ARM® Streamline™ Performance Analyzer

Using ARM Streamline

ARM®

ARM DUI 0482A (ID100210)

ARM Streamline Performance Analyzer
Using ARM Streamline

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
September 2010	A	Non-Confidential	ARM Streamline Performance Analyzer 1.0

Proprietary Notice

Confidentiality Status

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

Product Status

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

Web Address

`http://www.arm.com`

Contents
ARM Streamline Performance Analyzer Using ARM Streamline

Chapter 1
Conventions and feedback

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

Typographical conventions

The following typographical conventions are used:

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

`mono`space Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.

`monospace italic`

Denotes arguments to commands and functions where the argument is to be replaced by a specific value.

`monospace bold`

Denotes language keywords when used outside example code.

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

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

Feedback on this product

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

- your name and company

- the serial number of the product
- details of the release you are using
- details of the platform you are using, such as the hardware platform, operating system type and version
- a small standalone sample of code that reproduces the problem
- a clear explanation of what you expected to happen, and what actually happened
- the commands you used, including any command-line options
- sample output illustrating the problem
- the version string of the tools, including the version number and build numbers.

Feedback on content

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

- the title
- the number, ARM DUI 0482A
- if viewing online, the topic names to which your comments apply
- if viewing a PDF version of a document, the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

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

Other information

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

Chapter 2
Getting Started with ARM Streamline

ARM Streamline Performance Analyzer is a system-wide visualizer and profiler for systems running ARM Linux. Combining an ARM Linux kernel driver, target daemon, and an Eclipse-based user interface, it transforms sampling data and system trace into reports that present the data in both visual and statistical forms. Streamline leverages hardware performance counters with kernel metrics to provide an accurate representation of system resources. Streamline supports ARM9 and Cortex-A8 running ARM Linux.

The following topics describe the system requirements, how to set up a run configuration, and give an overview of the available views:

2.1 Setting up the gator driver and daemon

To enable profiling, you must load the gator daemon and driver on your target.

Prerequisites

- Linux kernel source code for the target platform. Note that Streamline supports only Linux kernel version 2.6.32 and above.
- Cross compiler for building the Linux kernel. You could alternatively use the ARM Linux GCC that comes with DS-5.

2.1.1 Setting up the gator driver and daemon

The gator daemon and driver set the target up to communicate Streamline session data to the host. They collect the statistics produced by the performance counters and kernel metrics and send them to the host, where you can view them in Eclipse for DS-5.

To install the gator driver and daemon, prepare and build your kernel by following these steps:

1. Navigate to the root source directory of the Linux kernel.
2. Invoke the following command in your shell:

   ```
   make ARCH=arm CROSS_COMPILE=${CROSS_TOOLS}/bin/arm-none-linux-gnueabi- Your_SoC_defconfig
   ```

3. Invoke the following command in your shell:

   ```
   make ARCH=arm make ARCH=arm
   CROSS_COMPILE=${CROSS_TOOLS}/bin/arm-none-linux-gnueabi- menuconfig
   ```

4. Activate the Profiling Support option in General Setup: `CONFIG_PROFILING=y`.
5. Under Kernel hacking, activate the **Tracers** and **Trace process context switches and events** options: `CONFIG_FTRACE=y` and `CONFIG_ENABLE_DEFAULT_TRACERS=y`.
6. Invoke the following command in your shell to build the kernel:

   ```
   make -j5 ARCH=arm CROSS_COMPILE=${CROSS_TOOLS}/bin/arm-none-linux-gnueabi- uImage
   ```

2.1.2 Building the gator module

To run Streamline against your target, you need to build the gator driver, `gator.ko` and place it in the same directory as the gator daemon, `gatord`, on the target file system. Assuming that you have all of the required tools for building kernel modules, invoke the following command to create the gator.ko module:

```
make -C kernel_build_dir M=`pwd` ARCH=arm CROSS_COMPILE=<...> modules
```

2.1.3 Running the gator daemon on your target

Now that you have all of the necessary files in place, it is time to start the gator daemon. To run `gatord`:

1. Load the kernel onto the target
2. Copy `gatord` and `gator.ko` into the file system on the target

   Note

   `gatord` is located in *installdir*`/arm/armv5t/` on your host. `gatord` must be placed in the same directory as the `gator.ko` on the target.

3. Ensure gatord has execute permission by invoking the following command:

   ```
   chmod +x gatord
   ```

4. Make sure that you have root privileges and enter the following to execute the gator daemon:

   ```
   ./gatord &
   ```

By default, `gatord` uses port 8080 for communication with the host, but this can be adjusted by launching `gatord` with the port number as a parameter and changing the **Port** option in the run configuration dialog within Eclipse for DS-5.

2.1.4 See also

Tasks

- *Creating a Streamline-enabled configuration* on page 2-4
- *Capturing and analyzing data for the threads example* on page 2-8

Reference

- *Run configuration - Profiling tab* on page 2-6
- *The Timeline view* on page 2-10
- *The Call Paths view* on page 2-13
- *The Code view* on page 2-15

2.2 Creating a Streamline-enabled configuration

If you have not already set up an Eclipse workspace, either create a new one or use the default location.

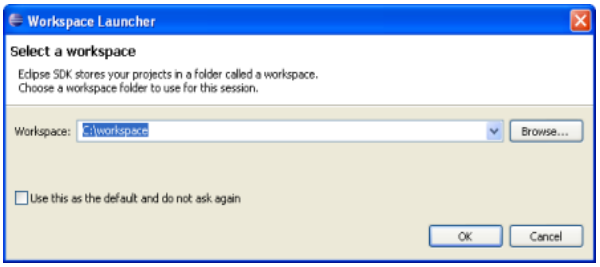

Figure 2-1 The workspace launcher

If this is your first time opening the workspace, Eclipse for DS-5 displays the welcome screen. There is a lot of good information here to explore, but to get started, click on the **Go to Workbench** link.

If you have used this workspace before, your workbench is already open.

To create a run configuration:

1. Make sure that you are in the DS-5 C/C++ perspective.
2. Select **Run** → **Run Configurations** from the menu.
3. Double-click on **ARM Streamline** in the list of available run configuration types on the left side of the window.

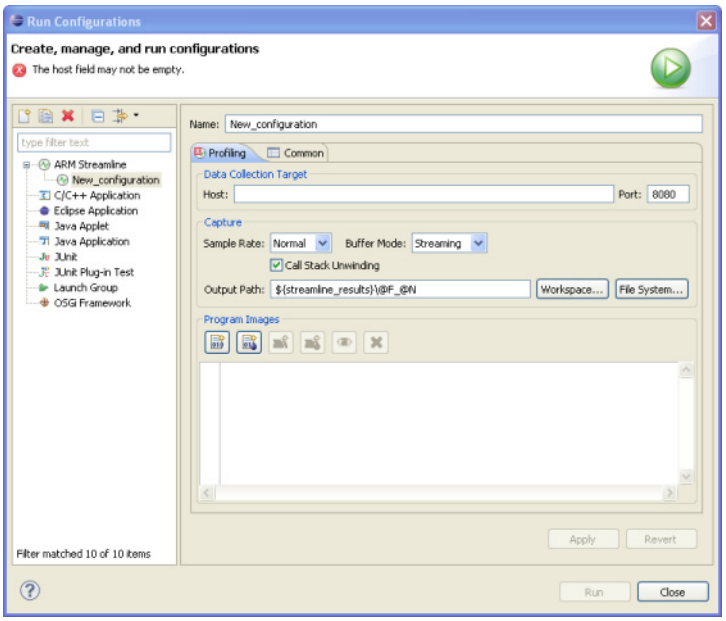

Figure 2-2 The run configuration window

4. Enter the configuration name in the Name field:

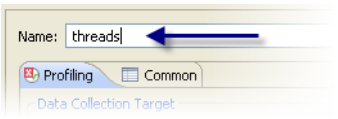

Figure 2-3 Naming the configuration

5. Enter the name or IP address of your target in the **Host** field.
6. Click **Add Program** button in the Program Images section.
7. Navigate to the threads example directory.
8. Select the threads executable.
9. Click **Open**.
10. Click **Apply** to save the settings.
11. Close the new configuration.

2.2.1 See also

Tasks

- *Setting up the gator driver and daemon* on page 2-2
- *Capturing and analyzing data for the threads example* on page 2-8

Reference

- *Run configuration - Profiling tab* on page 2-6
- *The Timeline view* on page 2-10
- *The Call Paths view* on page 2-13
- *The Code view* on page 2-15

2.3 Run configuration - Profiling tab

The **Profiling** tab of the run configuration dialog presents various options you can use to customize a profiling session.

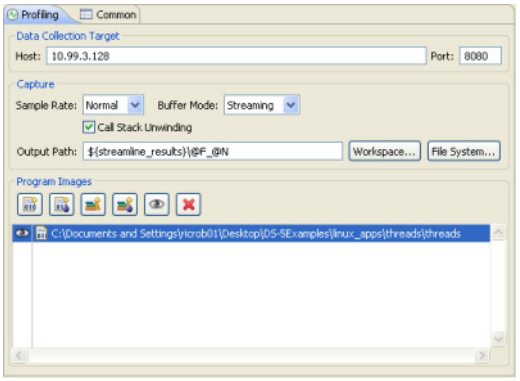

Figure 2-4 Other configuration options

The following settings are available on the **Profiling** tab of the Run Configuration dialog box:

Host The Host field defines the hardware target, either by name or by IP address.

Port The port on which your host collects data from the target.

Sample Rate

The **Normal** setting works well in most instances, but if you want to reduce the footprint of Streamline for performance reasons, select **Low**.

Buffer Mode

The default setting is unbounded **Streaming** of target data directly to your host. You can have the target fill either a **Normal**, **Large**, or **Small** sized buffer, that is then transferred to the host when collection is complete. Using buffers can improve performance as the target sends data to the host less frequently if one of the buffer options is active.

Call stack unwinding

Selecting this checkbox ensures that Streamline records call stacks, which greatly improves your visibility to the behavior of your target. Make sure to compile your EABI images and libraries with frame pointers using the `-fno-omit-frame-pointer` compilation option.

Output path

Use this field to define the directory location and name of the file generated by the capture and analysis session. By default, the file is saved to a results directory defined by an install variable and given the name `@F_@N.apd`. `@F` is a variable for the given configuration name, while `@N` is a sequential number. For example, if you titled your project `threads` and this is your first execution, the resulting capture file would be called `threads_001.apc` and analysis file `threads_001_001.apd`.

Program images

Use this area to explore your file system and define all of the images and libraries you want to profile. Attach libraries to a Program so that statistics are only recorded in that context. Use the **Add Program...** button to add images, and the **Add Library...** button to attach libraries to programs.

Note

When compiling images file, make sure to set the `-g` compilation option to enable debug symbols. Disable inlining with the `-fno-inline` compiler setting to improve the callpath quality.

2.3.1 See also

Tasks

- *Setting up the gator driver and daemon* on page 2-2
- *Creating a Streamline-enabled configuration* on page 2-4
- *Capturing and analyzing data for the threads example* on page 2-8

Reference

- *The Timeline view* on page 2-10
- *The Call Paths view* on page 2-13
- *The Code view* on page 2-15

2.4 Capturing and analyzing data for the threads example

After you have created a Streamline run configuration, you can capture, analyze, and view a report in Streamline. To do so:

1. Click the **Run** button at the bottom of the run configuration window to start the profiling session. If you have defined your target correctly, an analysis file appears in the ARM Streamline Data view with a **Stop** button.

Figure 2-5 ARM Streamline Data View: From working .apc file to .apd file

Note

Although .apd files are not compatible across different versions of Streamline, .apc files are. If you have upgraded Streamline and rendered your .apd files incompatible, re-run the .apc files and Streamline creates new, compatible .apd files based on the stored capture sessions.

2. During the collection of profiling data, switch to your target and open a terminal.
3. Navigate to the threads directory
4. Enter `./threads` on command line to execute the threads example on your target.
5. After you have run threads, return to your host machine and click the **Stop** button. This action processes the data and makes it ready for report viewing.

2.4.1 Viewing the .apd report

When the analysis completes, Streamline automatically opens the profiling report. To open the report again, double-click on the .apd in the Streamline Data view.

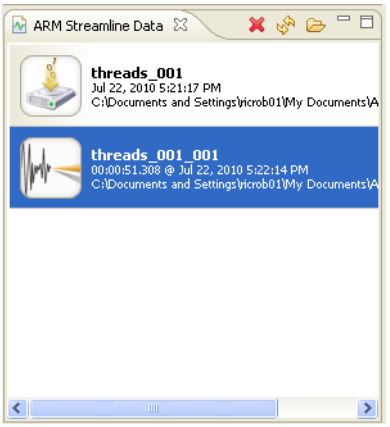

Figure 2-6 ARM Streamline Data view

To analyze the captured .apc data again with other settings, double-click on the captured data icon.

You can use the resutling settings dialog box to add or remove images and libraries, so that your final reports reflect only the execution you want to focus on.

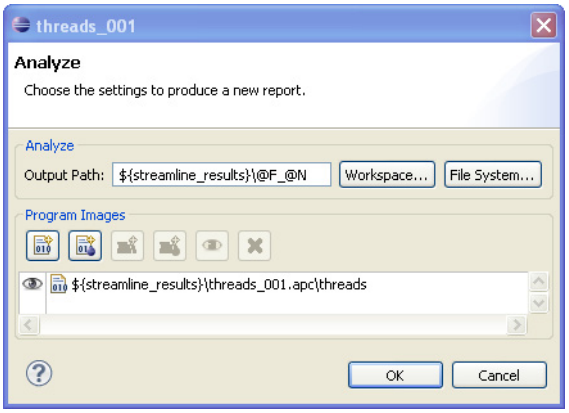

Figure 2-7 Changing the settings for a new report

2.4.2 See also

Tasks

- *Setting up the gator driver and daemon* on page 2-2
- *Creating a Streamline-enabled configuration* on page 2-4

Reference

- *Run configuration - Profiling tab* on page 2-6
- *The Timeline view* on page 2-10
- *The Call Paths view* on page 2-13
- *The Code view* on page 2-15

2.5 The Timeline view

After you have successfully generated a report, Streamline opens it automatically and displays the Timeline view.

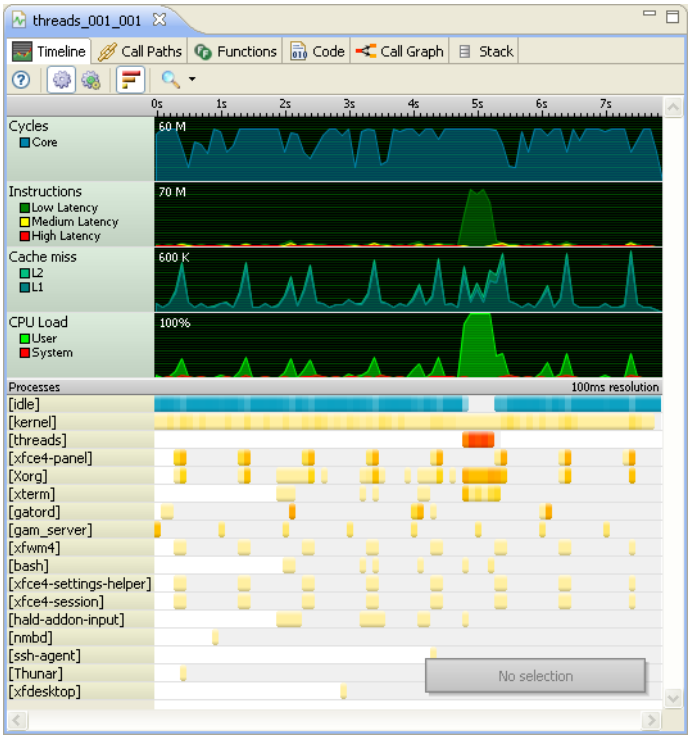

Figure 2-8 The Timeline view

2.5.1 Finding the threads execution window

The Timeline view shows you what the target was doing at every step. The bright red area in the **[threads]** row indicates that `threads` actively used a large percentage of time during this small window.

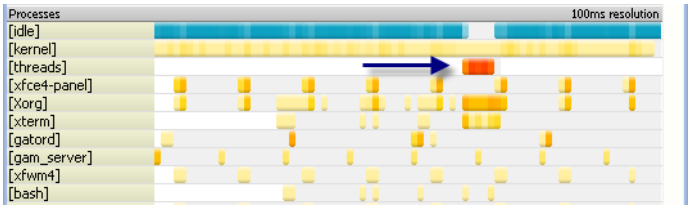

Figure 2-9 `threads` activity

2.5.2 Bins

The Timeline view breaks up its data into bins, a unit of time defined by the unit drop down menu at the top of the view. For example, if the unit is set to 100ms, every point in the charts and every color-coded bin in the processes section represents trace data captured during a 100ms window.

2.5.3 The marker

Clicking anywhere in the graphs places the marker at that spot. The marker is a vertical line that bisects the graphs and overlays the exact values of each of the graphs for that bin.

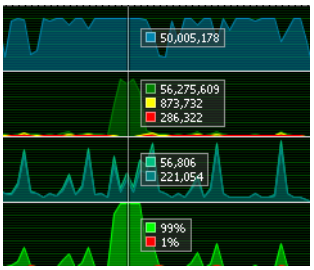

Figure 2-10 Timeline marker

2.5.4 Charts

At a glance, the Timeline view charts in our `threads_001_001.apd` example analysis file show that `threads` had a major impact on the Instruction and CPU workloads while it was active. Streamline collects data for the charts from hardware and software performance counter resources. The data is dependent on the type of system you use.

Figure 2-11 Timeline chart

Common graphs include:

CPU Load

The percentage of the CPU time spent in system or user code.

Instructions

A count of instructions executed.

Cache Miss

The number of cache misses in a particular cache level.

2.5.5 Processes

While the charts show you the overall performance of the system during profiling, the Processes section of the Timeline view shows you the active processes in each bin. The color code for the process and thread bars is:

White Inactive

Gray Active

Yellow to red

Responsible for some percentage of total instructions during this bin. Red indicates a higher percentage.

Notice that in the threads example, the bin is white, and all bins are bright red when `threads` is running.

2.5.6 Detail bars

The detail bars show the sampling points in the selected trace bin. Selecting a bar jumps you into the relevant context in the Call Paths view.

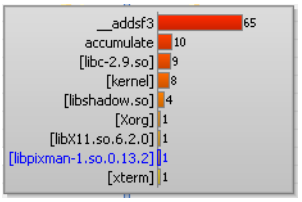

Figure 2-12 Timeline Detail Bar

2.5.7 See also

Tasks

- *Setting up the gator driver and daemon* on page 2-2
- *Capturing and analyzing data for the threads example* on page 2-8
- *Capturing and analyzing data for the threads example* on page 2-8

Reference

- *Run configuration - Profiling tab* on page 2-6
- *The Call Paths view* on page 2-13
- *The Code view* on page 2-15

2.6 The Call Paths view

Use the Call Paths view to see statistics for the whole system. The report presents data hierarchically. Expand the Processes and Threads to expose the Function children.

To get the best results from the Call Paths report, compile your programs and libraries with frame pointers, and check the call stack unwinding checkbox in the Run Configuration dialog box.

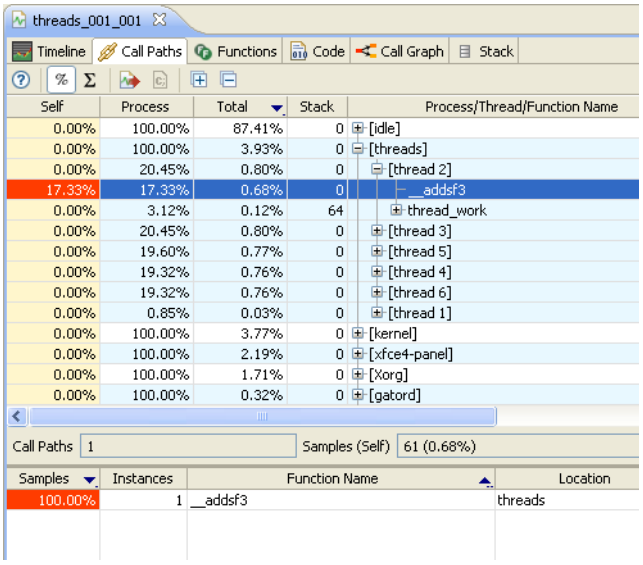

Figure 2-13 The Call Paths view

The Call Paths View shows you statistics for each process, thread, or function in the following categories:

Self Time Streamline measures time when it records the program callstack on interrupt events. On each interrupt, the sample is attributed to the function that was active.

The percentages are color-coded based on their value. Self Time does not include time spent in child functions, threads, or processes.

Total Time Total Time works the same as Self Time, only the values here represent the time spent in the function, thread or process and its children.

Stack The number of bytes used by the stack after the call of this function.

Process/Thread/Function Name

The name field also includes the disclosure control. Click the plus button to open a thread, process or function up to expose its children.

Location The Location field lists the name of the source file that contains function next to a line number so you can quickly look the function up.

Double-click on any item in the Call Paths view to open the functions view with the relevant function selected.

2.6.1 See also

Tasks

- *Setting up the gator driver and daemon* on page 2-2
- *Creating a Streamline-enabled configuration* on page 2-4

- *Capturing and analyzing data for the threads example* on page 2-8

Reference

- *Run configuration - Profiling tab* on page 2-6
- *The Timeline view* on page 2-10
- *The Code view* on page 2-15

2.7 The Code view

The Code view helps you discover function-level hot spots. Unlike the Call Paths view, which breaks data down by each function instance in the call tree, this report presents flattened data. You might find it useful to use the .apc re-analyze facility to change the images that Streamline includes in the reports. Doing so refines the Code view statistics so that you can focus on the application of interest.

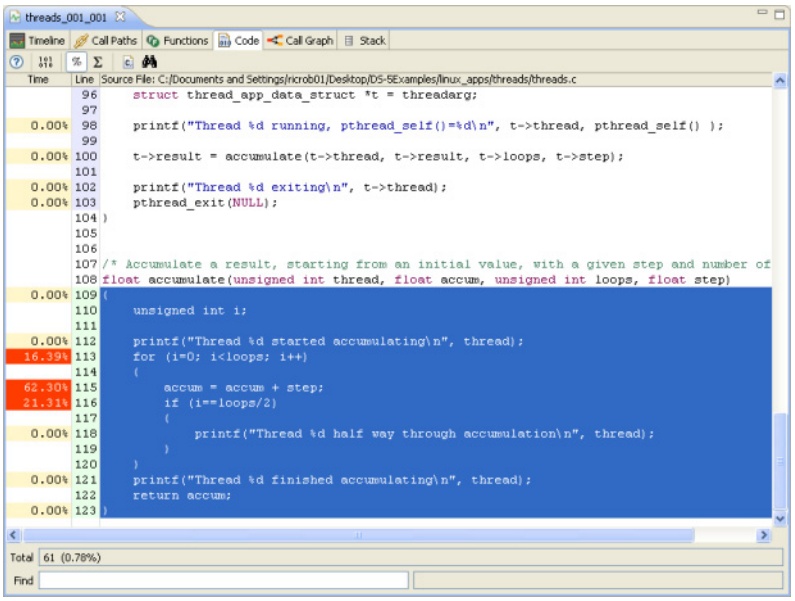

Figure 2-14 The Code view

Next to every line of code in source file, the Code view presents the percentage each line contributed to the total samples collected for the function. Use the totals panel at the bottom of the Code view to see accumulated statistics.

Figure 2-15 The Totals panel

Note

Press **F1** (or **SHIFT + F1** in Linux) in any open analysis window to display contextual help.

2.7.1 See also

Tasks

- *Setting up the gator driver and daemon* on page 2-2
- *Creating a Streamline-enabled configuration* on page 2-4
- *Capturing and analyzing data for the threads example* on page 2-8

Reference

- *Run configuration - Profiling tab* on page 2-6
- *The Timeline view* on page 2-10
- *The Call Paths view* on page 2-13

Chapter 3
Troubleshooting

If you're having trouble running Streamline, consult this collection of solutions to common problems.

The following topics describe how to troubleshoot common Streamline issues:

3.1 Target connection issues

Each of the error messages provided by Streamline on a connection failure indicates a different issue.

Symptom You receive the following error message: `Unable to connect to the gator daemon at` *`your_target_address`*`. Please verify you have installed the gator daemon on your target and it is running. Installation instructions can be found in:` *`DS-5 root`*`/arm/README_Streamline.txt`

Solutions Make sure the gator daemon is running on your target. Enter the following command in the shell of your target:

```
ps -d | grep gatord
```

If this command returns no results, `gatord` is not active. Start it by navigating to the directory that contains gatord and entering the following command:

```
sudo ./gatord &
```

Re-try connecting to the target.

If gator is active and you still receive the above error message, try disabling any firewalls on your host machine that might be interfering with communication between it and the target.

Symptom You receive the following error message: `Unknown host`

Solution Make sure that you've correctly entered the name or IP address of the target in **Host** field. If you've entered a name, try an IP address instead.

Symptom You receive the following error message: `Unable to launch. Only one instance of Streamline may be running on this machine. Please close all instances of Eclipse and try again.`

Solution Stop any other running Streamline session and re-try connecting to the target. If you can't find another session, try closing DS-5 for Eclipse, then re-starting it.

3.1.1 See also

Reference

- *Report issues* on page 3-3

3.2 Report issues

If the data in your reports seems incomplete, it could indicate that you did not include a compilation option essential to Streamline. Consult the following common report problems and solutions.

Symptom Streamline does not show any source code in the Code view.

Solution Make sure that you used the `-g` option during compilation. Streamline must have debug symbols turned on to match instructions to source code.

Symptom Streamline does not show source code for shared libraries.

Solution Add the libraries using the run configuration dialog. Press the **Add Library...** button in the images section, navigate to your shared library, and then add it.

Symptom The data in the call paths view is flat. The presented table is a list, rather than a hierarchy.

Solution Use the `-f-no-omit-frame-pointer` option during compilation and be sure to check the **Call Stack Unwinding** option in the run configuration dialog.

Note

Streamline does not walk the stack for kernels or statically linked drivers. Both generate flat data in the call paths view.

Symptom Functions that you know are highly used are missing from the reports. Other functions might seem artificially large.

Solution This can be due to code inlining done by the compiler. Turn inlining off by adding `-fno-inline` as an option during compilation.

3.2.1 See also

Reference

- *Target connection issues* on page 3-2