Address**: The start address of the function.
- **End Address**: The end address of the function.
- **Size**: The size of the function in bytes.
- **Compilation Unit**: The location of the compilation unit containing the function.
- **Image**: The location of the ELF image containing the function.

The Name, Start Address, End Address, Compilation Unit, and Image columns are displayed by default.

### 10.11.1 Toolbar and context menu options

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

- **Linked**: Links this view to the selected connection in the Debug Control view. This is the default. Alternatively you can link the view to a specific connection. If the connection you want is not shown in the drop-down list you might have to select it first in the Debug Control view.
Search  Searches the data in the current view for a function.

Copy  Copies the selected functions.

Select All  Selects all the functions in the view.

Run to Selection
  Runs to the selected address

Set PC to Selection
  Sets the PC register to the start address of the selected function.

Show in 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 Memory
  Displays the Memory view starting at the address of the selected function.

Show in Disassembly
  Displays the Disassembly view starting at the address of the selected function.

Toggle Breakpoint  Sets or removes a breakpoint at the selected address.

Resolve Breakpoint  Resolves a pending breakpoint at the selected address.

Enable Breakpoint  Enables the breakpoint at the selected address.

Disable Breakpoint  Disables the breakpoint at the selected address.

Toggle Trace Start Point
  Sets or removes a trace start point at the selected address.

Toggle Trace Stop Point
  Sets or removes a trace stop point at the selected address.

Toggle Trace Trigger Point
  Starts a trace trigger point at the selected address.

View Menu  The following View Menu options are available:

  New Function View  Displays a new instance of the Functions view.

  Refresh  Refreshes the view.

  Freeze Data  Toggles the freezing of data in the current view. This also disables or enables the Refresh option.

  Filter...  Displays the Filter dialog box.

10.11.2  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 2-38.*

**Reference**

• *Functions Filter dialog box on page 10-75*
• *Disassembly view on page 10-25*
• *Memory view on page 10-37*
• *DS-5 Debugger menu and toolbar icons on page 10-99*
• *ARM® DS-5™ Using Eclipse:*
  — *Perspectives and views on page 2-22.*
10.12 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 10-14 History view](image)

--- 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.

10.12.1 Toolbar and context menu options

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

**Linked: context**

Links this view to the selected connection in the Debug Control view. This is the default. Alternatively you can link the view to a specific connection. If the connection you want is not shown in the drop-down list you might have to select it first in the Debug Control view.

**Exports the selected lines as a script**

Displays 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**

Clears the contents of the History view.
Toggles Scroll Lock
   Enables or disables the automatic scrolling of messages in the History view.

Copy    Copies the selected commands.
Select All  Selects all commands.

Save the selected lines as a script...
   Displays 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
   Runs the selected commands.

New History View
   Displays a new instance of the History view.

10.12.2 See also

Tasks
   •  ARM® DS-5™ Using Eclipse:
      —  Accessing the dynamic help on page 2-38.

Reference
   •  Commands view on page 10-17
   •  Scripts view on page 10-50
   •  Functions Filter dialog box on page 10-75
   •  DS-5 Debugger menu and toolbar icons on page 10-99
   •  ARM® DS-5™ Debugger Command Reference:
      —  DS-5 Debugger commands listed in alphabetical order on page 2-26.
   •  ARM® DS-5™ Using Eclipse:
      —  Perspectives and views on page 2-22.
10.13 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, $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.

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.
10.13.1 Toolbar and context menu options

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

**Linked:** context

Links this view to the selected connection in the Debug Control view. This is the default. Alternatively you can link the view to a specific connection. If the connection you want is not shown in the drop-down list you might have to select it first in the Debug Control view.

**Back, Forward**

Navigates 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.

**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**

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

**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. \$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.

**Search**

Searches through debug information for symbols.

**View Menu**

The following View Menu options are available:

**New Memory View**

Displays a new instance of the Memory view.

**Show Tooltips**

Toggles the display of tooltips on memory cell values.

**Byte Order**

Selects the byte order of the memory. The default is **Auto**(LE).

**Clear History**

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

**Import Memory**

Reads data from a file and writes it to memory.
Export Memory
Reads data from memory and writes it to a file.

Refresh
Refreshes the view.

Freeze Data
Toggles the freezing of data in the current view. This also disables or enables the Address and Size fields and the Refresh option.

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
Copies and deletes the selected value.

Copy
Copies the selected value.

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

Delete
Deletes the selected value.

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

Select All
Selects all the address.

Toggle Breakpoint
Sets or removes a breakpoint at the selected address.

Resolve Breakpoint
Resolves a pending breakpoint at the selected address.

Enable Breakpoint
Enables the breakpoint at the selected address.

Disable Breakpoint
Disables the breakpoint at the selected address.

Toggle Trace Start Point
Sets or removes a trace start point at the selected address.

Toggle Trace Stop Point
Sets or removes a trace stop point at the selected address.

Toggle Trace Trigger Point
Starts a trace trigger point at the selected address.

10.13.2 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 2-38.

Reference
- Export memory dialog box on page 10-61
- Import memory dialog box on page 10-63
- DS-5 Debugger menu and toolbar icons on page 10-99
- ARM® DS-5™ Using Eclipse:
  — Perspectives and views on page 2-22.
- ARM® DS-5™ Debugger Command Reference:
  — dump on page 2-62
— memory set on page 2-120
— restore on page 2-137
— x on page 2-217.
10.14 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 dynamically loaded Operating System (OS) modules
• load and unload debug information for a specific module or shared library.

![Figure 10-16 Modules view showing shared libraries](image1)

**Figure 10-16 Modules view showing shared libraries**

![Figure 10-17 Modules view showing operating system modules](image2)

**Figure 10-17 Modules view showing operating system modules**

---

**Note**

A connection must be established and OS support enabled within the debugger before a loadable module can be detected. OS support is automatically enabled when a Linux kernel image is loaded into the debugger. However, you can manually control this by using the `set os` command.

---

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.

10.14.1 Toolbar and context menu options

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

Linked: context
Links this view to the selected connection in the Debug Control view. This is the default. Alternatively you can link the view to a specific connection. If the connection you want is not shown in the drop-down list you might have to select it first in the Debug Control view.

Refresh
Refreshes the view.

Copy Copies the selected data.

Select All Selects 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. 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.

10.14.2 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 2-38.

Reference
- DS-5 Debugger menu and toolbar icons on page 10-99
- ARM® DS-5™ Using Eclipse:
  — Perspectives and views on page 2-22.
ARM® DS-5™ Debugger Command Reference:
- add-symbol-file on page 2-30
- discard-symbol-file on page 2-59
- file, symbol-file on page 2-68.
10.15 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.

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:

```plaintext
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.
10.15.1 Toolbar and context menu options

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

**Linked: context**
Links this view to the selected connection in the Debug Control view. This is the default. Alternatively you can link the view to a specific connection. If the connection you want is not shown in the drop-down list you might have to select it first in the Debug Control view.

**Search**
Searches the data in the current view for a register.

**Copy**
Copies 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**
Selects 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 Disassembly 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**.

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

- **New Register View**
  Creates a new instance of the Registers view.

- **Refresh**
  Refreshes the view.

- **Freeze Data**
  Toggles the freezing of data in the current view. This also disables or enables the **Refresh** option.

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

- **Cut**
  Copies and deletes the selected value.

- **Copy**
  Copies the selected value.

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

- **Delete**
  Deletes the selected value.

- **Undo**
  Reverts the last change you made to the selected value.
10.15.2 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 2-38.

Concepts
- About debugging TrustZone enabled targets on page 6-18.

Reference
- DS-5 Debugger menu and toolbar icons on page 10-99
- ARM® DS-5™ Using Eclipse:
  — Perspectives and views on page 2-22.
10.16 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.

Figure 10-19 Screen buffer parameters for the Fireworks example running on a BeagleBoard

- Freeze the selected view to prevent the screen display being updated by the running target when it next stops.
10.16.1 Toolbar options

The following toolbar options are available:

**Linked: context**

Links this view to the selected connection in the Debug Control view. This is the default. Alternatively you can link the view to a specific connection. If the connection you want is not shown in the drop-down list you might have to select it first in the Debug Control view.

**Refresh**

Refreshes the view.

**Set Screen Buffer Parameters**

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

- **Base Address**
  
  Sets the base address of the screen buffer.

- **Screen Width**
  
  Sets the width of the screen in pixels.

- **Screen Height**
  
  Sets the height of the screen in pixels.

- **Scan Line Alignment**
  
  Sets the byte alignment required for each scan line.

- **Pixel Type**
  
  Selects the pixel type.

- **Pixel Byte Order**
  
  Selects 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**

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

**New Screen Buffer View**

Creates 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.

10.16.2 See also

**Tasks**

- *ARM® DS-5™ Using Eclipse:*
  — Accessing the dynamic help on page 2-38.

**Reference**

- *DS-5 Debugger menu and toolbar icons on page 10-99*
• ARM® DS-5™ Using Eclipse:
  — Perspectives and views on page 2-22.
10.17 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.

--- 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.

10.17.1 Toolbar and context menu options

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

Creates a new script

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

Execute Selected Scripts

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

Edit Selected Scripts

Enables you to edit the selected scripts. The scripts are opened in the C/C++ editor.

Delete Selected Scripts

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

Import Script...

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

Cut

Copies and removes the selected script filename. You are also prompted to delete the script from file system.

Copy

Copies the selected script.

Paste

Pastes 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** Deletes the selected script from the favorites list. You are also prompted to delete the script from file system.

**Select All** Selects all script files.

### 10.17.2 See also

**Tasks**
- *ARM® DS-5™ Using Eclipse:*
  - Accessing the dynamic help on page 2-38.

**Reference**
- *C/C++ editor* on page 10-12
- *Commands view* on page 10-17
- *History view* on page 10-35
- *DS-5 Debugger menu and toolbar icons* on page 10-99
- *ARM® DS-5™ Using Eclipse:*
  - Perspectives and views on page 2-22.
10.18 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.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>

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.

10.18.1 Toolbar and context menu options

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

Linked: context

Links this view to the selected connection in the Debug Control view. This is the default. Alternatively you can link the view to a specific connection. If the connection you want is not shown in the drop-down list you might have to select it first in the Debug Control view.

Refresh the Target Capabilities

Refreshes the view.
Copy

Copies 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

Selects all capabilities currently expanded in the view.

10.18.2 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 2-38.

Reference

• Breakpoints view on page 10-8
• Memory view on page 10-37
• App Console view on page 10-3
• DS-5 Debugger menu and toolbar icons on page 10-99
• ARM® DS-5™ Using Eclipse:
  — Perspectives and views on page 2-22.
10.19 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.

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.
The Trace view contains several tabs:

- Trace tab showing the graphical timeline and disassembly

### 10.19.1 Toolbar and context menu options

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

**Linked:**

Links this view to the selected connection in the Debug Control view. This is the default. Alternatively you can link the view to a specific connection. If the connection you want is not shown in the drop-down list you might have to select it first in the Debug Control view.

**Show Next Match**

Moves the focus of the navigation chart and disassembly trace to the next matching occurrence for the selected function or instruction.

**Show Previous Match**

Moves the focus of the navigation chart and disassembly trace to the previous matching occurrence for the selected function or instruction.

**Don’t mark other occurrences - click to start marking, Mark other occurrences - click to stop marking**

When function trace is selected, marks all occurrences of the selected function with a shaded highlight. This is disabled when instruction trace is selected.

**Clear Trace**

Clears the raw trace data that is currently contained in the trace buffer and the trace view.

**Showing instruction trace - click to switch to functions, Showing function trace - click to switch to instructions**

Toggles the disassembly trace between instruction trace or function trace.

**Export Trace Report**

Displays the Export Trace Report dialog box to save the trace data to a file.

**Home**

Where enabled, moves the trace view to the beginning of the trace buffer. Changes might not be visible if the trace buffer is too small.

**Page Back**

Where enabled, moves the trace view back one page. You can change the page size by modifying the **Set Maximum Instruction Depth** setting.

**Page Forward**

Where enabled, moves the trace view forward one page. You can change the page size by modifying the **Set Maximum Instruction Depth** setting.

**End**

Where enabled, moves the trace view to the end of the trace buffer. Changes might not be visible if the trace buffer is too small.

**Switch between navigation resolutions**

Changes the timeline resolution in the navigation chart.

**Switch between alternate views**

Changes the view to display the navigation chart, disassembly trace or both.
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:

New Trace View
Displays a new instance of the Trace view.

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. The number must be within the range of one thousand to one million instructions.

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.

Find Trigger Packet
Enables you to search for trigger packets in the trace capture buffer.

Show in Disassembly
Displays the Disassembly view starting at the address of the selected instruction.

Refresh
Refreshes the view.

Freeze Data
Toggles the freezing of data in the current view.

10.19.2 See also

Tasks
• Setting a tracepoint on page 4-14
• 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 2-38.

Concepts
• About trace support on page 5-6

Reference
• ARM assembler editor on page 10-6
• Breakpoints view on page 10-8
• C/C++ editor on page 10-12
- *Disassembly view* on page 10-25
- *Functions view* on page 10-32
- *Memory view* on page 10-37
- *Export trace report dialog box* on page 10-65
- *Tracepoint properties dialog box* on page 10-72
- *DS-5 Debugger menu and toolbar icons* on page 10-99
- *ARM® DS-5™ Using Eclipse:*
  — *Perspectives and views* on page 2-22.
10.20 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 10-24 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.

10.20.1 Toolbar and context menu options

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

- **Linked: context**: Links this view to the selected connection in the Debug Control view. This is the default. Alternatively you can link the view to a specific connection. If the connection you want is not shown in the drop-down list you might have to select it first in the Debug Control view.

- **Search**: Searches the data in the current view for a variable.
Copy

Copies 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

Selects 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, the address of an array, \&name
- the location of the variable, 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 Dereference 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 Signed Decimal.

View Menu

The following View Menu options are available:

New Variable View

Displays a new instance of the Variables view.

Refresh

Refreshes the view.

Freeze Data

Toggles the freezing of data in the current 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.

Editing context menu options

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

Cut

Copies and deletes the selected value.

Copy

Copies the selected value.

Paste

Pastes a value that you have previously cut or copied into the selected variable value.

Delete

Deletes the selected value.

Undo

Reverts the last change you made to the selected value.
Right to left reading order
Sets the reading order for the selected variable value to be left or right justified.

Show unicode control characters
Shows any unicode control characters in the selected variable value.

Insert unicode control character
Selects the unicode control character to insert into the selected variable value.

10.20.2 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 2-38.

Reference
• Expressions view on page 10-29
• Functions view on page 10-32
• Memory view on page 10-37
• Registers view on page 10-44
• DS-5 Debugger menu and toolbar icons on page 10-99
• ARM® DS-5™ Using Eclipse:
  — Perspectives and views on page 2-22.
10.21 Export memory dialog box

This dialog box enables you to generate a text file containing the data from a specific region of memory.

**Memory Bounds**

Specifies the memory region:

- **Start Address**  Specifies the start address for the memory.
- **End Address**  Specifies the inclusive end address for the memory.
- **Length in Bytes**  Specifies the number of bytes.

**Output Format**

Specifies the output format:

- **Binary.** This is the default.
- **Intel Hex-32.**
- **Motorola 32-bit (S-records).**
- **Byte oriented hexadecimal (Verilog Memory Model).**

**Export Filename**

Enter the current location of the output file in the field provided or click on:

- **File System...** to locate the output file in an external folder
- **Workspace...** to locate the output file in a workspace project.

![Figure 10-25 Memory Exporter dialog box](image)

**10.21.1 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 2-38.*

**Reference**

- *Import memory dialog box on page 10-63*
- *Memory view on page 10-37*
- *DS-5 Debugger menu and toolbar icons on page 10-99*
- *ARM® DS-5™ Using Eclipse:*
  - *Perspectives and views on page 2-22.*
10.22 Import memory dialog box

This dialog box enables you to generate a text file containing the data from a specific region of memory.

Offset to Embedded Address

Specifies an offset that is added to all addresses in the image prior to writing to memory. Some image formats do not contain embedded addresses and in this case the offset is the absolute address where the image is restored.

Memory Limit

Enables you to define a specific region of memory that you want to import:

Limit to memory range

Select to specify an address range.

Start

Specifies the minimum address that can be written to. Any data prior to this address is not written. If no address is given then the default is address zero.

End

Specifies the maximum address that can be written to. Any data after this address is not written. If no address is given then the default is the end of the address space.

Import Filename

Select Import file as binary image if the file format is a binary file.

Enter the current location of the output file in the field provided or click on:

• File System... to locate the output file in an external folder
• Workspace... to locate the output file in a workspace project.

![Memory Importer dialog box](image)

Figure 10-26 Memory Importer dialog box

10.22.1 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 2-38.
Reference

- Export memory dialog box on page 10-61
- Memory view on page 10-37
- DS-5 Debugger menu and toolbar icons on page 10-99
- ARM® DS-5™ Using Eclipse:
  — Perspectives and views on page 2-22.
10.23 Export trace report dialog box

This dialog box enables you to generate a trace report.

Select source for trace report

Selects the required trace data:

Use trace view as report source

Instructions that are currently visible in the trace view.

Use trace buffer as report source

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

Configures the report:

Output Format

Selects 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

Selects the report location in the file system.

Figure 10-27 Export trace report dialog box
10.23.1 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 2-38.

**Concepts**
- About trace support on page 5-6

**Reference**
- Trace view on page 10-54
- DS-5 Debugger menu and toolbar icons on page 10-99
- ARM® DS-5™ Using Eclipse:
  - Perspectives and views on page 2-22.
10.24 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 or processor, if available.

![Breakpoint Properties dialog box](image)

**Figure 10-28 Breakpoint properties dialog box**

10.24.1 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, `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, (i==3).
- The name of the image.
  : 
  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.
  :
  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.
10.24.2 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**

Specify 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.

**Silent**

Controls the printing of stop messages for the selected breakpoint.

**Break on Selected Threads or Cores**

Select this option if you want to set a breakpoint for a specific thread or processor. This option is disabled if none 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)
  ```

10.24.3 See also

Tasks

- *Setting an execution breakpoint on page 4-10*
- *Setting a conditional breakpoint on page 4-15*
• Pending breakpoints and watchpoints on page 4-20
• 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 2-38.

Reference
• Breakpoints view on page 10-8
• DS-5 Debugger menu and toolbar icons on page 10-99
• ARM® DS-5™ Debugger Command Reference:
  — clear on page 2-46
  — delete breakpoints on page 2-53
  — disable breakpoints on page 2-56
  — enable breakpoints on page 2-64
  — ignore on page 2-77
  — set breakpoint on page 2-147
  — silence on page 2-192.
10.25 Watchpoint properties dialog box

This dialog box enables you to:
- display the properties of a selected watchpoint
- change the watchpoint type.

![Watchpoint Properties dialog box](image)

**Figure 10-29 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.

10.25.1 See also

Tasks
- Setting a data watchpoint on page 4-12
- 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 2-38.

Reference
- Breakpoints view on page 10-8
- DS-5 Debugger menu and toolbar icons on page 10-99
- ARM® DS-5™ Debugger Command Reference:
  - awatch on page 2-34
  - delete breakpoints on page 2-53
  - disable breakpoints on page 2-56
  - enable breakpoints on page 2-64
  - ignore on page 2-77
  - rwatch on page 2-139
  - watch on page 2-212.
10.26 **Tracepoint properties dialog box**

This dialog box enables you to display the properties of a selected tracepoint.

![Tracepoint Properties](image)

**Figure 10-30 Tracepoint properties dialog box**

The following types are available:

- **Trace Start Point** Enables trace capture when it is hit.
- **Trace Stop Point** Disables trace capture when it is hit.
- **Trace Trigger Point** Starts trace capture when it is hit.

--- **Note** ---

Tracepoint behavior might vary depending on the selected target.

10.26.1 **See also**

**Tasks**
- *Setting a tracepoint* on page 4-14
- *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 2-38.

**Reference**
- *ARM assembler editor* on page 10-6
- *Breakpoints view* on page 10-8
- *C/C++ editor* on page 10-12
- *Disassembly view* on page 10-25
- *Functions view* on page 10-32
- *Memory view* on page 10-37
- *Trace view* on page 10-54
- *DS-5 Debugger menu and toolbar icons* on page 10-99.
10.27 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.

10.27.1 See also

Tasks
- Handling Unix signals on page 4-28
- Handling processor exceptions on page 4-30
- ARM® DS-5™ Using Eclipse:
  — Accessing the dynamic help on page 2-38.
Reference

- *Target view on page 10-52*
- *DS-5 Debugger menu and toolbar icons on page 10-99.*
10.28 Functions Filter dialog box

This dialog box enables you to filter the list of symbols that are displayed in the Functions view.

Figure 10-33 Function filter dialog box

10.28.1 See also

Tasks
- ARM® DS-5™ Using Eclipse:
  - Accessing the dynamic help on page 2-38.

Reference
- Functions view on page 10-32
10.29 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.
  
  --- Note ---
  Contact your support representative if your target is not listed.

- **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.
  
  **APK native library debug**
  Select to debug an application running on an Android 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, Download and debug an Android 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 and starts a new gdbserver session.
  
  **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 connection
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.

Android devices
A list of Android devices that you have previously configured. Select the required device that you want to use for this debug configuration.

Connect as root
Select to give root access when starting gdbserver. This option is dependent on the selected debug operation and might not be available.

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.

If you select the workspace, 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.
10.29.1 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 3-8
- Configuring a connection to a bare-metal target on page 3-10
- Configuring an Event Viewer connection to a bare-metal target on page 3-12
- Importing an existing launch configuration on page 3-22
- Exporting an existing launch configuration on page 3-20
- ARM® DS-5™ Getting Started with DS-5:
  - Loading the Gnometris application on a Real-Time System Model on page 3-7
  - Loading the Gnometris application on to an ARM Linux target on page 3-8
  - Using an RSE connection to work with an ARM Linux target on page 3-9.
- ARM® DS-5™ Using Eclipse:
  - Accessing the dynamic help on page 2-38
  - Chapter 5 Working with Remote System Explorer.
Reference

- *Debug Configurations - Files tab* on page 10-81
- *Debug Configurations - Debugger tab* on page 10-86
- *Debug Configurations - Arguments tab* on page 10-90
- *Debug Configurations - Environment tab* on page 10-93
- *Debug Configurations - Event Viewer tab* on page 10-96
- *DS-5 Debugger menu and toolbar icons* on page 10-99
10.30 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>
<thead>
<tr>
<th>Application on host to download</th>
<th>Download and debug application</th>
<th>Debug target resident application</th>
<th>Connect to already running gdbserver</th>
<th>Debug via DSTREAM/RVI</th>
<th>Debug and ETB Trace via DSTREAM/RVI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Application on target</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Target download directory</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Target working directory</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Load symbols from file</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Other file on host to download</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Path to target system root directory</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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.
10.30.1 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) GnomeGC application image select the gnomeGC/stripped/gnomeGC 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.
Project directory

Specify the Android project directory on the host:
• enter the host location in the field provided
• click on File System... to locate the project directory in an external location from the Eclipse workspace
• click on Workspace... to locate the project directory from within the Eclipse workspace.

APK file

Specify the Android APK 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 a project directory or sub-directory within the Eclipse workspace.

Process

This field is automatically populated from the AndroidManifest.xml file.

Activity

This field is automatically populated from the AndroidManifest.xml file.

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.

### Path to target system root directory

Specifies the system root directory to search for shared library symbols.

The debugger uses this directory to search for a copy of the debug versions of target shared libraries. The system root on the host workstation must contain an exact representation of the libraries on the target root filesystem.

### 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.

### 10.30.2 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 Linux Kernel** on page 3-8
- **Configuring a connection to a bare-metal target** on page 3-10
- **Configuring an Event Viewer connection to a bare-metal target** on page 3-12
- **Importing an existing launch configuration** on page 3-22
- **Exporting an existing launch configuration** on page 3-20
- **ARM® DS-5™ Getting Started with DS-5**:  
  — **Loading the Gnometris application on a Real-Time System Model** on page 3-7
Loading the Gnometris application on to an ARM Linux target on page 3-8.

- ARM® DS-5™ Using Eclipse:
  - Accessing the dynamic help on page 2-38.

Reference

- Debug Configurations - Connection tab on page 10-76
- Debug Configurations - Debugger tab on page 10-86
- Debug Configurations - Arguments tab on page 10-90
- Debug Configurations - Environment tab on page 10-93
- Debug Configurations - Event Viewer tab on page 10-96
- DS-5 Debugger menu and toolbar icons on page 10-99.
10.31 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 target initialization scripts (a file containing debugger commands) immediately 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 debug initialization scripts (a file containing debugger commands) after execution of any target initialization scripts and also running to an image entry point or symbol, if selected.

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 have 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 that run on completion of any initialization scripts. 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 10-36 Debugger configuration to set application starting point and search paths

10.31.1 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 3-8
- Configuring a connection to a bare-metal target on page 3-10
- Configuring an Event Viewer connection to a bare-metal target on page 3-12
- Importing an existing launch configuration on page 3-22
- Exporting an existing launch configuration on page 3-20
  - ARM® DS-5™ Getting Started with DS-5:
    - Loading the Gnometris application on a Real-Time System Model on page 3-7
    - Loading the Gnometris application on to an ARM Linux target on page 3-8.
- ARM® DS-5™ Using Eclipse:
  - Accessing the dynamic help on page 2-38.

Reference
- Debug Configurations - Connection tab on page 10-76
- Debug Configurations - Files tab on page 10-81
• Debug Configurations - Arguments tab on page 10-90
• Debug Configurations - Environment tab on page 10-93
• Debug Configurations - Event Viewer tab on page 10-96
• DS-5 Debugger menu and toolbar icons on page 10-99.
10.32 Debug Configurations - Arguments tab

If your application accepts command-line arguments to \texttt{main()}, you can specify the values to pass to the application when execution starts.

The \textbf{Arguments} tab in the Debug Configurations dialog box enables you to enter arguments that are passed to the application.

\begin{itemize}
\item \textbf{Note} \hfill \\
The settings in this tab are not used for connections that use the \textbf{Connect to already running gdbserver} debug operation.
\end{itemize}

The \textbf{Arguments} tab contains the following elements:

\begin{itemize}
\item \textbf{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 ${\texttt{var\_name}}$ 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.

\item \textbf{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.

\item \textbf{Apply} \\
Save the current configuration. This does not connect to the target.

\item \textbf{Revert} \\
Undo any changes and revert to the last saved configuration.

\item \textbf{Debug} \\
Connect to the target and close the Debug Configurations dialog box.

\item \textbf{Close} \\
Close the Debug Configurations dialog box.
\end{itemize}
Figure 10-37 Application arguments configuration

10.32.1 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 3-8
- Configuring a connection to a bare-metal target on page 3-10
- Configuring an Event Viewer connection to a bare-metal target on page 3-12
- Importing an existing launch configuration on page 3-22
- Exporting an existing launch configuration on page 3-20
- ARM® DS-5™ Getting Started with DS-5:
  — Loading the Gnometris application on a Real-Time System Model on page 3-7
  — Loading the Gnometris application on to an ARM Linux target on page 3-8.
- ARM® DS-5™ Using Eclipse:
  — Accessing the dynamic help on page 2-38.

Reference

- Debug Configurations - Connection tab on page 10-76
- Debug Configurations - Files tab on page 10-81
- Debug Configurations - Debugger tab on page 10-86
- Debug Configurations - Environment tab on page 10-93
- Debug Configurations - Event Viewer tab on page 10-96
- DS-5 Debugger menu and toolbar icons on page 10-99.
10.33 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 ---

The 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 10-38 Setting up target environment variables](image)

**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.
10.33.1 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 3-8
- Configuring a connection to a bare-metal target on page 3-10
- Configuring an Event Viewer connection to a bare-metal target on page 3-12
- Importing an existing launch configuration on page 3-22
- Exporting an existing launch configuration on page 3-20
- ARM® DS-5™ Getting Started with DS-5:
  - Loading the Gnometrism application on a Real-Time System Model on page 3-7
  - Loading the Gnometrism application on to an ARM Linux target on page 3-8.
- ARM® DS-5™ Using Eclipse:
  - Accessing the dynamic help on page 2-38.

Reference
- Debug Configurations - Connection tab on page 10-76
- Debug Configurations - Files tab on page 10-81
• Debug Configurations - Debugger tab on page 10-86
• Debug Configurations - Arguments tab on page 10-90
• Debug Configurations - Event Viewer tab on page 10-96
• DS-5 Debugger menu and toolbar icons on page 10-99.
10.34 Debug Configurations - Event Viewer tab

The Event Viewer tab in the Debug Configurations dialog box enables you to configure the Instrumentation Trace Macrocell (ITM) channels that you want the debugger to use when capturing event data.

--- Note ---

The settings in this tab are only enabled for targets where ITM capture is supported.

---

The Event Viewer tab contains the following elements:

**Enable ITM capture**

Enable or disable the use of ITM capture.

**Max analysis trace depth**

Specify the size of the trace buffer. The maximum is 4GB but you can choose to use only a portion of the buffer in order to minimise data transfer and processing times. The data stored in the buffer is encoded as ITM packets, and might be interleaved with other trace data. Therefore the actual amount of your own data that can be recovered from the buffer might be less than the configured buffer size.

**ITM Channel Configuration**

Configure ITM channels to match the annotations used in the application source code.

**ITM Channel**

Specify the channel number.

**Encoding**

Select the encoding for the selected channel.

- RAW TEXT
  - Select for raw text.

- ENCODED
  - Select to encode the data in a format that enables timestamps to be generated in addition to the raw text.

**File Output**

Specify where you want to send the event data.

- None
  - Select to display data in the Channel editor, not in a file.

- Append
  - Select to display data in the Channel editor and append to an existing data in a text file.

- Overwrite
  - Select to display data in the Channel editor and write to a text file. Any existing data in the text file is discarded.

**Output File**

Specify the output file that you want to use when saving event data:

- 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.
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.

![Debug Configurations dialog box](image)

**Figure 10-40 Event Viewer configuration**

### 10.34.1 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 3-8
- Configuring a connection to a bare-metal target on page 3-10
- Configuring an Event Viewer connection to a bare-metal target on page 3-12
- Importing an existing launch configuration on page 3-22
- Exporting an existing launch configuration on page 3-20
- ARM® DS-5™ Getting Started with DS-5:
  - Loading the Gnometris application on a Real-Time System Model on page 3-7
  - Loading the Gnometris application on to an ARM Linux target on page 3-8.
• ARM® DS-5™ Using Eclipse:
  — Accessing the dynamic help on page 2-38.

Reference
• Channel editor for the Event Viewer on page 10-15
• Debug Configurations - Connection tab on page 10-76
• Debug Configurations - Files tab on page 10-81
• Debug Configurations - Debugger tab on page 10-86
• Debug Configurations - Environment tab on page 10-93
• DS-5 Debugger menu and toolbar icons on page 10-99.
10.35 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.
### 10.35.1 DS-5 Debugger icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Icon]</td>
<td>Connect to target</td>
<td>![Icon]</td>
<td>Connected to target</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Disconnect from target</td>
<td>![Icon]</td>
<td>Delete connection</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Start application and run to main</td>
<td>![Icon]</td>
<td>Start application and run to entry point</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Run application from entry point</td>
<td>![Icon]</td>
<td>Restart the application</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Continue running application</td>
<td>![Icon]</td>
<td>Stop application</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Step into</td>
<td>![Icon]</td>
<td>Step over</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Step out</td>
<td>![Icon]</td>
<td>Toggle stepping mode</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Collapse all configurations in stack trace</td>
<td>![Icon]</td>
<td>Call stack</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Thread</td>
<td>![Icon]</td>
<td>Process</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Kernel module</td>
<td>![Icon]</td>
<td>Define a new RSE connection</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Refresh the RSE resource tree</td>
<td>![Icon]</td>
<td>Save view contents to a file</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Clear view contents</td>
<td>![Icon]</td>
<td>Switch to History view</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Synchronize view contents</td>
<td>![Icon]</td>
<td>Toggle scroll lock</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Run commands from a script file</td>
<td>![Icon]</td>
<td>Export commands to a script file</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Remove selected breakpoint, watchpoints, or expression (view dependent)</td>
<td>![Icon]</td>
<td>Remove all breakpoints, watchpoints, or expressions (view dependent)</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Display breakpoint location in source file</td>
<td>![Icon]</td>
<td>Deactivate all breakpoints and watchpoints</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Import from a file</td>
<td>![Icon]</td>
<td>Export to a file</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Create new script file or add new expression (view dependent)</td>
<td>![Icon]</td>
<td>Run select script file</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Open selected file for editing</td>
<td>![Icon]</td>
<td>Delete the selected files</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Set display width</td>
<td>![Icon]</td>
<td>Set display format</td>
</tr>
<tr>
<td>Icon</td>
<td>Description</td>
<td>Icon</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>------------------------------------------</td>
<td>------</td>
<td>------------------------</td>
</tr>
<tr>
<td>ab</td>
<td>Toggle the display of ASCII characters</td>
<td></td>
<td>Toggle freeze mode</td>
</tr>
<tr>
<td></td>
<td>Edit Screen view parameters</td>
<td></td>
<td>Add new Screen view</td>
</tr>
<tr>
<td></td>
<td>Add new Disassembly view</td>
<td></td>
<td>Add new Variables view</td>
</tr>
<tr>
<td></td>
<td>Add new Registers view</td>
<td></td>
<td>Add new Memory view</td>
</tr>
<tr>
<td></td>
<td>Add new Expression view</td>
<td></td>
<td>Add new Trace view</td>
</tr>
<tr>
<td></td>
<td>Add Functions view</td>
<td></td>
<td>View update in progress</td>
</tr>
<tr>
<td></td>
<td>Toggle trace marker</td>
<td></td>
<td>Show next match</td>
</tr>
<tr>
<td></td>
<td>Show previous match</td>
<td></td>
<td>Show instruction trace</td>
</tr>
<tr>
<td></td>
<td>Show function trace</td>
<td></td>
<td>Toggle navigation resolution</td>
</tr>
<tr>
<td></td>
<td>Toggle the views</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 10.35.2 Perspective icons

#### Table 10-3 Perspective icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Open new perspective" /></td>
<td>Open new perspective</td>
<td><img src="image" alt="C/C++" /></td>
<td>C/C++</td>
</tr>
<tr>
<td><img src="image" alt="DS-5 Debug" /></td>
<td>DS-5 Debug</td>
<td><img src="image" alt="Fast view bar" /></td>
<td>Fast view bar</td>
</tr>
</tbody>
</table>

### 10.35.3 View icons

#### Table 10-4 View icons

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Display drop-down menu" /></td>
<td>Display drop-down menu</td>
<td><img src="image" alt="Synchronize view contents" /></td>
<td>Synchronize view contents</td>
</tr>
<tr>
<td><img src="image" alt="Minimize" /></td>
<td>Minimize</td>
<td><img src="image" alt="Maximize" /></td>
<td>Maximize</td>
</tr>
<tr>
<td><img src="image" alt="Restore" /></td>
<td>Restore</td>
<td><img src="image" alt="Close" /></td>
<td>Close</td>
</tr>
</tbody>
</table>

### 10.35.4 View markers

#### Table 10-5 View markers

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Software breakpoint enabled" /></td>
<td>Software breakpoint enabled</td>
<td><img src="image" alt="Hardware breakpoint enabled" /></td>
<td>Hardware breakpoint enabled</td>
</tr>
<tr>
<td><img src="image" alt="Access watchpoint enabled" /></td>
<td>Access watchpoint enabled</td>
<td><img src="image" alt="Read watchpoint enabled" /></td>
<td>Read watchpoint enabled</td>
</tr>
<tr>
<td><img src="image" alt="Write watchpoint enabled" /></td>
<td>Write watchpoint enabled</td>
<td><img src="image" alt="Software breakpoint disabled" /></td>
<td>Software breakpoint disabled</td>
</tr>
<tr>
<td><img src="image" alt="Hardware breakpoint disabled" /></td>
<td>Hardware breakpoint disabled</td>
<td><img src="image" alt="Access watchpoint disabled" /></td>
<td>Access watchpoint disabled</td>
</tr>
<tr>
<td><img src="image" alt="Read watchpoint disabled" /></td>
<td>Read watchpoint disabled</td>
<td><img src="image" alt="Write watchpoint disabled" /></td>
<td>Write watchpoint disabled</td>
</tr>
<tr>
<td><img src="image" alt="Software breakpoint pending" /></td>
<td>Software breakpoint pending</td>
<td><img src="image" alt="Hardware breakpoint pending" /></td>
<td>Hardware breakpoint pending</td>
</tr>
<tr>
<td><img src="image" alt="Access watchpoint pending" /></td>
<td>Access watchpoint pending</td>
<td><img src="image" alt="Read watchpoint pending" /></td>
<td>Read watchpoint pending</td>
</tr>
<tr>
<td><img src="image" alt="Write watchpoint pending" /></td>
<td>Write watchpoint pending</td>
<td><img src="image" alt="Software breakpoint disconnected" /></td>
<td>Software breakpoint disconnected</td>
</tr>
<tr>
<td><img src="image" alt="Hardware breakpoint disconnected" /></td>
<td>Hardware breakpoint disconnected</td>
<td><img src="image" alt="Access watchpoint disconnected" /></td>
<td>Access watchpoint disconnected</td>
</tr>
<tr>
<td><img src="image" alt="Read watchpoint disconnected" /></td>
<td>Read watchpoint disconnected</td>
<td><img src="image" alt="Write watchpoint disconnected" /></td>
<td>Write watchpoint disconnected</td>
</tr>
</tbody>
</table>
### Table 10-5 View markers (continued)

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Symbol" /></td>
<td>Multiple-statement software breakpoint enabled</td>
<td><img src="image2.png" alt="Symbol" /></td>
<td>Multiple-statement software breakpoint disabled</td>
</tr>
<tr>
<td><img src="image3.png" alt="Symbol" /></td>
<td>Error</td>
<td><img src="image4.png" alt="Symbol" /></td>
<td>Current location</td>
</tr>
<tr>
<td><img src="image5.png" alt="Symbol" /></td>
<td>Warning</td>
<td><img src="image6.png" alt="Symbol" /></td>
<td>Bookmark</td>
</tr>
<tr>
<td><img src="image7.png" alt="Symbol" /></td>
<td>Information</td>
<td><img src="image8.png" alt="Symbol" /></td>
<td>Task</td>
</tr>
<tr>
<td><img src="image9.png" alt="Symbol" /></td>
<td>Search result</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 10.35.5 Miscellaneous icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image10.png" alt="Symbol" /></td>
<td>Open a new resource wizard</td>
</tr>
<tr>
<td><img src="image11.png" alt="Symbol" /></td>
<td>Open new folder wizard</td>
</tr>
<tr>
<td><img src="image12.png" alt="Symbol" /></td>
<td>Open search dialog box</td>
</tr>
<tr>
<td><img src="image13.png" alt="Symbol" /></td>
<td>Open import wizard</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image14.png" alt="Symbol" /></td>
<td>Open new project wizard</td>
</tr>
<tr>
<td><img src="image15.png" alt="Symbol" /></td>
<td>Open new file wizard</td>
</tr>
<tr>
<td><img src="image16.png" alt="Symbol" /></td>
<td>Display context-sensitive help</td>
</tr>
<tr>
<td><img src="image17.png" alt="Symbol" /></td>
<td>Open export wizard</td>
</tr>
</tbody>
</table>
Chapter 11
Troubleshooting

The following topics describe how to diagnose problems when debugging applications using DS-5 Debugger.

Concepts

• ARM Linux problems and solutions on page 11-2
• Enabling internal logging from the debugger on page 11-3
• Target connection problems and solutions on page 11-4.

Other information

• ARM Technical Support Knowledge Articles,
11.1 **ARM Linux problems and solutions**

You might encounter the following problems when debugging a Linux application.

11.1.1 **ARM Linux permission problem**

If you receive a permission denied error message when starting an application on the target then you might have to change the execute permissions on the application.:

```bash
chmod +x myImage
```

11.1.2 **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 require debug information.

11.1.3 **Operating system support is not active**

When Operating System (OS) support is required, the debugger activates it automatically where possible. If OS support is required but cannot be activated, the debugger produces an error:

```bash
ERROR(CMD16-LKN36):
! Failed to load image "gator.ko"
! Unable to parse module because the operating system support is not active
```

OS support cannot be activated if:

- debug information in the `vmlinux` file does not correctly match the data structures in the kernel running on the target
- it is manually disabled by using the `set os enabled off` command.

To determine whether the kernel versions match:

- stop the target after loading the `vmlinux` image
- enter the `print init_nsproxy.uts_ns->name` command
- verify that the `$1` output is correct:

```bash
$1 = {sysname = "Linux", nodename = "AEL", release = "2.6.35-ds5", 
      version = "#1 Thu Nov 25 11:34:07 GMT 2010", machine = "armv7l", domainname = 
      "(none)"}
```

11.1.4 **See also**

**Tasks**

- *About debugging shared libraries on page 6-11.*
11.2 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.

11.2.1 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.

11.2.2 See also

Tasks
- Chapter 1 Conventions and feedback.

Reference
- Commands view on page 10-17.
- ARM® DS-5™ Debugger Command Reference:
  - log config on page 2-113
  - log file on page 2-114.
- ARM® DS-5™ Using Eclipse:
  - Perspectives and views on page 2-22.
11.3 Target connection problems and solutions

You might encounter the following problems when connecting to a target.

11.3.1 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.

11.3.2 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 Base (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.

11.3.3 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 3-10
- Disconnecting from a target on page 3-24
- About semihosting and top of memory on page 8-2.

Other Information

- The documentation supplied with the board.