ARM® DS-5

Version 6.2

Streamline User Guide

ARM

ARM 100769_0602_00_en

ARM® DS-5

Streamline User Guide

Release Information

Document History

Issue	Date	Confidentiality	Change
A	19 October 2010	Non-Confidential	ARM Streamline Performance Analyzer 1.0
B	19 January 2011	Non-Confidential	Update for DS-5 version 5.4
C	19 April 2011	Non-Confidential	Update for DS-5 version 5.5
D	19 July 2011	Non-Confidential	Update for DS-5 version 5.6
E	19 September 2011	Non-Confidential	Update for DS-5 version 5.7
F	19 November 2011	Non-Confidential	Update for DS-5 version 5.8
G	19 February 2012	Non-Confidential	Update for DS-5 version 5.9
H	19 May 2012	Non-Confidential	Update for DS-5 version 5.10
I	19 July 2012	Non-Confidential	Update for DS-5 version 5.11
J	19 October 2012	Non-Confidential	Update for DS-5 version 5.12
K	19 December 2012	Non-Confidential	Update for DS-5 version 5.13
L	15 March 2013	Non-Confidential	Update for DS-5 version 5.14
M	14 June 2013	Non-Confidential	Update for DS-5 version 5.15
N	13 September 2013	Non-Confidential	Update for DS-5 version 5.16
O	13 December 2013	Non-Confidential	Update for DS-5 version 5.17
P	14 March 2014	Non-Confidential	Update for DS-5 version 5.18
Q	27 June 2014	Non-Confidential	Update for DS-5 version 5.19
R	17 October 2014	Non-Confidential	Update for DS-5 version 5.20
S	20 March 2015	Non-Confidential	Update for DS-5 version 5.21
T	15 July 2015	Non-Confidential	Update for DS-5 version 5.22
U	15 October 2015	Non-Confidential	Update for DS-5 version 5.23
V	15 March 2016	Non-Confidential	Update for DS-5 version 5.24
W	31 May 2016	Non-Confidential	Update for DS-5 version 5.25.0.1
X	11 August 2016	Non-Confidential	Update for DS-5 version 5.25.0.2
0600-00	30 September 2016	Non-Confidential	Update for Streamline version 6.0. Document numbering scheme changed.
0601-00	04 November 2016	Non-Confidential	Update for Streamline version 6.1
0602-00	10 March 2017	Non-Confidential	Update for Streamline version 6.2

Non-Confidential Proprietary Notice

ARM Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

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.

Unrestricted Access is an ARM internal classification.

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 Streamline User Guide

List of Figures
ARM® DS-5 Streamline User Guide

List of Tables
ARM® DS-5 Streamline User Guide

Preface

This preface introduces the *ARM® DS-5 Streamline User Guide*.

It contains the following:

- *About this book* on page 13.

About this book

ARM® DS-5 Streamline User Guide. Provides instructional documentation for real-time, non-intrusive analysis of applications.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction to Streamline Performance Analyzer

ARM® Streamline Performance Analyzer is a system-wide visualizer and profiler for hardware targets that have an ARMv7-A or ARMv8-A architecture core.

Chapter 2 Target Setup

To get started using Streamline, you must ensure that gator, the mechanism that Streamline uses to communicate with your target, is running on the target. You can either build and install gator yourself, or you can use a pre-built version. This chapter describes how to set up your target device for Streamline.

Chapter 3 Streamline Data View

Describes how to use the **Streamline Data** view to select a target, configure, start, and stop a capture session, and manage your existing captures.

Chapter 4 Capture and Analysis Options

Describes how to use the **Capture & Analysis Options** dialog box to change capture session settings, such as duration, sample rate, and buffer size.

Chapter 5 Counter Configuration

Describes how to use the **Counter Configuration** dialog box to select the events that Streamline collects.

Chapter 6 Live View and Timeline View

Describes the **Live** and **Timeline** views, which display charts showing the data collected during the capture session. **Live** view is displayed while the capture takes place, and charts the data in real time. **Timeline** view is displayed after the capture session ends and the data has been analyzed. It provides additional information in a details panel.

Chapter 7 Table Views: Call Paths and Functions

Describes the **Call Paths** and **Functions** views, which provide tabular data about the capture.

Chapter 8 Code View

Describes the **Code** view, which provides statistics for lines of source code and for disassembled instructions.

Chapter 9 Streamline Annotate

Describes the Streamline Annotate feature. It enables you to add annotations to your code, which are propagated into the **Timeline** and **Log** views.

Chapter 10 Log View

Describes the **Log** view, which lists the annotations generated in your code along with information about them.

Chapter 11 Capturing Energy Data

Describes how to set up and use the ARM Energy Probe with Streamline to view the power metrics of code running on target hardware.

Chapter 12 Advanced gator Customizations

Describes how to customize the more advanced collection and reporting features of Streamline.

Chapter 13 Bare-Metal Support

Describes the bare-metal support available within Streamline.

Chapter 14 Profiling with System Trace Macrocell

Describes the collection of profiling data using System Trace Macrocell (STM).

Chapter 15 Troubleshooting Common Streamline Issues

Describes how to troubleshoot some common Streamline issues.

Chapter 16 Using Streamline on the Command Line

Describes how to use the `streamline` command to access much of the functionality of Streamline from the command line.

Glossary

The ARM Glossary is a list of terms used in ARM documentation, together with definitions for those terms. The ARM Glossary does not contain terms that are industry standard unless the ARM meaning differs from the generally accepted meaning.

See the *ARM Glossary* for more information.

Typographic conventions

italic
: Introduces special terminology, denotes cross-references, and citations.

bold
: Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

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

`mono`space
: Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

`monospace italic`
: Denotes arguments to monospace text where the argument is to be replaced by a specific value.

`monospace bold`
: Denotes language keywords when used outside example code.

`<and>`
: Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example:

```
MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>
```

SMALL CAPITALS
: Used in body text for a few terms that have specific technical meanings, that are defined in the *ARM Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Feedback

Feedback on this product

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

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

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

- The title *ARM DS-5 Streamline User Guide*.
- The number ARM 100769_0602_00_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Note

ARM tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

Other information

- *ARM Developer*.
- *ARM Information Center*.
- *ARM Technical Support Knowledge Articles*.
- *Support and Maintenance*.
- *ARM Glossary*.

Chapter 1
Introduction to Streamline Performance Analyzer

ARM® Streamline Performance Analyzer is a system-wide visualizer and profiler for hardware targets that have an ARMv7-A or ARMv8-A architecture core.

Streamline samples the PC address at regular intervals to generate a profile of where the processor spends most of the time. It also interrogates PMU counters on the device so that you can get counter data from the target.

This chapter provides an overview of how Streamline works, and details of the new features and updates you can find in this release.

It contains the following sections:

1.1 Overview of Streamline Performance Analyzer

Streamline Performance Analyzer is software that gathers data on the performance of a target device. Streamline collects this data by using gator or barman, and then generates an analysis report.

gator consists of `gatord` and, optionally, `gator.ko`. `gatord` is a daemon that must be installed and running on the target for Streamline to be able to communicate with it. `gator.ko` is a Linux kernel driver module that collects data from the operating system and applications running on the target. `gatord` reads and processes this data and creates an `.apc` directory containing the capture data. When gator consists of `gatord` and `gator.ko`, it is called kernel space gator. If `gator.ko` is not present, gator collects performance data using `perf`, and is called user space gator. User space gator does not have all of the functionality of kernel space gator.

Barman is a bare-metal agent. It allows you to visualize elements of the system state of targets without connected trace devices or operating systems for which `gator` is provided. Barman consists of two C source files that you build into the executable on the target. The data that barman collects is stored in a RAM buffer, from which you can extract it and import it into Streamline.

By default, Streamline uses the best-fit set of hardware performance counters for your target. These counters can be modified in the **Counter configuration** dialog, which lists the counters available for your target hardware. The counters are plotted in a series of charts. The default charts are **CPU Activity**, **Cache**, **Clock**, **Disk I/O**, **Instruction**, **Interrupts**, and **Memory**. Target-specific charts, for example charts for Mali-based targets, can also be displayed. The charts aggregate the data from all cores, however they can be expanded to display individual charts for each core.

The following figure shows the Streamline interface with the default charts displayed in the **Timeline** view.

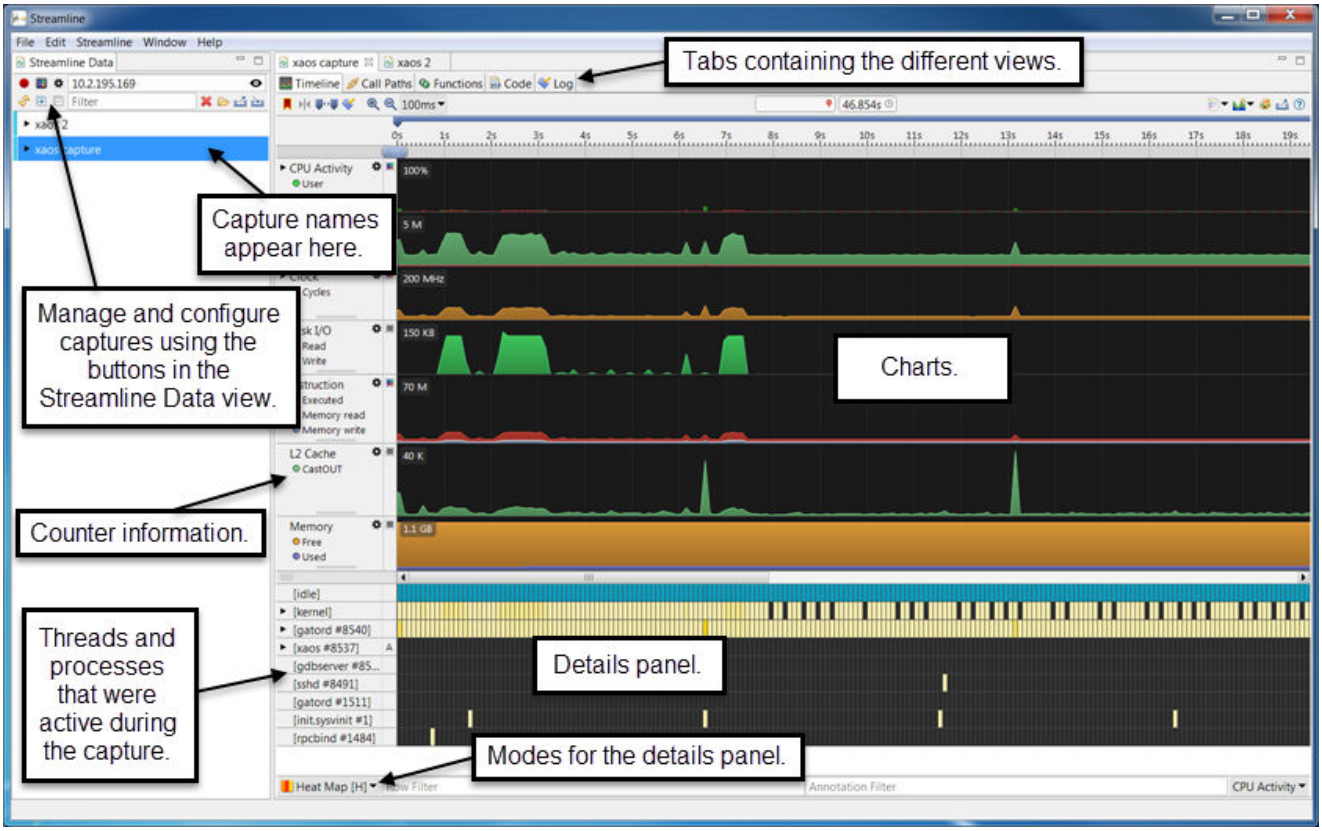


Figure 1-1 Streamline graphical user interface

Samples are collected for the counters at your chosen intervals, which you specify with **Sample Rate** in the **Capture & Analysis Options** dialog. Selecting event-based sampling overrides the default sampling

to sample on context switches and when an event has been triggered the threshold number of times. You can focus on specific data or time intervals by using the cross section marker, or the calipers in the **Live** and **Timeline** views.

Streamline contains various different views:

Streamline Data
: Edit capture options and counter configurations, start and stop new captures, and manage existing ones. You can also import captures from `.apc` directories, or perf recordings.

Live
: Opens when a capture is started, and plots the counters in real time.

Timeline
: Displays high-level information in charts and a details panel. You can customize the charts and create templates to quickly apply these customizations in future. There are a range of modes for the details panel. Some modes are color-coded to indicate levels of activity caused by threads and processes. Other modes list functions and processes that are active in the current cross section. There is also an **OpenCL** mode for use with Mali Midgard targets, and an **Images** mode for displaying images if visual annotations are in the capture.

Call Paths
: Displays a hierarchy of functions, threads, and processes. Disclosure controls on each function, thread, and process allow you to show more of the hierarchy by displaying the functions and threads that they call.

Functions
: Lists all the functions that were called during a capture, alongside data about their usage.

Code
: Displays statistics for lines of the source code and, optionally, the disassembled instructions.

Log
: Lists the annotations that were generated in the code and information about them. This information includes when and where the annotation was generated, along with the content, and any groups or channels it belongs to.

Annotations allow you to add further context to the target information that Streamline gathers. For example, string annotations display text in the details panel of the **Timeline** view. Bookmark annotations allow you to easily return to points of interest in the **Timeline** and **Log** views. All annotations added are integrated into the capture report.

You can gather GPU-specific profiling data for Mali-based devices with Streamline. The analysis reports then gain a significant amount of data about the graphical performance of your target.

Examples of different aspects of the functionality of Streamline are in the following locations:

- *DS-5_install_directory*`/examples/Linux_examples.zip`
- *DS-5_install_directory*`/sw/streamline/examples`

1.2 New features and updates

The following new features and updates are available in this release of Streamline:

Table 1-1 New features

Feature	Summary	Documentation
Images download.	Download process images from the target device at the end of a live capture session. Use the **Image Download** dialog to select the images to download.	See *6.5 Image download* on page 6-94.
Profiling with System Trace Macrocell.	An extension to the bare-metal support that allows the collection of profiling data using STM. You can import STM traces from the **Barman Generator Wizard** in Streamline, or from the command-line.	See *Chapter 14 Profiling with System Trace Macrocell* on page 14-201, *16.6 Importing an STM trace from the command-line* on page 16-223.
Non-root support.	Run `gatord` on a device without root permissions. For Android targets, an Android daemon application is supplied.	See *2.4.1 Comparison of user space gator and kernel space gator* on page 2-24, *2.10.1 Installing and using the Android daemon application* on page 2-35.
Export **Heat Map**.	Export the data contained in the **Heat Map** from the GUI or the command-line.	See *6.6.2 Heat Map mode* on page 6-97, *16.4 Exporting the Heat Map from the command-line* on page 16-221.

Table 1-2 Updates

Update	Documentation
Bare Metal menu renamed to **Streamline**. **Generate Agent Sources** dialog renamed to **Barman Generator Wizard.**.	Screenshots updated to show this.
The **Counter Configuration** dialog has been improved.	See *Chapter 5 Counter Configuration* on page 5-67.
Command line options `-pmus` and `-events` added to documentation.	See *16.2 Streamline command-line options* on page 16-217.

Chapter 2
Target Setup

To get started using Streamline, you must ensure that gator, the mechanism that Streamline uses to communicate with your target, is running on the target. You can either build and install gator yourself, or you can use a pre-built version. This chapter describes how to set up your target device for Streamline.

It contains the following sections:

2.1 Features of Streamline Ultimate, Professional, and Community Editions

Streamline is available in DS-5 Ultimate, Professional, and Community Editions.

Community Edition has the following restrictions:

- Only the **Timeline** and **Functions** views are available. All other views are only available in the Ultimate and Professional Editions.
- Bare-metal support is not available.
- A limited set of hardware performance counters is available.
- Only one image can be selected at a time for analysis.
- It supports Mali GPU activity, but does not support **OpenCL** mode.
- The following features are also not supported:
  - Per-core views.
  - **Core map** mode, **Cluster map** mode, and **Samples** mode.
  - Energy capture.
  - Custom activity maps.

See the *DS-5 Editions* page for an up-to-date list of Community, Professional, and Ultimate Edition features. See the *DS-5 Customized Editions* page for more information about the features supported by customized toolkit editions.

Note

- Community Edition, Professional Edition, and Ultimate Edition support ARMv8 targets. If you open a capture generated on an ARMv8 target and do not have a valid license for ARMv8 features, Streamline displays a warning and opens a Community Edition version of the report for that capture.
- If your Eclipse for DS-5 installation supports switching between toolkit licenses, you must restart Streamline for the new license to take effect.
- You cannot open a report that was generated using a different toolkit license. The report must first be re-analyzed using the new license.

2.2 Standards compliance in Streamline

Lists the levels of compliance that Streamline conforms to.

ELF

Streamline can read executable images in ELF format.

Note

Streamline can scan Android application package file (APK) archives for ELF images. It extracts the valid ELF executable images and includes them as Program Images for analysis.

DWARF

Streamline can read debug information from ELF images in the DWARF 2, DWARF 3, and DWARF 4 formats.

Note

The DWARF 2 and DWARF 3 standards are ambiguous in some areas such as debug frame data. This means that there is no guarantee that the debugger can consume the DWARF produced by all third-party tools.

2.3 Streamline prerequisites

Before configuring your target, ensure that you have the following prerequisites for running Streamline:

- A hardware target with an ARMv7-A or ARMv8-A architecture core. Streamline is supported on all ARMv7-A and ARMv8-A cores, and is tested on ARM Cortex®-A cores.
- The Linux kernel source code for the target platform. In DS-5 version 5.25, Streamline supports gator versions 17-25. gator versions 22 and later only support Linux kernel versions 3.4 and later. gator versions 21 and earlier support Linux kernel versions 2.6.32 and later.
  You need the Linux kernel source code for either of the following reasons:
  - If the Linux kernel is not properly configured for Streamline, you must enable the required kernel configuration menu options and rebuild it.
  - If you want to use kernel space gator. This provides more features than user space gator, but requires you to build the kernel module, `gator.ko`.
- GCC to build the Linux kernel or `gator.ko` natively on the target, or to cross compile it on a Linux host. You can obtain GCC at the Linaro website: *http://www.linaro.org*, or, depending on the GCC version you require, from the DS-5 installation folder.

If the kernel is properly configured and you are using user space gator, you do not need the Linux kernel source code or GCC. User space gator is supported for Linux kernel versions 3.4 and later.

Related tasks

2.5 Preparing and building your Linux kernel on page 2-28.
2.8 Building the gator module on page 2-31.
2.9 Running the gator daemon on your target on page 2-33.
2.11 Starting a capture session on page 2-37.

Related references

2.6 Required kernel configuration menu options on page 2-29.

2.4 Setup scenarios for Linux targets

The purpose of these scenarios is to make it easier for you to set up Streamline and perform a capture for the first time.

Before you can use Streamline to profile a Linux or Android target, you need to decide whether to use user space gator or kernel space gator. You also need to check that the target is configured correctly and that gator is installed and running on the target.

See also `README.md`, located in *`DS-5_install_directory`*`/sw/streamline/gator/` for key information about setting up the target.

2.4.1 Comparison of user space gator and kernel space gator

Streamline requires an agent called gator to be installed and running on the target.

gator consists of the following components:

- A daemon, `gatord`.
- Optionally, a Linux kernel driver module, `gator.ko`.

The role of `gator.ko` is to collect data from the operating system and applications that are running on the target. `gatord` reads and processes this data, and creates a directory, whose name ends in `.apc`, containing the capture data.

`gatord` must be installed and running on the target for Streamline to communicate with the target, but `gatord` can run with or without `gator.ko`. gator can operate in the following modes:

Kernel space gator
When `gatord` is launched, it inserts `gator.ko`, if it exists in the same directory as `gatord`, into the Linux kernel. When `gatord` is used together with `gator.ko`, this is referred to as *kernel space gator*.

User space gator
When `gatord` runs without `gator.ko`, this is referred to as *user space gator*. User space gator is supported on Linux kernel versions 3.4 and later. You can run user space gator on a device with or without root permissions:

Root user space gator
When you run `gatord` on a device with root permissions, this is called *root user space gator*. When this document uses the term *user space gator*, it is referring to *root user space gator*.

Non-root user space gator
When you run `gatord` on a device without root permissions, this is called *non-root user space gator*. In this case, `gator.ko` and `perf` are not required. However, the following limitations apply:

- The available counters are limited to the basic process and system statistics. For example, **CPU Activity** and **Memory**.
- Counters are collected at a lower frequency.
- **CPU Activity** and **Heat Map** display approximate data.
- No profiling information is available, which means the **Call Paths** and **Functions** views are empty.

In the absence of `gator.ko`, user space gator collects most of the performance data using `perf`. User space gator is restricted to using user space APIs and does not support the following features that kernel space gator supports:

- Call stack unwinding.
- Mali GPU activity, although it does support OpenCL.
- `CPU I/O: Wait` and `Idle: State` Linux counters.
- Access to L2C-310 counters.

User space gator has some other restrictions, for example:

- It polls the following Linux counters every 100ms, instead of every 1ms or when they change because files in the `/proc` or `/sys` filesystem are read:
  — Memory.
  — Disk IO.
  — Network.

  This rate is fixed and overrides the sample rate that is specified in the **Capture & Analysis Options** dialog.
- When using user space gator, the `Memory: Used` counter does not contain per-process information. As a result, memory statistics are not available in **Processes** mode.

Related concepts

6.6.7 Processes mode on page 6-102.

2.4.2 Validating the target setup

There are various commands that you can run on your target to test whether it is set up correctly.

- To test whether the Linux kernel is properly configured to work with gator, check your kernel configuration file. For example, if `/proc/config.gz`, exists on your system, use the following command to confirm whether `CONFIG_PROFILING` is enabled:

  ```
  zcat /proc/config.gz | grep CONFIG_PROFILING
  ```
- Check that the `gatord` process is running on the target. If not, Streamline reports an error when you try to start a capture.
- The version of gator running on the target must be compatible with the version of Streamline you are using. You can use the following command to print the version number of `gatord`:

  ```
  ./gatord -v
  ```

  If you are using kernel space gator, you can use:

  ```
  cat /dev/gator/version
  ```

  Streamline also displays the gator version number in the **Connection Browser** dialog.

  ———— **Note** ————

  This version of Streamline supports gator protocol versions 17 and later. ARM recommends that you use the version of gator that matches the version of Streamline you are using.

  ————————————
- To use kernel space gator, `gator.ko` must be installed on the target. To check whether it is installed, use the following command:

  ```
  lsmod | grep gator
  ```

  To help you to identify captures that were performed using user space gator, Streamline displays this icon on the left side of the toolbar in the **Live** and **Timeline** views: 

Related concepts

2.4.3 Building and installing user space gator on page 2-26.
2.4.4 Building and installing kernel space gator on page 2-26.

Related tasks

2.5 Preparing and building your Linux kernel on page 2-28.

Related references

2.6 Required kernel configuration menu options on page 2-29.
3.6 Connection Browser dialog box on page 3-53.

2.4.3 Building and installing user space gator

If you want to use user space gator, the gator daemon, `gatord`, must be installed and running on the target.

If `gatord` is not already installed on the target, the simplest way to install it is to use the pre-built statically-linked `gatord` binary. You can automatically install and run this on a Linux or Android target by clicking **Setup target...** in the **Connection Browser** dialog, specifying the target name, your user name and, if required, a password. If an older version of `gatord` is already running on the target, this automatically kills it and replaces it.

Two pre-built `gatord` binaries are available. One for ARMv7 targets, and ARMv8 targets that support AArch32 execution state, and one for ARMv8 AArch64 targets. You cannot install them on a target that is based on another ARM architecture using the **Setup Target...** dialog. If you want to do this, or if the supplied `gatord` binary does not work correctly on your system, you must build, install, and run it manually.

The source code for `gatord` is available from the following locations:

- *`DS-5_install_directory`*`/sw/streamline/gator/daemon/`.
- *https://github.com/ARM-software/gator*. This site is the official distribution channel for all gator releases, and contains the latest source updates between DS-5 releases.

To build `gatord`, follow the instructions in *2.7 Building the gator daemon* on page 2-30. To install and run it on the target, see *2.9 Running the gator daemon on your target* on page 2-33. If an existing version of `gatord` is running, you first need to kill it, see *2.12 Stopping the gator daemon* on page 2-38.

Note

For Android targets, root permissions are required. If you do not have root permissions, you must use the APK in *`DS-5_install_directory`*`/sw/streamline/android`.

Related tasks

2.7 Building the gator daemon on page 2-30.
2.9 Running the gator daemon on your target on page 2-33.
2.12 Stopping the gator daemon on page 2-38.

Related references

3.6 Connection Browser dialog box on page 3-53.

2.4.4 Building and installing kernel space gator

If you want to use kernel space gator, `gatord` must be running on the target, and in addition, the gator driver, `gator.ko`, must be loaded into the Linux kernel on the target.

If `gator.ko` is not present on the target, you need to build and install it yourself. The source code is available from either of the following locations:

- *`DS-5_install_directory`*`/sw/streamline/gator/driver/`
- *https://github.com/ARM-software/gator*

You also need the source code for the Linux kernel that is running on your target. If possible, use the same toolchain to build `gator.ko` as was used to build the kernel.

You can build `gator.ko` in either of the following ways:

- Integrate `gator.ko` into the Linux kernel build system. This involves the following steps:
  1. Copy the gator driver source into the kernel source tree, for example *`<path_to_kernel_build_dir>`*`/drivers/gator/`.
  2. Add references to gator in the files `Makefile` and `Kconfig`, located in the parent directory, for example *`<path_to_kernel_build_dir>`*`/drivers/`.

3. Launch `menuconfig` and select `GATOR` to configure the required kernel configuration options.
4. Rebuild the kernel.

- Use a `make` command, then load `gator.ko` into the kernel in one of the following ways:
  - Copy `gator.ko` into the same directory on the target as `gatord`. When you run `gatord`, it automatically inserts `gator.ko` into the kernel.
  - Copy `gator.ko` into a different directory on the target to `gatord`. When you run `gatord`, specify the location of `gator.ko` using the `-m` option to automatically insert `gator.ko` into the kernel.
  - Manually insert `gator.ko` yourself using `insmod`.

If you want to replace an existing kernel space gator installation running on the target, for example to upgrade it to a newer version, you first need to kill and remove `gatord`. In addition, you need to unload and rebuild `gator.ko` to ensure it matches the `gatord` version.

See also `README.md`, located in *`DS-5_install_directory`*`/sw/streamline/gator/`.

Related tasks

2.8 Building the gator module on page 2-31.
2.12 Stopping the gator daemon on page 2-38.

2.5 Preparing and building your Linux kernel

Streamline requires that you build your Linux kernel with certain options enabled. These instructions are specific to building a Linux kernel. Ignore these steps if you are running Android.

The options that you need to enable are described in the topic *2.6 Required kernel configuration menu options* on page 2-29.

To prepare your kernel for use with Streamline, follow these steps:

Procedure

1. Download one of the supported versions of the Linux kernel and configure it. See the topic *2.3 Streamline prerequisites* on page 2-23 for the list of supported versions.

   For instructions on how to do this and the required kernel code, visit *http://www.kernel.org*.

   Note

   You can build and configure the gator driver `gator.ko` by copying the gator driver source directly into the Linux kernel source tree. On Linux, this code is located in the directory `/usr/local/DS-5/sw/streamline/gator/driver/`. For more information, see the topic *2.8 Building the gator module* on page 2-31.

2. Enter the following command in your shell to export the cross compiler:

   ```
   export CROSS_COMPILE=<cross_compiler_directory>/bin/arm-linux-gnueabihf-
   ```

3. To specify that this build is for an ARM architecture, enter the following command in your shell:

   ```
   export ARCH=arm
   ```

4. Enter the following to build the configuration file specific to your platform:

   ```
   make platform_defconfig
   ```

   Replace *platform_defconfig* in the command with one of the configuration files located in the `your_kernel/arch/arm/configs` directory appropriate for your platform or with a configuration file provided by a vendor.

5. To launch `menuconfig`, the command-line kernel configuration tool, enter the following in your shell:

   ```
   make menuconfig
   ```

6. Set the required kernel configuration menu options.

7. Use the following command to build the kernel image:

   ```
   make -j5 uImage
   ```

   Depending on your target system, you might need to generate the uImage file with a device tree blob, for example:

   ```
   make -j5 dtbs uImage
   ```

   The `uImage` should be installed and booted before moving on to the next step.

8. Verify all of your kernel options on a running system using `/proc/config.gz`, if it exists on your system. For example, to confirm that `CONFIG_PROFILING` is enabled, enter:

   ```
   zcat /proc/config.gz | grep CONFIG_PROFILING
   ```

Related tasks

2.8 Building the gator module on page 2-31.

Related references

2.6 Required kernel configuration menu options on page 2-29.
2.3 Streamline prerequisites on page 2-23.

2.6 Required kernel configuration menu options

Whether you are running Linux or Android on your target, you must enable certain kernel configuration options to run Streamline.

The following `menuconfig` menus have options that are required for Streamline:

Note

- If these options are not set correctly, you must change them and rebuild your kernel. If they are set correctly, you are ready to build and install the gator driver.
- The location of these options might change between releases. If so, use the search option in `menuconfig` to find them.
- Additional options are required to enable Mali GPU support.

General Setup

Enable the **Profiling Support** option `CONFIG_PROFILING`, and the **Kernel performance events and counters** option `CONFIG_PERF_EVENTS`. `CONFIG_PERF_EVENTS` is required for kernel versions 3.0 and later. Enable the **Timers subsystem** > **High Resolution Timer Support** option `CONFIG_HIGH_RES_TIMERS`. Optionally enable the **Enable loadable module support** option `CONFIG_MODULES`, and the **Module unloading** option `MODULE_UNLOAD`. These two options are only required if the gator driver is not built into the kernel. They are not needed for user space gator.

Kernel Features

The **Enable hardware performance counter support for perf events** option `CONFIG_HW_PERF_EVENTS`. `CONFIG_HW_PERF_EVENTS` is required for kernel versions 3.0 and later. If you are using Symmetric MultiProcessing (SMP), enable the **Use local timer interrupts** option `CONFIG_LOCAL_TIMERS`. The `CONFIG_LOCAL_TIMERS` option is not necessary if you are running on Linux version 3.12 or later.

CPU Power Management

Optionally enable the **CPU Frequency scaling** option `CONFIG_CPU_FREQ` to enable the CPU Freq **Timeline** view chart. gator requires kernel version 2.6.38 or greater to enable this chart.

Kernel hacking

The **Trace process context switches and events** option `CONFIG_ENABLE_DEFAULT_TRACERS` might not be visible in `menuconfig` as an option if other trace configuration options are enabled. Enabling one of these other trace configurations, for example `CONFIG_GENERIC_TRACER`, `CONFIG_TRACING`, or `CONFIG_CONTEXT_SWITCH_TRACER`, is sufficient to enable tracing. Optionally enable the **Compile the kernel with debug info** option `CONFIG_DEBUG_INFO`. This is only required for profiling the Linux kernel.

Related tasks

2.14 Setting up Streamline to support an ARM® Mali™-based device on page 2-41.

2.7 Building the gator daemon

To communicate with the target device, Streamline requires the gator daemon, `gatord`, to be running on the device.

Streamline includes a pre-built `gatord` binary, which you can install and run on a Linux or Android target by clicking **Setup target...** in the **Connection Browser** dialog. Alternatively, you can build `gatord` yourself following the steps outlined in this topic. For more information, see `README.md`, located in *`DS-5_install_directory`*`/sw/streamline/gator/`.

If you want to build `gatord` for a Linux target, you must have a `g++`-enabled build host toolchain. If the target is `g++`-enabled, you can build directly on it. On a Linaro Ubuntu target, enter the following command to install `g++`:

```
sudo apt-get install g++
```

If you want to build `gatord` for an Android target, you must first install the Android NDK. For information, see the Android NDK website, *http://developer.android.com/sdk/ndk*.

Note

It is not possible to build `gatord` on a Windows host.

To build `gatord`, follow these steps:

Procedure

1. Either download the `gatord` source from *https://github.com/ARM-software/gator*, or copy the source supplied in *`DS-5_install_directory`*`/sw/streamline/gator/daemon/`.
2. Change to the gator `daemon` directory by using either of the following commands:
   - For Linux, enter:

     ```
     cd daemon
     ```
   - For Android, enter:

     ```
     mv daemon jni
     ```
3. Issue the commands to build `gatord`.
   - To build `gatord` for an ARMv7 Linux target, enter:

     ```
     make CROSS_COMPILE=<cross_compiler_directory>/bin/arm-linux-gnueabihf-
     ```
   - To build `gatord` for an ARMv8 Linux target, enter:

     ```
     make -f Makefile_aarch64 CROSS_COMPILE=<cross_compiler_directory>/bin/arm-
     linux-gnueabihf-
     ```
   - To build `gatord` for Android, enter:

     ```
     <NDK_install_directory>/ndk-build
     ```
4. If you did not build `gatord` on the target, transfer it to the target and then move it to the appropriate directory. Which directory is appropriate is dependent on the target. Root should have write permission for this directory.
5. Make `gatord` executable by entering the following command:

   ```
   chmod +x gatord
   ```

Related tasks

2.9 Running the gator daemon on your target on page 2-33.
2.11 Starting a capture session on page 2-37.

Related references

3.6 Connection Browser dialog box on page 3-53.

2.8 Building the gator module

To get the full functionality of Streamline, you must build the gator driver, `gator.ko`, and place it in the target file system.

If you do not build the `gator.ko` driver, Streamline uses user space gator, which provides a subset of the functionality that kernel space gator provides. For more information, see `README.md` located in *`DS-5_install_directory`*`/sw/streamline/gator/`.

Note

It is not possible to build `gator.ko` on a Windows host.

Procedure

1. Either download the `gator.ko` source from *https://github.com/ARM-software/gator*, or copy the source supplied in *`DS-5_install_directory`*`/sw/streamline/gator/driver/`.
2. Change to the gator `driver` directory:

   ```
   cd driver
   ```
3. Assuming that you have all of the required tools for building kernel modules, you can build `gator.ko` either by using a build command or by integrating it into the kernel build system. The build command to use depends on whether you are building on a Linux host or a target.

   Use one of the following ways to build `gator.ko`:

   - On a Linux host, use the following build command:

     ```
     make -C <kernel_build_dir> M=`pwd` ARCH=arm
     CROSS_COMPILE=<cross_compiler_directory>/bin/arm-linux-gnueabihf- modules
     ```

     Note

     You must remove the comment hashtag from the following line in the makefile of the gator module to enable kernel stack unwinding:

     ```
     # EXTRA_CFLAGS += -DGATOR_KERNEL_STACK_UNWINDING
     ```

   - On a target, use the following build command:

     ```
     make -C <kernel_build_dir> M=`pwd`
     ```
   - To integrate `gator.ko` into the kernel build system, use the following instructions:

     ```
     cd <kernel_build_dir>
     cd drivers
     mkdir gator
     cp -r <path_to_gator_driver_src>/* gator
     ```

     Add the following line to the end of `Makefile` in the kernel drivers folder:

     ```
     obj-$(CONFIG_GATOR) += gator/
     ```

     Add the following line before the last `endmenu` in `Kconfig` in the kernel drivers folder:

     ```
     source "drivers/gator/Kconfig"
     ```

     This enables you to select and rebuild gator when using `menuconfig` to configure the kernel.

   To add support for an ARM Mali™-based processor, you must specify some additional options when building the gator module.

Related tasks

2.14 Setting up Streamline to support an ARM® Mali™-based device on page 2-41.
12.10 Getting L2C-310 memory-mapped peripherals working with Streamline on page 12-175.
12.11 Profiling the Linux kernel on page 12-176.

2.5 Preparing and building your Linux kernel on page 2-28.
2.11 Starting a capture session on page 2-37.

Related references

2.3 Streamline prerequisites on page 2-23.

2.9 Running the gator daemon on your target

When all the necessary files are in place, you can start `gatord`, the gator daemon. `gatord` needs to be actively running for Streamline to initiate a capture session over Ethernet or USB.

Note

- This setup task applies to both Linux and rooted Android targets. For non-rooted Android targets, install the APK in *`DS-5_install_directory`*`/sw/streamline/android`.
- You can install and run user space gator on the target automatically by clicking the **Setup target...** button in the **Connection Browser** dialog.

To run `gatord` manually, follow these steps:

Procedure

1. Copy `gatord` and, optionally, `gator.ko` into the file system on the target. On Android, you can do this on your host using `adb push`.
2. To ensure `gatord` has execute permission, enter the following command:

   ```
   chmod +x gatord
   ```

   On Android, you can execute this command using `adb shell`.
3. Optional: If `gatord` is in a different directory to `gator.ko` on the target, you must do one of the following:
   - Insert the `gator.ko` module manually using the following command:
     ```
     insmod gator.ko
     ```

     Note

     If `gatord` does not load `gator.ko`, you must reload `gator.ko` between runs of `gatord` using the following commands:

     ```
     umount /dev/gator; rmmod gator; insmod gator.ko
     ```

   - Include the path to `gator.ko` using the `-m` option to `gatord`. For example:

     ```
     ./gatord -m /home/gator/gator.ko &
     ```

   If `gatord` does not exist in the same directory as `gator.ko` and you do not either manually insert `gator.ko` or include a path to it when running `gatord`, a user-space only version of `gatord` runs when the command is executed. This user-space version of `gatord` has most, but not all of the functionality of the standard version of `gatord` running with the `gator.ko` module.

   For more information, see `README.md` located in *`DS-5_install_directory`*`/sw/streamline/gator/`.
4. Ensure that you have root privileges, if necessary, then enter the following to execute the gator daemon:

   ```
   ./gatord &
   ```

   By default, `gatord` uses port 8080 for communication with the host, but you can adjust this by launching `gatord` with the port number as a parameter. For example:

   ```
   ./gatord -p 5050 &
   ```

   Additionally, specify the port number using the **Capture & Analysis Options** dialog box by appending a colon followed by the port number to the IP address in the address field. For example, if the address is 10.99.28.54 and the port is 5050 you enter 10.99.28.54:5050. If you do not provide a port number, the default port is used.

If you use Security-Enhanced Linux (SELinux), you might see one of the following errors when running `gatord`:

- ```
  Unable to mount the gator filesystem needed for profiling.
  ```
- ```
  Unable to load (insmod) gator.ko driver:
   >>> gator.ko must be built against the current kernel version & configuration
   >>> See dmesg for more details
  ```

If you see one of these error messages, enter the `dmesg` command for more details.

If the output from `dmesg` contains the text `SELinux: initialized (dev gatorfs, type gatorfs), not configured for labeling`, enter the following command to disable SELinux:

```
setenforce 0
```

After `gatord` has started, you can re-enable SELinux by using the following command:

```
setenforce 1
```

Related tasks

2.11 Starting a capture session on page 2-37.
2.12 Stopping the gator daemon on page 2-38.

Related references

3.6 Connection Browser dialog box on page 3-53.
2.13 gatord command-line options on page 2-39.

2.10 Connecting to an Android target

Streamline supports connecting to your Android target through USB, or using an Ethernet or WiFi connection. Rooted Android devices can use `gatord` for full data collection. Non-rooted Android devices must use the supplied Android daemon application instead.

This section contains the following subsection:

2.10.1 Installing and using the Android daemon application

Use the Android daemon application with non-rooted Android devices.

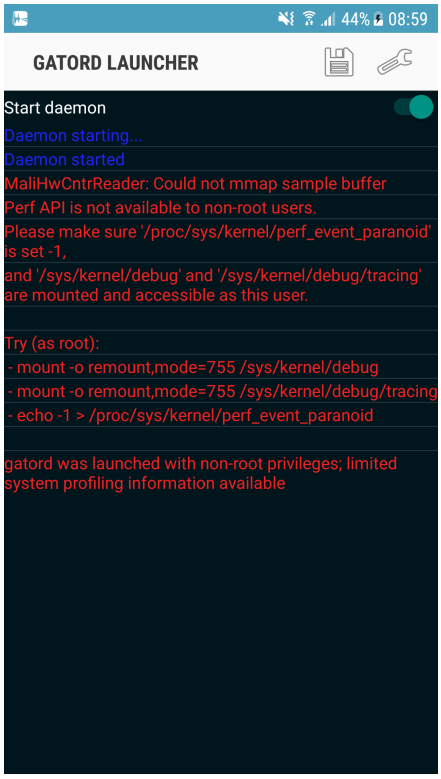
Install the Android daemon application by entering the following command:

```
adb install DS-5_install_directory/sw/streamline/android/streamline-daemon-app.apk
```

Use the application as follows:

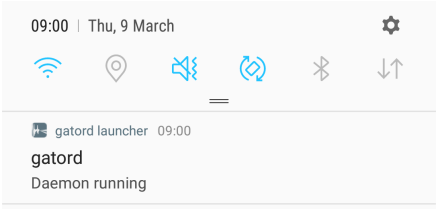
Procedure

1. Start the application.
2. Use the toggle switch to start the daemon.

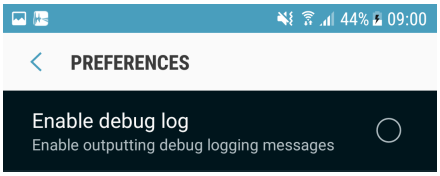


Note

The application continues to run until you press the toggle switch again. It is safe to leave the daemon application to use another process. While the daemon is running, there is an icon in the notification bar that you can use to return to the daemon.



3. Tap  to configure whether the debug log is shown.



4. Tap  to store the current log contents to, for example, an SD card.
5. Use the toggle switch to stop the daemon when no longer required.

2.11 Starting a capture session

When you have the gator daemon and optionally, the gator driver, up and running, you are ready to run a capture session using Streamline.

To initiate a capture session, follow these steps:

Procedure

1. Select a target using the **Connection Browser** dialog. You open this dialog by clicking the **Browse for a target** button. Alternatively, enter an IP address in the **Address** field at the top of the **Streamline Data** view.

   If you have an Android target and use the ADB to forward the port, enter `localhost` in the **Address** field. This field is automatically populated if you selected an Android device connected using ADB in the **Connection Browser** dialog.
2. Click the **Start capture** button.

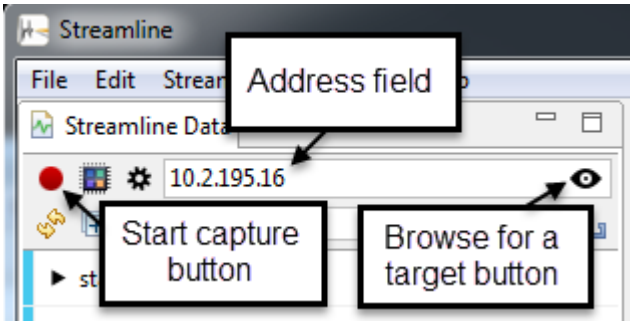


Figure 2-1 Starting a capture session

If an up-to-date version of gator is running on your target, **Live** view opens and begins plotting capture data on the bar graphs in real time.

Related references

3.6 Connection Browser dialog box on page 3-53.
15.1 Troubleshooting target connection issues on page 15-207.

2.12 Stopping the gator daemon

You might want to shut down the gator daemon, for example when updating it with a newer version.

To stop the gator daemon, follow these steps:

Procedure

1. Determine the process id of `gatord` using the following command:

   ```
   ps ax | grep gatord
   ```

2. Kill the identified process using the following command:

   ```
   kill process_id
   ```

   Replace `process_id` with the process identification number obtained in the previous step.

Note

If you stop the gator daemon on the target, you must also unload the gator module, if it is loaded. To check if the gator module is loaded, enter the following command on the target:

```
lsmod | grep gator
```

If the gator module is loaded, unload it using the following command:

```
umount /dev/gator; rmmod gator
```

You must be logged in as root to do this.

2.13 gatord command-line options

When you start `gatord` from the command line, you can pass various options to it, including options that define the location of the `gator.ko` module and the configurable xml files it uses to define the parameters of the capture session.

You can use the following options with `gatord`:

`-a`

Allows users to run a command on the target during profiling. The command is specified in the **Capture & Analysis Options** dialog.

——— Caution ———

If you use this option, an unauthenticated user will be able to run arbitrary commands on the target using Streamline.

———————————

`-c` *path*`/configuration.xml`

Sets the location of the `configuration.xml` file that defines the capture options. Include the directory location and the file name. This option is useful when the directory containing `gatord` is not writable.

`-d`

Displays `gatord` debug messages.

`-e` *path*`/events.xml`

Specifies the location of the `events.xml` file to use with `gatord`. `events.xml` defines all of the counters that Streamline collects during the capture session. Include the directory location and the file name.

`-E` *filename*

Specifies an XML file that defines one or more event counters to append to the `events.xml` file. This option allows you to add new events to gator without having to rebuild `gatord` or to entirely replace `events.xml`.

The XML file must include the XML header and elements shown in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<events>
  <category name="Filesystem">
    <event counter="filesystem_loginuid" path="/proc/self/loginuid" title="loginuid"
name="loginuid" class="absolute" description="loginuid"/>
  </category>
</events>
```

`-h`

Lists all of the available `gatord` command-line options.

`-m` *path*`/gator.ko`

If `gator.ko` is located in a different directory to `gatord`, this option defines its location.

`-p` *port_num*

Sets the port number that `gatord` uses to communicate with the host.

`-P` *filename*

Specifies an XML file that defines a new PMU to add to the list of PMUs that `gatord` has built-in support for. The list is defined in `pmus.xml`, which is located in the gator daemon source directory.

This option, used in combination with `-E` to define the events generated by the PMU, allows you to add support for a new PMU without having to rebuild `gatord`.

`-s` *path*`/session.xml`

Defines the location of the `session.xml` file. Include the directory location and the file name. This option is most useful when performing a local capture.

`-o` *apc_dir*

When performing a local capture, defines the location of the resulting APC directory.

`-v`

Displays version information for `gatord`.

Related references

4.1 Capture & Analysis Options dialog box settings on page 4-61.

2.14 Setting up Streamline to support an ARM® Mali™-based device

Streamline enables you to gather GPU-specific profiling data for a Mali-based device. This adds a significant amount of data about the graphical performance of your target to the Analysis Reports.

Mali Utgard and Mali Midgard are architectures that underlie a number of GPUs. Kernel space gator supports both Mali Utgard and Mali Midgard-based devices. In addition, user space gator version 22 and later supports Mali Utgard-based devices. Mali Utgard DDK version r6p0-00rel0 supports user space gator.

To use Streamline with a Mali-based device, you must have the following:

- A supported Mali-based device.
- A sufficiently recent version of the Mali driver. Consult your supplier to see if this version of the driver is available for your device. For more information, see the Mali Developer site, *http://www.malideveloper.com*.

Follow the normal installation and setup instructions for Streamline and `gatord`. If you are using kernel space gator, see the topic *2.8 Building the gator module* on page 2-31 for information on how to build `gator.ko`. The following instructions describe the additional configuration options required by `gator.ko` to support Mali-based devices. They assume you are building `gator.ko` out-of-tree.

Procedure

1. Streamline only supports one type of GPU (and driver version) at once, and you choose this at build time.
   - For a Mali-4xx GPU, specify the following options to your `gator.ko` make command:

     ```
     GATOR_WITH_MALI_SUPPORT=MALI_4xx  # Set by CONFIG_GATOR_MALI_4XXMP

     # gator source needs to #include "linux/mali_linux_trace.h"
     CONFIG_GATOR_MALI_PATH=".../<path_to_Mali_DDK_kernel_files>/src/devicedrv/mali"

     GATOR_MALI_INTERFACE_STYLE=<3|4>  # 3=Mali-400 DDK >= r3p0-04rel0 and < r3p2-01rel3
                                       # 4=Mali-400 DDK >= r3p2-01rel3
                                       # A default of 4 is set in
                                       # gator-driver/gator_events_mali_4xx.c
     ```

     To add the corresponding support to the Mali drivers, user space needs the following options:

     — `MALI_TIMELINE_PROFILING_ENABLED=1`
     — `MALI_FRAMEBUFFER_DUMP_ENABLED=1`
     — `MALI_SW_COUNTERS_ENABLED=1`

     Kernel space needs `USING_PROFILING=1 # Sets CONFIG_MALI400_PROFILING=y`

     These settings are the defaults in later driver versions. See the documentation supplied with the ARM Driver Development Kit (DDK) for Mali-4xx GPUs for more details.
   - For a Mali Midgard GPU, specify the following options to your `gator.ko` make command:

     ```
     GATOR_WITH_MALI_SUPPORT=MALI_MIDGARD # Set by CONFIG_GATOR_MALI_MIDGARD

     # gator source needs access to headers under .../kernel/drivers/gpu/arm/...
     # a default of . is suitable for in-tree builds
     DDK_DIR=".../<path_to_Mali_DDK_kernel_files>"
     ```

     For details of the corresponding options required by the Mali drivers, see the documentation supplied with the ARM DDK for Mali Midgard GPUs.
2. Install the gator driver as a kernel module as normal:

   `insmod gator.ko`

   Note

   If you chose to build the Mali driver out of tree, you should install it before installing the gator driver.

3. Verify that you built the module successfully:

```
ls -l /dev/gator/events/ARM_Mali*
```

This command should produce a list of counters.

If you have successfully built the gator driver with support for Mali technology, you can run a capture session on a Mali-based target.

Follow the normal instructions for setting capture options and triggering a capture session.

Related tasks

2.11 Starting a capture session on page 2-37.
2.8 Building the gator module on page 2-31.

2.15 Setting up Streamline to support ARM® Mali™-V500

Streamline supports the Mali-V500 hardware video encoder/decoder.

Assuming you have the required kernel setup for Mali-V500, the steps to capture Mali-V500 specific counters are as follows:

Procedure

1. Start `gatord`. You can use either user space or kernel space gator. It checks whether the file `/dev/mv500` exists. If it exists, this means that the kernel is configured for Mali-V500.
2. Select the Mali-V500 specific counters you are interested in using the **Counter Configuration** dialog box.
3. Start video decoding and start a capture, in either order.

Streamline displays the captured information.

2.16 Recommended compiler options

When building executables for profiling using Streamline, it is best practice to use the compiler options that are listed in this topic.

When using GCC, use the following options:

`-g`
: Turns on the debug symbols necessary for quality analysis reports.

`-fno-inline`
: Disables inlining and substantially improves the call path quality.

`-fno-omit-frame-pointer`
: Compiles your EABI images and libraries with frame pointers. This option enables Streamline to record the call stack with each sample taken.

`-marm`
: When building for ARMv7 and earlier, this option is required if GCC was compiled with the `--with-mode=thumb` option enabled. Using the `--with-mode=thumb` option without `-marm` breaks call stack unwinding in Streamline.

Note

Streamline does not support call stack unwinding for T32 (Thumb®) code. It also does not support call stack unwinding for code that is generated by ARM Compiler version 5 and earlier (`armcc`).

For Android, Streamline can profile OAT files that are generated by Android runtime (ART), down to function level.

To enable OAT files to be built with debug symbols, ensure that `dex2oat` runs with the `--no-strip-symbols` option. This includes function names, but not line numbers, in the OAT files. As a result, the Streamline report for the application shows function names and disassembly in the **Code** view, but not source code.

To do this, run the following command on the device and then re-install the APK file:

```
setprop dalvik.vm.dex2oat-flags --no-strip-symbols
```

To verify the options for `dex2oat` are set correctly, run the command:

```
getprop dalvik.vm.dex2oat-flags
```

To check whether DEX files contain `.debug_*` sections, you could use the GNU tools `readelf` command, for example:

```
readelf –S …/images/*.dex
```

Related information

readelf.

Chapter 3
Streamline Data View

Describes how to use the **Streamline Data** view to select a target, configure, start, and stop a capture session, and manage your existing captures.

It contains the following sections:

3.1 Streamline Data view

The **Streamline Data** view is your command center for much of the functionality of Streamline. Using this view, you can change capture options, edit counter configurations, manage your existing Streamline captures, and start new capture sessions.

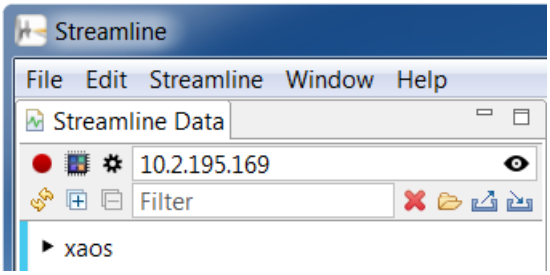


Figure 3-1 Streamline Data view

Streamline embeds the report data in the capture, instead of producing a separate analysis report. To generate new report data from an existing capture, use the **Analyze…** contextual menu. This overwrites the existing report. Use the **Duplicate** contextual menu if you want to preserve the original report.

Related tasks

3.9 Re-analyzing stored Streamline capture data on page 3-57.

3.2 Streamline Data view toolbar options

The toolbar of the **Streamline Data** view enables you to change capture options, edit counter configurations, start new capture sessions, and manage your existing Streamline captures.

The following controls are available in the toolbar of the **Streamline Data** view:


Collapse all
Collapses all captures in the **Streamline Data** view, hiding the detailed information for each one. This option is disabled if all captures are already collapsed.


Expand all
Expands all captures in the **Streamline Data** view, exposing detailed information for each one. This option is disabled if all captures are already expanded.


Delete
Deletes the selected captures from the file system.


Refresh
Refreshes the contents of the **Streamline Data** view. If you have added Streamline capture files to any of the analysis data locations and want to see them in the **Streamline Data** view immediately, use **Refresh** to sync the view. Otherwise, the view is refreshed automatically in the background.


Edit Locations...
Opens the **Data Locations** preferences dialog, which enables you to define the folders on your file system that contain Streamline data.


Export Capture File(s)...
Opens a dialog box that enables you to save the currently selected `.apc` directories as zip archives that you can later import into Streamline using the **Import Capture File(s)...** option. ARM recommends using this option rather than copying `.apc` directories, because it significantly reduces the file sizes.


Import Capture File(s)...
Opens a **File System** dialog box. To import a capture, navigate to a `.zip` archive that contains a valid `.apc` directory, or to a perf recording. Perf files are expected to match the naming pattern `*perf.data*`, `*.perf`, or `*.data`. If you have used a different name, select the `*.*` option. Select the file, then click **Open**. The newly imported capture then appears in the list of captures in the **Streamline Data** view.


Start capture
Starts a capture session. Streamline uses your settings from the **Capture & Analysis Options** dialog box. If you have not defined settings using the **Capture & Analysis Options** dialog box, Streamline connects to the target at the address that you entered in the **Address field** using default values for each of the capture options. Before the capture starts, you are prompted to enter a name for the capture.

Streamline adds the parent directory of the capture to the **Analysis Data Locations** list if it is not already there. When the capture session stops, Streamline generates report data based on the settings in the **Capture & Analysis Options** dialog box. You can use this dialog box to set a capture session length. If you want to control the length of the session yourself, you can use the **Streamline Data** view to terminate it manually. To do so, click the either the **Stop** button that appears on the right in the new capture entry or the **Stop capture and analyze** button in the **Live** view.


Counter Configuration
Opens the **Counter Configuration** dialog box. Use it to modify the performance counters tracked during your capture session.

Capture & Analysis Options

Opens the **Capture & Analysis Options** dialog box. Use it to set the capture session parameters.

Browse for a target

Part of the **Connection** field. Enter the network name or IP address of your target or click this button to open the **Connection Browser** dialog box, in which you can select from a list of available targets on your network. Select one, then click the **Select** button. The **Connection** field updates to show the newly selected target.

Filter field

Enter a string in this field to display only the captures whose name contains this string.

Change analysis options and regenerate the report

This button appears if you use the disclosure control to show more information about a capture. Use this button, or the **Streamline Data** view contextual menu to launch the **Analyze** dialog box to re-analyze the capture session data.

3.3 Importing perf data

Streamline can import event data that is created using the Linux perf command-line tools. This event data is then converted to APC format, which can be visualized in the **Timeline** view.

The `perf record` command, one of the Linux perf command-line tools, captures an event trace.

Import the event trace into Streamline by clicking **Import Capture File(s)…** and selecting the file containing the event trace. The imported event data is converted to APC format, which can be displayed in the different views.

Counters that are in the default `events.xml` file, which gator recognizes, are configured as though gator captured them. Counters that gator does not recognize are displayed in the default style. If this style is not appropriate for a particular counter, use the chart configuration panel to change the default settings.

Tracepoint events that are not explicitly recognized in events are treated as event counters. The value of the counter is the number of times the tracepoint event occurred. **Heat Map** mode and **Core Map** mode require `sched:sched_switch` tracepoint to be captured.

To emulate gator, do a global capture of all cores with hardware events configured to be sampled into a group using:

```
perf record -a -c 1 -e "{sched:sched_switch, cpu-clock/period=<SAMPLE_PERIOD>/, <EVENT_1>,
<EVENT_2>...<EVENT_N>}:S"
```

The parameters for this command are as follows:

`sched:sched_switch`
: causes the scheduler information to be captured as required for **Heat Map** and **Core Map** modes, and triggers a sample of the counters on each context switch.

`cpu-clock/period=<SAMPLE_PERIOD>/`
: configures a periodic sampling event, causing the counters in the group to be sampled periodically. Set this parameter according to the **Sample Rate** as follows:

  Sample Rate: Normal
  : `cpu_clock/period=1000000/`

  Sample Rate: Low
  : `cpu_clock/period=10000000/`

  Sample Rate: None
  : Do not include this parameter.

`<EVENT_1> ... <EVENT_N>`
: are the counters to be sampled.

An example of this command in use could be:

```
perf record -a -c 1 -e "{sched:sched_switch, cpu-clock/period=10000000/, branch-
instructions, branch-misses}:S"
```

Extra counters, particularly tracepoints, do not need to be part of the sample group. Sometimes they cannot be part of the same group as `perf` does not mix counters from different PMUs.

An example for multiple different groups:

```
perf record -a -c 1 -e "{sched:sched_switch, cpu-clock/period=10000000/, branch-
instructions, branch-misses}:S, {cpu-clock/period=10000000/, alignment-faults, page-
faults}:S"
```

An example with more, generally low frequency, events:

```
perf record -a -c 1 -e "{sched:sched_switch, cpu-clock/period=10000000/, branch-
instructions, branch-misses}:S, power:cpu_frequency:S"
```

Other supported combinations of events and flags are as follows:

Frequency-based sampling

```
perf record -a -F 1000 -e branch-misses
```

Single process sampling

```
perf record -e instructions -- dmesg
```

The content of the data that is captured using these commands is not as complete as the data captured in the earlier examples. Some of the user interface features are missing as a result. For example, **Heat Map** mode is incomplete.

Related references

15.2 Troubleshooting perf import issues on page 15-209.

3.4 Streamline preferences

Use the **Edit** menu in the **Streamline Data** view to configure the following Streamline preferences:

Memory Monitor
: Select this option to display the heap status at the bottom of the Streamline window.

Reset All Deletion Prompts
: Select this option to receive a prompt before deleting captures, charts, or series.

Customer Experience Improvement Program
: Select this option to allow ARM to collect anonymous information about how you are using Streamline. ARM uses this information to improve the Streamline features that you use most often.

3.5 Capture color-coding

Each of the captures in the Streamline Data view has a color-coded bar to its left. The color of the bar tells you the status of the capture and dictates what happens when you double-click on it.

The following colors can appear next to a capture:

Blue

Blue indicates a working capture that contains a report compatible with the current version of Streamline. Double-click on a blue capture to open its report.

Amber

Amber indicates a capture that needs to be re-analyzed, as its report is missing or incompatible with the current version of Streamline. Generate a new capture by either clicking the **Analyze** button or by double-clicking on the capture.

Red

Red indicates the capture does not contain valid data and cannot be re-analyzed. You must run another capture session on your target. This often occurs when the capture was created using a much earlier version of Streamline. It might indicate a capture error, in which case the error message is displayed in a tooltip.

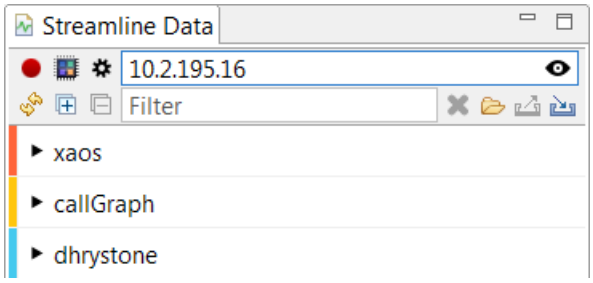


Figure 3-2 Color-coded captures

3.6 Connection Browser dialog box

The **Connection Browser** dialog box, opened from the **Streamline Data** view, enables you to select a target without having to look up its name or IP address.

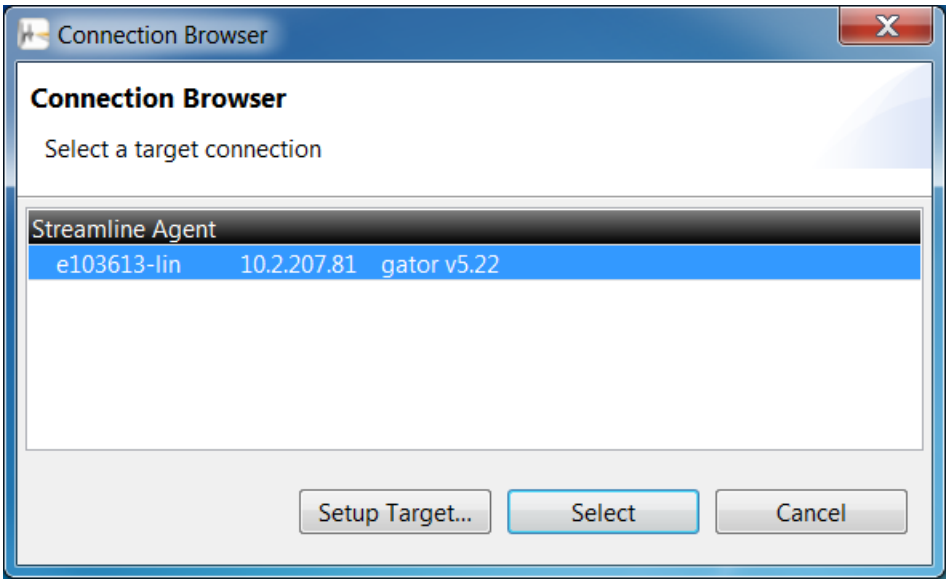


Figure 3-3 Connection Browser in Streamline

When you click the **Browse for a target** button in the **Connection** field of the **Streamline Data** view, Streamline searches your network and provides a selectable list of possible targets. The **Connection Browser** sorts the targets by type:

Streamline Agent
: Lists all targets that are in the same subnet as the host or are connected to it by USB and have gator installed and running.

Streamline Agent via ADB
: Lists all Android targets that are connected to the host using ADB or Tizen targets that are connected using SDB, and have gator installed and running. You must have installed ADB or SDB and set the path to it in the **Capture & Analysis Options** dialog in order for this list to be populated.

Click **Setup Target…** to open the **Setup Target** dialog box.

Related references

4.1 Capture & Analysis Options dialog box settings on page 4-61.
3.2 Streamline Data view toolbar options on page 3-47.
3.7 Setup Target dialog box on page 3-54.

3.7 Setup Target dialog box

This dialog allows you to install gator automatically onto a rooted Android or Linux target, without having to carry out the manual steps for configuring and running gator.

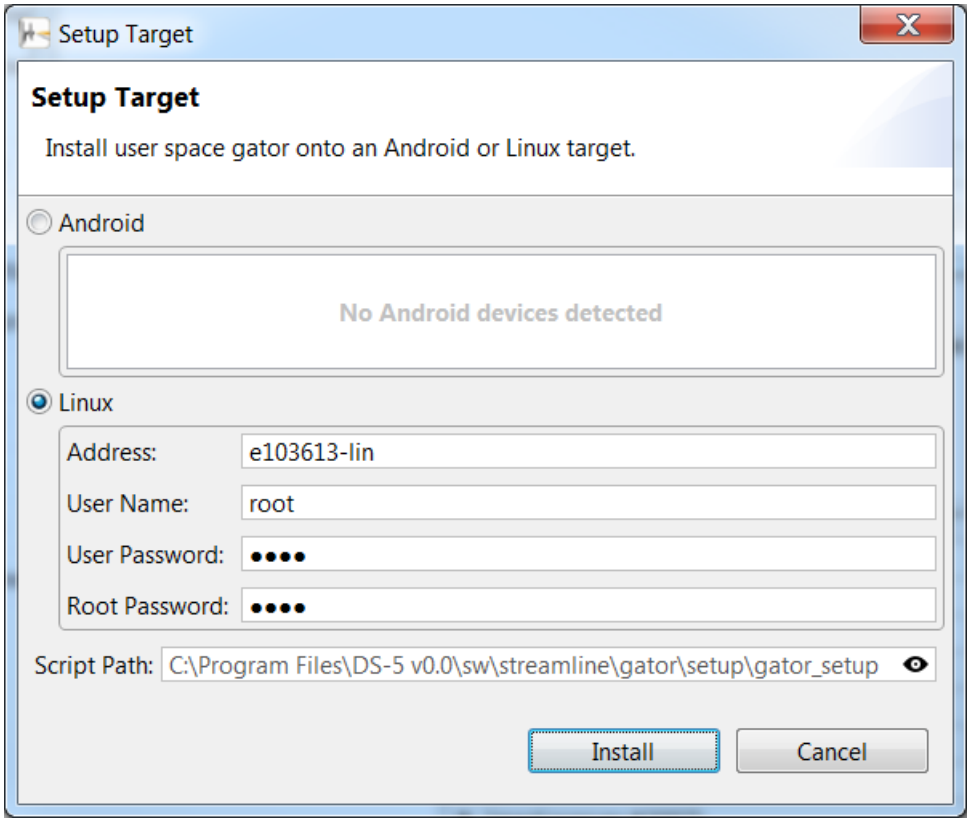


Figure 3-4 Automatically setting up Streamline on the target

For an Android target, select the device. For a Linux target, specify the IP address, user name, and passwords.

Click **Install** to install and run a pre-built user space `gatord` binary which runs on Android or Linux and on ARMv7 or ARMv8 targets. It runs using the `-a` flag, so that you can enter a command in the **Capture & Analysis Options** dialog box without having to restart it.

Clicking **Install** also installs a file called `notify.dex` into the same directory as `gatord` on the target. `gatord` uses `notify.dex` on Android targets to notify running applications when atrace annotation tags are enabled. This feature is supported on Linux kernel versions 3.10 and later.

A shell script controls the target setup process. When you click **Install**, Streamline copies the script to the target and runs it. If you leave the **Script Path** field empty, Streamline uses the default script, *`DS-5_install_directory`*`/sw/streamline/gator/setup/gator_setup`. If this script fails to set up gator correctly on your target, you can modify it or create a new script and enter its location in the **Script Path** field.

Note

- `gatord` must run as root. If you are using ADB, `adbd` also must run as root.
- **Setup Target** does not set up `gatord` to start automatically on boot, so you must carry out target setup every time the device is rebooted.

Related concepts

2.4.1 Comparison of user space gator and kernel space gator on page 2-24.
12.9 Enabling atrace and ttrace annotations on page 12-174.

Related references

2.13 gatord command-line options on page 2-39.

3.8 Analysis Data Locations dialog box options

Use the **Analysis Data Locations** dialog box to define locations on your file system that contain Streamline analysis data.

The following buttons are included in the **Analysis Data Locations** dialog box:

Add

Opens another dialog box that enables you to search your file system to add a new folder to the list.

———— **Note** ————

There is an example Streamline capture in the `xaos` directory. `xaos` is one of the DS-5 Linux examples in *`DS-5_install_directory`*`/examples/Linux_examples.zip`. After unzipping the file, use the **Add** button and choose the `xaos` project directory to display the example capture in the Streamline **Data** view.

————————————

Remove

Deletes a folder from the list.

OK

Closes the dialog box. All captures in the newly defined folders appear in the Streamline **Data** view.

Cancel

Discards any current changes to the list of locations and exits the dialog box.

Reset

Resets the list to the default value. The default folder depends on your OS:

- On Windows, it is `C:\Users\<`*`user_name`*`>\Documents\Streamline`.
- On Linux and Macintosh, it is `~/Documents/Streamline`.

3.9 Re-analyzing stored Streamline capture data

Re-analyzing an existing capture creates new report data, replacing the report data that already exists in the capture.

There are a number of reasons why you might want to do this, for example:

- To change your analysis options.
- Your sources might not have been available when the data was captured.
- A new version of Streamline might be incompatible with the existing report data.

To re-analyze a Streamline capture:

Procedure

1. In the **Streamline Data** view, use the disclosure control to open up the details of a capture, then click the **Analyze** button in the lower right of the capture.
2. In the resulting dialog box, make the required changes to the settings.

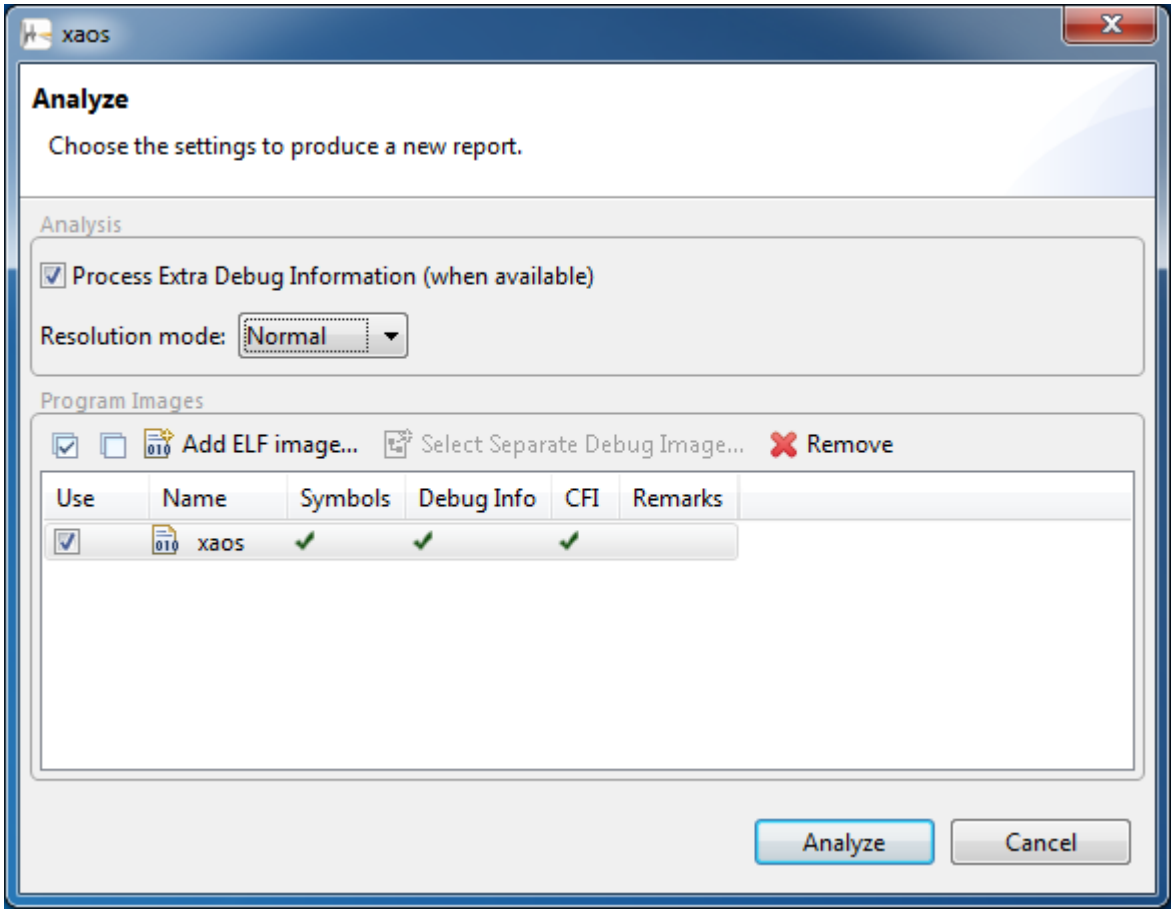


Figure 3-5 Analyze dialog box

Note

The options here are a subset of the options available in the **Capture & Analysis Options** dialog box and they work the same way. In the **Analysis** section, use the checkbox to toggle the **Process Debug Information** on and off. Use the **Resolution mode** drop-down menu to select **Normal**, **High**, or **Ultra-High** resolution for the **Timeline** view. Use the **Program Images** section to add or remove any number of images, executables, and APK archives.

3. Click **Analyze**.

Related tasks

12.1 Capturing data on your target without an Ethernet connection on page 12-164.

Related references

3.2 Streamline Data view toolbar options on page 3-47.

4.1 Capture & Analysis Options dialog box settings on page 4-61.

3.10 Duplicating a capture

You can duplicate an existing capture in the **Streamline Data** view using the contextual menu. For instance, you might want to re-analyze a capture with new options while preserving the original capture.

To duplicate a capture:

Procedure

1. Right-click on the capture.
2. Choose **Duplicate** from the resulting contextual menu.
3. In the **New Name** dialog box, give the **Duplicate Streamline Document** a new name.

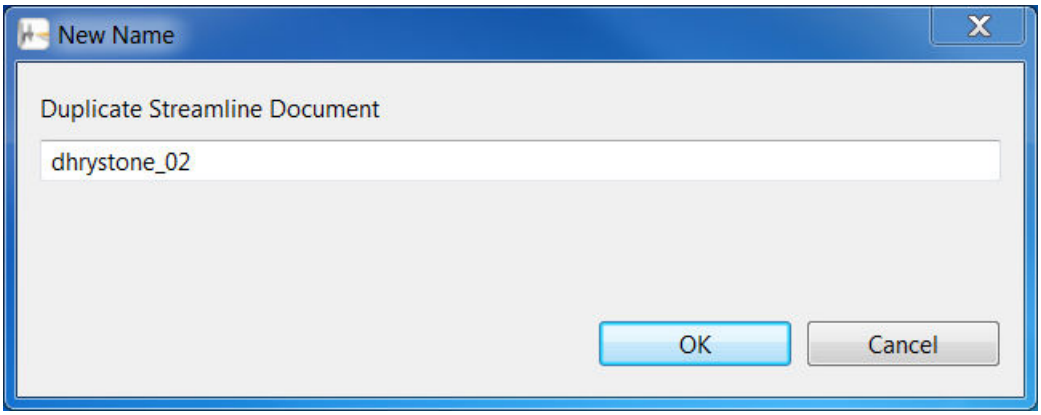


Figure 3-6 Duplicating a capture

Chapter 4
Capture and Analysis Options

Describes how to use the **Capture & Analysis Options** dialog box to change capture session settings, such as duration, sample rate, and buffer size.

It contains the following section:

4.1 Capture & Analysis Options dialog box settings

The **Capture & Analysis Options** dialog box enables you to change the capture session settings, including the IP address of the target, duration, sample rate, and buffer size.

To open it, click the **Capture & Analysis Options** button (⚙) in the **Streamline Data** view.

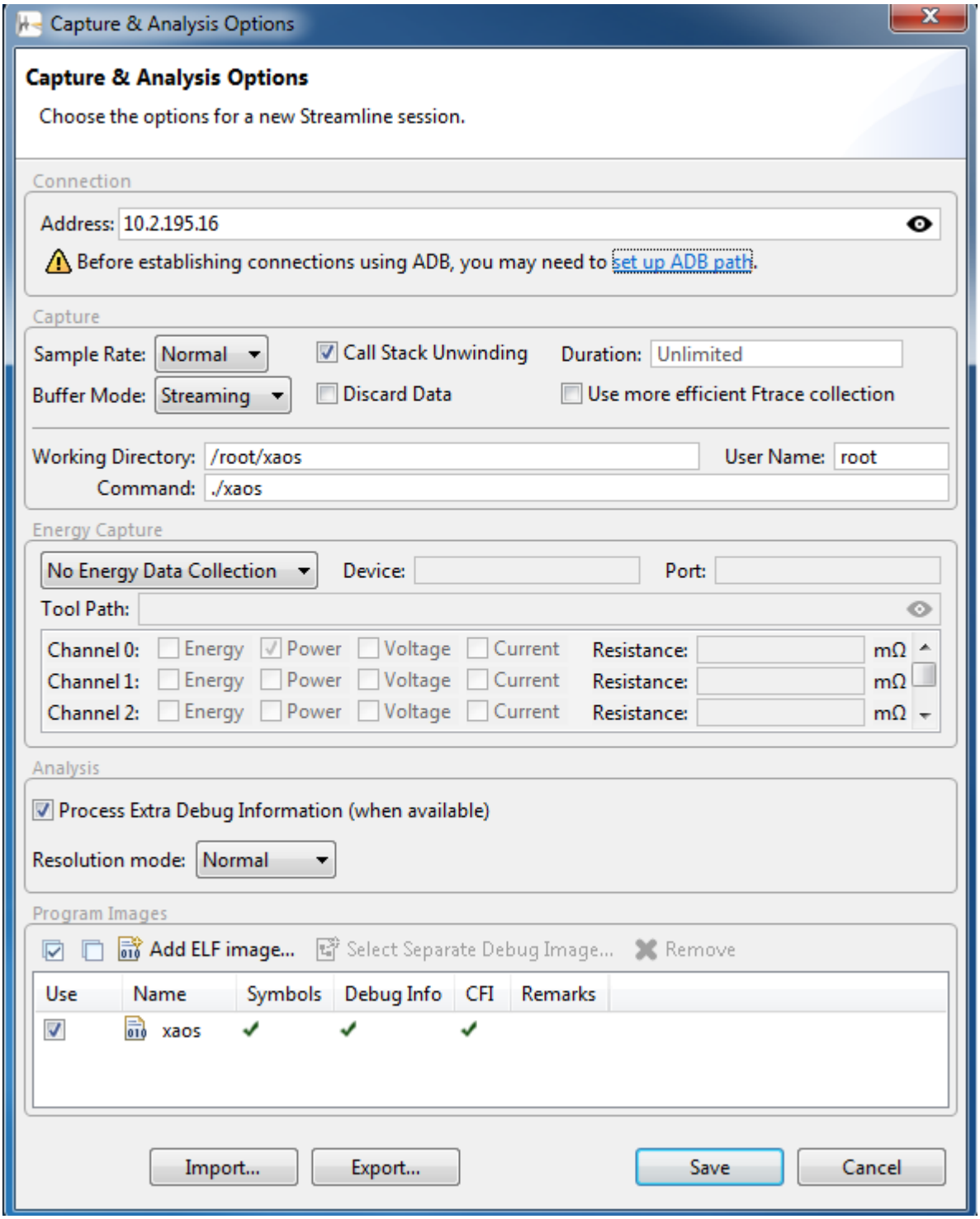


Figure 4-1 Capture & Analysis Options dialog box

Connection

The Connection section contains the following settings:

Address

The IP address of the target. You can alternatively enter the network name of your target. The value that is given in this field overwrites the value in the **Address** field of the **Streamline Data** view, if one has been given. The reverse is also true. If you enter a new address in the **Address** field of the **Streamline Data** view, it replaces the value that was entered here.

Note

- By default, Streamline uses port 8080 to connect to a target. To use a different port, specify one here by entering a colon and a port number after the IP address. For example, enter *`Your_IP_address`*`:1010` to use port 1010 to connect to the target.
- If you use the port forwarding of Android ADB with USB, enter `localhost` in the **Address** field.

Browse for a target button ()

The **Browse for a target** button, on the right side of the **Address** field, opens the **Connection Browser**. Streamline searches your network and produces a list of possible targets. Selecting one populates the **Address** field.

Capture

The **Capture** section contains the following options:

Sample Rate

The target generates periodic measurement interrupts according to the following settings: **Normal**=1kHz, **Low**=100Hz, and **None**. The **Normal** setting works well in most cases. **Low** is recommended if you have a slow target, or if the target is heavily loaded, because it means less intrusion by Streamline. The Low setting requires a longer capture to collect representative data. Set Sample Rate to **None** to ensure that Streamline has the lowest level of intrusion on your code, but this also means that resulting reports show only zeroes in any report columns that rely on sampling. Enabling event-based sampling for a counter overrides this timer-based sampling.

Buffer Mode

The default setting is **Streaming**, which enables unbounded streaming of target data directly to your host using a 1MB buffer. You can also use one of the following store-and-forward buffers:

Mode	Buffer size
Large	16MB
Medium	4MB
Small	1MB

If you select one of these sizes, the capture ends when the buffer is full, which prevents the intrusion that is caused by streaming data from the target to the host.

Note

You must set the Buffer Mode to Streaming to enable **Live** view. If you select one of the buffer sizes, **Live** view does not display real-time data during the capture session.

Call Stack Unwinding
Select this checkbox to ensure that Streamline records call stacks. This option greatly improves the visibility of the behavior of the target, but increases the amount of raw data Streamline sends from the target to the host. Ensure that you compile your EABI images and libraries with frame pointers enabled using the `-fno-omit-frame-pointer` compiler option. If GCC is compiled with the `--with-mode=thumb` option, you must also use the `-marm` option.

——— **Note** ———

- Streamline supports call stack unwinding for ARM binaries created using GCC or ARM Compiler 6 (`armclang`), provided you compile them with frame pointers enabled. Streamline does not support call stack unwinding for code that is generated by ARM Compiler 5 and earlier (`armcc`).
- User space gator does not support call stack unwinding.

———————————

Discard Data
If this option is enabled, Streamline discards all data when you terminate the capture. Use this option if you only want to see the data that streams during a capture session and do not want to generate a capture.

If you select this option, the **Stop capture and analyze** button in the **Live** view is disabled.

Duration
The length of the capture session, in minutes and seconds. For example, enter 1:05 for 1 minute and 5 seconds. If you do not provide a value here, the capture session continues until you stop it manually, or the buffer is full.

Use more efficient Ftrace collection
When this option is selected, Streamline uses a more efficient way of collecting ftrace, atrace, and ttrace counters. Charts that rely on these counters are empty in **Live** view, although the data is present when you view the report.

To use the following Command settings, you must use the `-a` option when starting `gatord`:

Working Directory
The absolute path of the directory on the target in which to run the command that is specified in the **Command** field.

User Name
The user account to run the command as.

Command
A command to run on the target. The command is run a few seconds after the capture begins. The capture is not automatically terminated when the command finishes.

Energy Capture

The following settings define your energy capture device:

Energy Capture drop-down menu
This menu has three options. Select **No Energy Data Collection** to turn energy capture off. When you select this option, all other energy capture options are disabled. Select **ARM Energy Probe** or **NI DAQ** to match your energy capture hardware.

Device
Use this field to give Streamline the name of your target energy capture device. Streamline attempts to auto-detect your device if this field is left blank.

When using ARM Energy Probe on Linux, enter `/dev/ttyACM0` in the **Device** field, if required.

When using NI DAQ, the device name depends on the drivers that are installed on the host. For example, when using NI DAQmx Base drivers, the device name is usually `Dev1`. You can determine the device name by using the National Instruments `List Devices` utility.

Port
The port Streamline uses to communicate with your chosen energy capture device. The default port is 8081.

Tool Path
Use this field to define the path to the `caiman` executable, which is required to use either the ARM Energy Probe or a NI DAQ device to gather power output statistics. The button to the right of this field enables you to search your file system.

In addition to the settings that define your capture device, the Energy Probe section has configuration options that apply to each channel:

Energy
When enabled, Streamline collects energy data for this channel in joules.

Power
When enabled, Streamline collects power data for this channel in watts.

Voltage
When enabled, Streamline collects voltage data for this channel in volts.

Current
When enabled, Streamline collects current data for this channel in amps.

Resistance
Use this field to define the value, in milliohms, of the shunt resistor that connects to each of the available channels. The default setting is 20 milliohms.

Analysis

The Analysis section contains the following controls:

Process Extra Debug Information (when available) checkbox
If you enable this option, Streamline processes DWARF debug information and line numbers. This option provides a higher level of detail in your captures, but results in higher memory usage. It does not affect the data that is collected during the capture session. It only affects the report data that is automatically generated after the termination of the capture session. This option can be changed when you re-analyze the stored capture.

Note

- To enable this feature, you must have built the image using the `-g` compiler option.
- If you disable this feature, the source section of the **Code** view does not display the source code or source code statistics. The disassembly is still available with this option disabled, but the source section shows only a `No source available` message.

Resolution mode drop-down menu
This menu has three options. Select **Normal** for the standard resolution in **Timeline** view. The highest resolution in this mode is milliseconds. Select **High** to instruct Streamline to process more data, enabling you to zoom in to microsecond bin sizes. Select **Ultra-High** to add one microsecond resolution to the data analysis. When **Ultra High** is selected, a warning is displayed explaining that the analysis time is increased, and that the captures are likely to require more disk space.

These options do not affect the data that is collected during the capture session. They only affect the report data that is automatically generated after the termination of the capture session. The selected option can be changed when you re-analyze the stored capture data.

MGD Mode

The MGD Mode section contains the following settings:

MGD Installation Directory
The location of the Mali Graphics Debugger (MGD) application within the DS-5 installation folder. Specify to allow MGD to be launched from **Live** view when the application is not currently running.

Continue live capture after MGD is activated

Select to instruct Streamline to continue displaying the live capture of data after MGD launches and begins tracing. Otherwise the **Live** view in Streamline is stopped.

Program Images

Use this area to explore your file system and define the images and libraries that you want to profile.

The images that you define here do not affect the data that is collected during the capture session. They only affect the report data that is automatically generated after the termination of the capture session. These images can be changed when you re-analyze the stored capture.

Note

- When compiling images on your host, ensure that you use the `-g` compiler option to enable debug symbols.
- Disabling inlining with the `-fno-inline` compiler option substantially improves the call path quality.
- As an alternative to manually selecting images and libraries on the host in the **Program Images** section, Streamline supports automatic image transfer. To use this feature, specify a regex in the **Live** view. Streamline automatically transfers images whose name matches the regex, and optionally any libraries that are used by them, from the target to the host.

  If you manually select an image in the **Program Images** section and also specify a regex that matches an image with the same name, the image that is specified manually is used.

The following buttons are included in the **Program Images** section:

Add ELF Image...

 Opens a file system dialog box that you can use to choose images to add. Select the image, executable, or Android Package File (APK) and click **Open** to add the file to the list.

Use all images for analysis / Use none of the images for analysis

Toggles the checkboxes that control symbol loading for the ELF images, executables, or APKs listed. Use the checkboxes instead of removing entries from the **Program Images** list to toggle entries on and off over multiple runs.

Remove

 Removes the selected entries.

Import

Use the resulting file system dialog box to find an existing `session.xml` file and import its settings to the **Capture & Analysis Options** dialog box.

Export

Saves the current settings as a `session.xml` file.

Save

Saves the settings and exits.

Cancel

Discards all changes and exits.

Related concepts

3.1 Streamline Data view on page 3-46.

6.1 Live view overview on page 6-78.

*6.7 Mali Graphics Debugger (MGD) Mode in **Live** view* on page 6-106.

12.9 Enabling atrace and ttrace annotations on page 12-174.

Related tasks

2.11 Starting a capture session on page 2-37.

3.9 Re-analyzing stored Streamline capture data on page 3-57.

11.8 Setting up National Instrument Multifunction Data Acquisition devices (NI DAQ) to capture energy data on page 11-161.

Related references

2.13 gatord command-line options on page 2-39.
3.6 Connection Browser dialog box on page 3-53.
15.1 Troubleshooting target connection issues on page 15-207.
15.5 Troubleshooting report issues on page 15-214.

Chapter 5
Counter Configuration

Describes how to use the **Counter Configuration** dialog box to select the events that Streamline collects.

It contains the following sections:

5.1 Opening the Counter Configuration dialog box

ARM Streamline uses a default best-fit set of hardware performance counters to aid in the analysis of your applications, but you can modify them using the **Counter Configuration** dialog box, accessed through the **Streamline Data** view.

To open the **Counter Configuration** dialog box:

Procedure

1. Choose a target using the **Connection Browser** button or enter an IP address for a valid target in the target field of the **Streamline Data** view.
2. Click the **Counter Configuration** button.

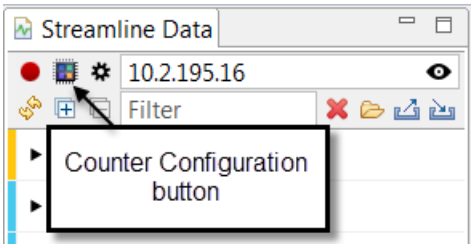


Figure 5-1 Streamline Data view - Counter Configuration

To open the **Counter Configuration** dialog box, you must be able to connect to a target on which the gator daemon is running, so that Streamline can determine which counters are available for your hardware. Clicking **Counter Configuration** without properly specifying a target produces an error message.

Related references

5.5 Counter Configuration dialog box settings on page 5-73.

5.2 Counter Configuration dialog box structure

The **Counter Configuration** dialog box contains two main areas, **Available Events** and **Events to Collect**. Select counters in the **Available Events** list to populate the **Events to Collect** list.

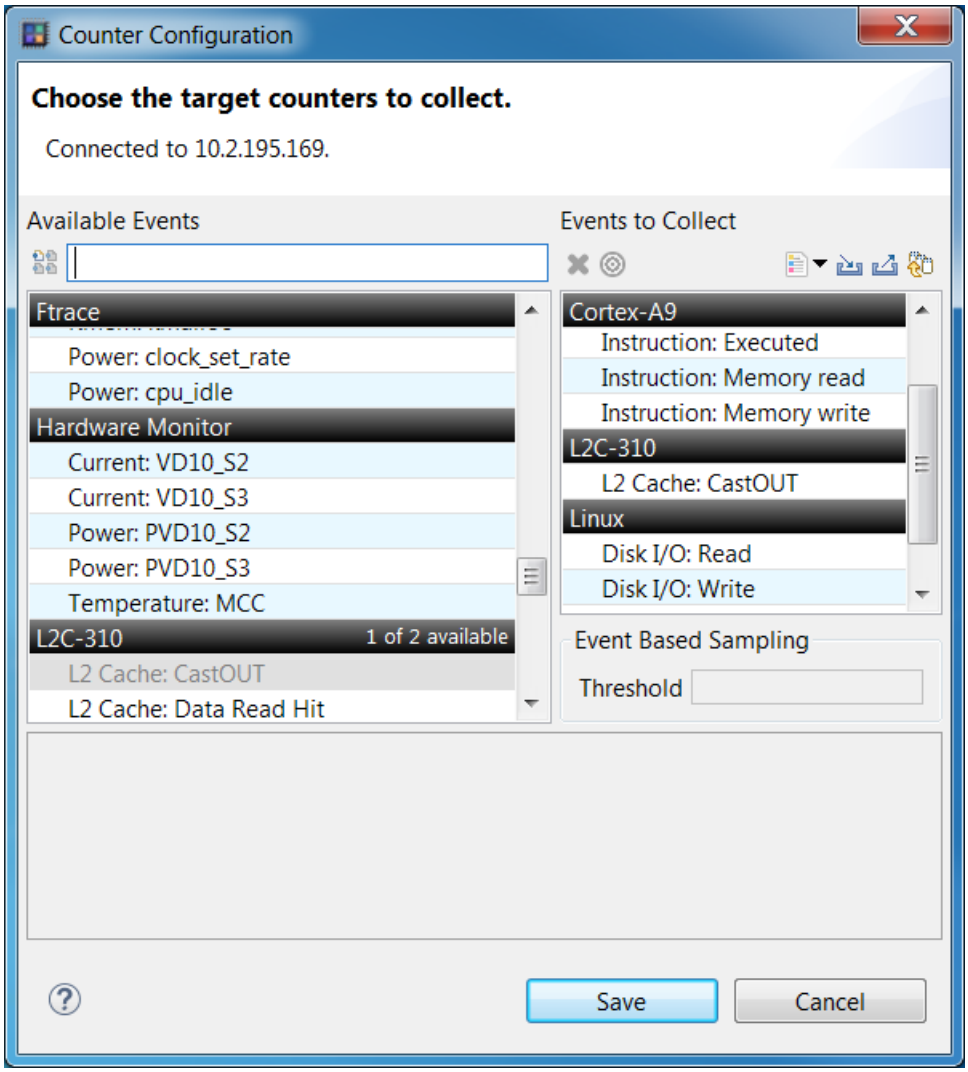


Figure 5-2 Counter Configuration dialog box

The dialog box contains the following areas:

Available Events

The **Available Events** list contains categorized events offered for each core on your target, as well as a list of other hardware and OS-specific events. The events contained in the processor lists are based on the PMU counters of the core, so can vary depending on the type of processor, as can the number of events that you can add. Events that are already in the **Events to Collect** list appear in gray.

——— **Note** ———

Streamline supports hwmon counters, for example temperature and energy consumption, if they are available on your target hardware.

———————————

The maximum number of available events and the amount remaining are shown in the upper right hand corner of the header for each category section in the **Available Events** list. When you have reached the maximum, all entries in the category list are grayed out and you cannot add any more events to the **Events to Collect** list.

——— **Note** ———

If problems occurred during gator setup, for example if the target is running a version of gator that does not support some counters, a warnings tag is displayed beside the filter field.

———————————

Events to Collect

This list of categories and events is used by the **Timeline** view for its graphs. Each event listed here is available for display in a chart in the **Timeline** view.

Event Based Sampling

This field is active only if an event in the **Events to Collect** list has event-based sampling turned on. In this case, the **Threshold** value indicates the number of times the event must be triggered in order for Streamline to sample it.

5.3 Adding new events to the Events to Collect list

Streamline uses the list of counters in the **Events to Collect** list to determine what data to collect during a capture session.

If an event you want does not appear in the list, you can add it using the following procedure:

Procedure

1. Open the **Counter Configuration** dialog box using the button in the **Streamline Data** view.
2. Double-click on an event, or select and drag events from the **Available Events** list and drop them in the **Events to Collect** area.
   The added events appear in the **Events to Collect** list under their category name.

   Note

   - The more counters that you select, the greater the probe effect is.
   - If you want Streamline to automatically collect the set of counters required to generate the charts defined in a chart configuration template, click the **Add counters from a template** button. Chart configuration templates are defined using the **Switch and manage templates** button in the **Live** and **Timeline** views.

3. In cases where a counter can be collected on one of a number of different cores or interfaces, a drop-down menu appears next to the counter in the **Events to Collect** list. Use this menu to select a specific core or interface.

   For example, the ARM Versatile™ Express TC2 hardware, featuring CCI-400 coherent interconnect, enables you to collect counters over a number of different interfaces. The interface drop-down menu in the **Events to Collect** list is specific to this hardware.

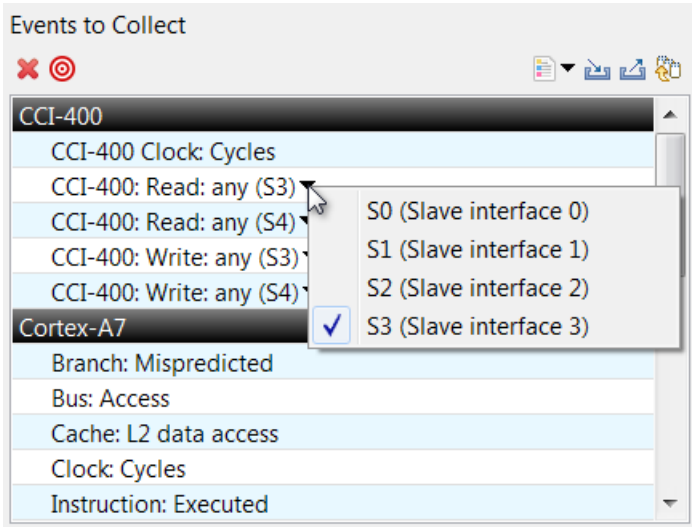


Figure 5-3 Interface drop-down menu

Related concepts

6.4.4 Chart configuration templates on page 6-92.

Related references

5.5 Counter Configuration dialog box settings on page 5-73.

5.4 Removing events from the Events to Collect list

You are limited in the number of hardware-specific events that you can collect during a capture session. Removing unwanted counters from the **Events to Collect** list frees up room to add the counters that you do want.

To remove events from the **Events to Collect** list:

Procedure

1. Open the **Counter Configuration** dialog box by clicking the button in the **Streamline Data** view.
2. Select one or more events in the **Events to Collect** list.
3. To remove the selected events from the list, press **Delete**.

Note

To replace the **Events to Collect** list with the set of counters that are required by a chart configuration template, click the **Add counters from a template** button.

Related references

5.5 Counter Configuration dialog box settings on page 5-73.

5.5 Counter Configuration dialog box settings

The **Counter Configuration** dialog box provides help resources and options that enable you to manage your configuration settings.

It contains the following options:


Warnings tag
This tag is displayed in the toolbar if one or more problems occurred during gator setup. To see the warning messages, click the tag.


Toggle advanced view
Turns the advanced view on or off. When the advanced view is on, the **Available Events** list includes any advanced events available for your target. Turn the advanced view off to display the default list of events.


Delete
Removes the selected counter from the **Events to Collect** list.


Toggle event-based sampling
Turns event-based sampling on or off for the selected counter. To define how many hits are needed for the counter before a sample is recorded, use the **Threshold** field. If this button is grayed out, the selected counter does not support event-based sampling.


Add counters from a template
Replaces the **Events to Collect** list with the counters that the selected chart configuration template requires. If you hold down the **Shift** key while clicking this button, the counters are appended to the list instead. An error message is displayed for any counters that the template requires but the target does not support.


Import...
Enables you to search for and load a counter configuration XML file that you previously generated.


Export...
Exports the current counter configuration to an XML file. If you choose to capture data locally, you can first create and export the counter configuration file and manually add it as an option when running `gatord` on the target.


Load Defaults
Resets the **Events to Collect** list to the Streamline defaults.

Save
Saves your current counter configuration and exits the dialog box. The counter configuration file, called `configuration.xml` by default, is saved to the same directory as `gatord` on your target. This directory must therefore be writeable.

Cancel
Exits the **Counter Configuration** dialog box without saving the defined settings.


Help
Opens **Counter Configuration** help.

Related concepts

5.7 Event-based sampling on page 5-75.
6.4.4 Chart configuration templates on page 6-92.

5.6 Events specific to ARM® Mali™ technology

If you connect to a Mali-based target that is configured to support Mali GPU-specific profiling, and is running a version of gator that supports the target GPU, the **Available Events** list contains events that are specific to Mali-based targets.

For information about the meaning of the Mali-specific events and how to interpret the profiling data, see the Mali GPU Application Optimization Guide. The guide is available on the Mali Developer Center, *http://malideveloper.arm.com/documentation/developer-guides/mali-gpu-application-optimization-guide*.

When choosing which Mali-specific events to add to the **Events to Collect** list, consider the following:

- The Mali-4xx GPUs contain two counters per block, each of which can count one of many events. You can add either or both to the **Events to Collect** list. You can add any number of the many available Mali Midgard counters because the Midgard hardware reports its hardware events through a block of shared memory rather than through dedicated hardware registers.
- For Mali-4xx GPUs, vertex and fragment processor counters are delivered as a single total at the end of each phase of activity.
- L2 counters report continuously because the cache is shared by the vertex and fragment processors and cannot easily be attributed to a single operation.
- Some Mali counters support multiple interfaces. Choose an interface for each counter using the drop down menu next to the counter in the **Events to Collect** list.
- Vertex, fragment, and compute activity counters are not available when using `gatord` only, and not `gator.ko`.

5.7 Event-based sampling

By default, Streamline records samples at an interval determined by the sample rate. You can override this behavior by selecting *event-based sampling* (EBS) instead.

With EBS, Streamline records samples only on context switches and when the selected event has been triggered a number of times equal to the **Threshold** value in the **Counter Configuration** dialog box. It does so for each core on your target. For standard, non-EBS captures, Streamline samples counters on every context switch and at the frequency specified in the **Sample Rate** drop-down menu in the **Capture & Analysis Options** dialog box.

For example, to trigger a sample every time a core causes 500 L2 cache misses, select L2 miss from the **Events to Collect** list and enter 500 in the **Threshold** field. Given an adequate capture session, the Samples statistic contained in many of the Streamline reports indicates which processes and functions are the potential cause of inefficient caching.

Note

- EBS is only possible when the PMU on the target hardware can generate interrupts.
- Not all counters support EBS.

Related tasks

5.8 Setting up event-based sampling on page 5-76.

Related references

5.5 Counter Configuration dialog box settings on page 5-73.

5.8 Setting up event-based sampling

In Streamline, you can override the default interval sampling with event-based sampling using the **Counter Configuration** dialog box.

To enable event-based sampling, follow these steps:

Procedure

1. Open the **Counter Configuration** dialog box using the button in the **Streamline Data** view.
2. Select an event from the **Events to Collect** list.
3. Click the **Toggle event-based sampling** button.

   If the **Toggle event-based sampling** button is not selectable after you have selected an event, then that event does not support event-based sampling.
4. Enter a value in the newly activated **Threshold** field.

   Avoid setting a very low threshold for high frequency events. If you enter a threshold value that generates too many samples, the capture could fail, and you might have to restart your target. To find an appropriate value to enter in the **Threshold** field, turn off event-based sampling to run a standard, time-based profile with the event counter that you want to use enabled. Look at the resulting **Timeline** view and note the peak per-second value in the chart for your counter. Your target for the **Threshold** field is 1000 samples per second, so if the peak for that event is 2000000, a good value to insert in the **Threshold** field is 2000.

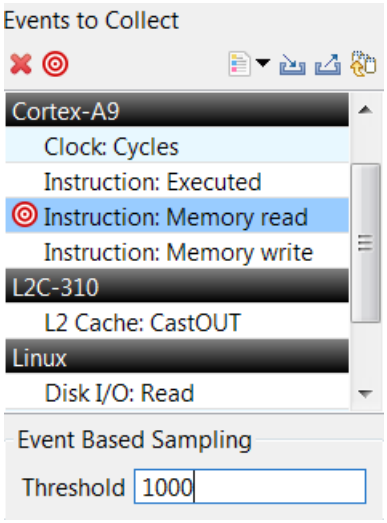


Figure 5-4 Setting up event-based sampling

Chapter 6
Live View and Timeline View

Describes the **Live** and **Timeline** views, which display charts showing the data collected during the capture session. **Live** view is displayed while the capture takes place, and charts the data in real time. **Timeline** view is displayed after the capture session ends and the data has been analyzed. It provides additional information in a details panel.

It contains the following sections:

6.1 Live view overview

When you trigger a capture session in Streamline, the **Live** view opens automatically. It charts capture data in real time and provides a list of processes alongside usage data.

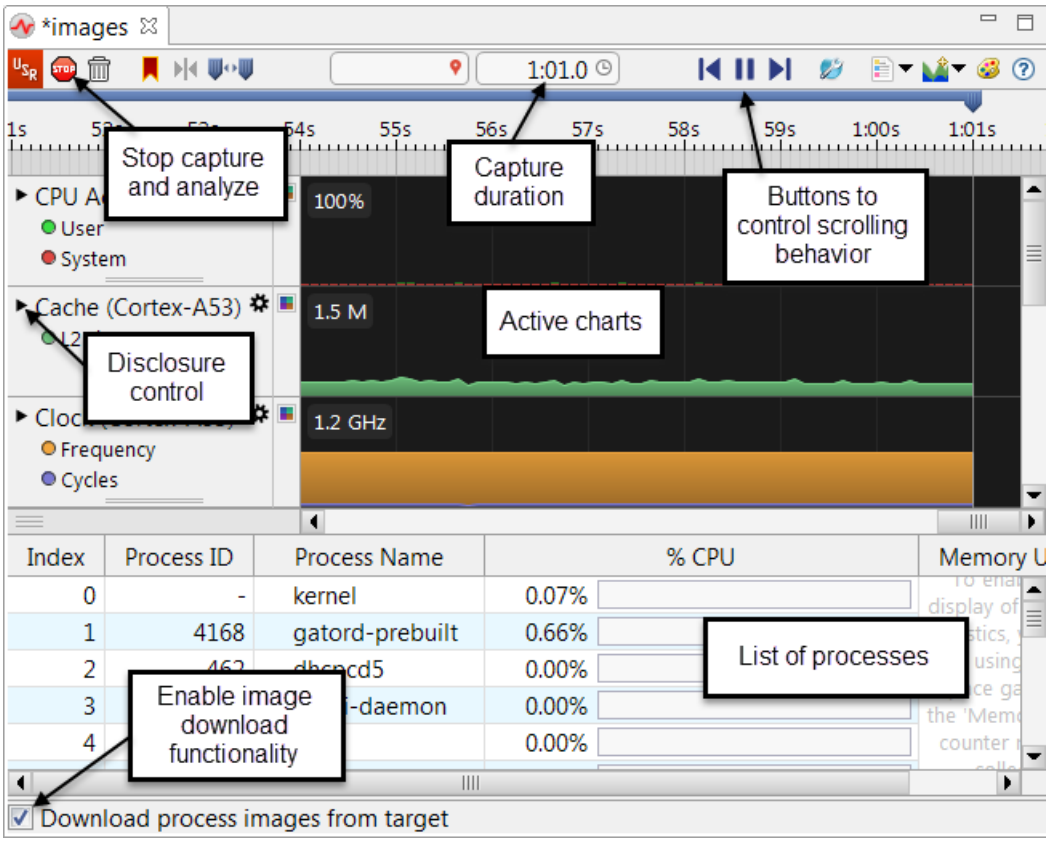


Figure 6-1 Live view

You can use the chart handles to reorder the active charts in **Live** view, and you can use the bottom of the handle to resize the height of a chart. Disclosure controls enable you to view chart data for an aggregate of all cores or for each core, for charts that support per-core data.

Below the charts is a list of the known processes and usage statistics for each of them. This list updates continuously as long as the capture session is running. When a process dies, its name and ID remain in the list but are shown in gray, for example, `xaos` in the following list:

Index	Process ID	Process Name	% CPU
0	24674	gatord	0.87%
1		kernel	0.06%
2	24666	xaos	
3	1	init.sysvinit	0.00%
4	24236	sshd	0.00%

Figure 6-2 Dead process in Live view

Note

To display data in real time in the **Live** view, you must set **Buffer Mode** to **Streaming** in the **Capture & Analysis Options** dialog box. If **Streaming** is not active during a capture session, **Live** view still appears, but without any real-time display of data.

At the bottom of the view is a checkbox for downloading the process images from the target. Select **Download process images from target**, then select the images to download in the dialog that opens when you stop the capture.

The toolbar of the **Live** view displays the real-time duration of the live capture in seconds. If there is latency in the data passing from the target, this displays the text **Target latency - resyncing…** beside the capture duration, and the **Live** view halts scrolling until the data can catch up.

If you selected the **Discard Data** option in the **Capture & Analysis Options** dialog box, the **Stop capture and analyze** button in the toolbar is inactive and appears grayed out. This means that when the capture session terminates, Streamline discards the data.

The **Live** view is intended to give feedback during a capture session. The full functionality and data provided by the various views of Streamline are only available after you have stopped your capture session and Streamline has processed the data in the capture.

Related concepts

6.5 Image download on page 6-94.

6.2 Timeline view overview

After you have successfully generated a report, Streamline opens it automatically and displays the **Timeline** view. It provides you with high level information about the performance of your target during the capture session.

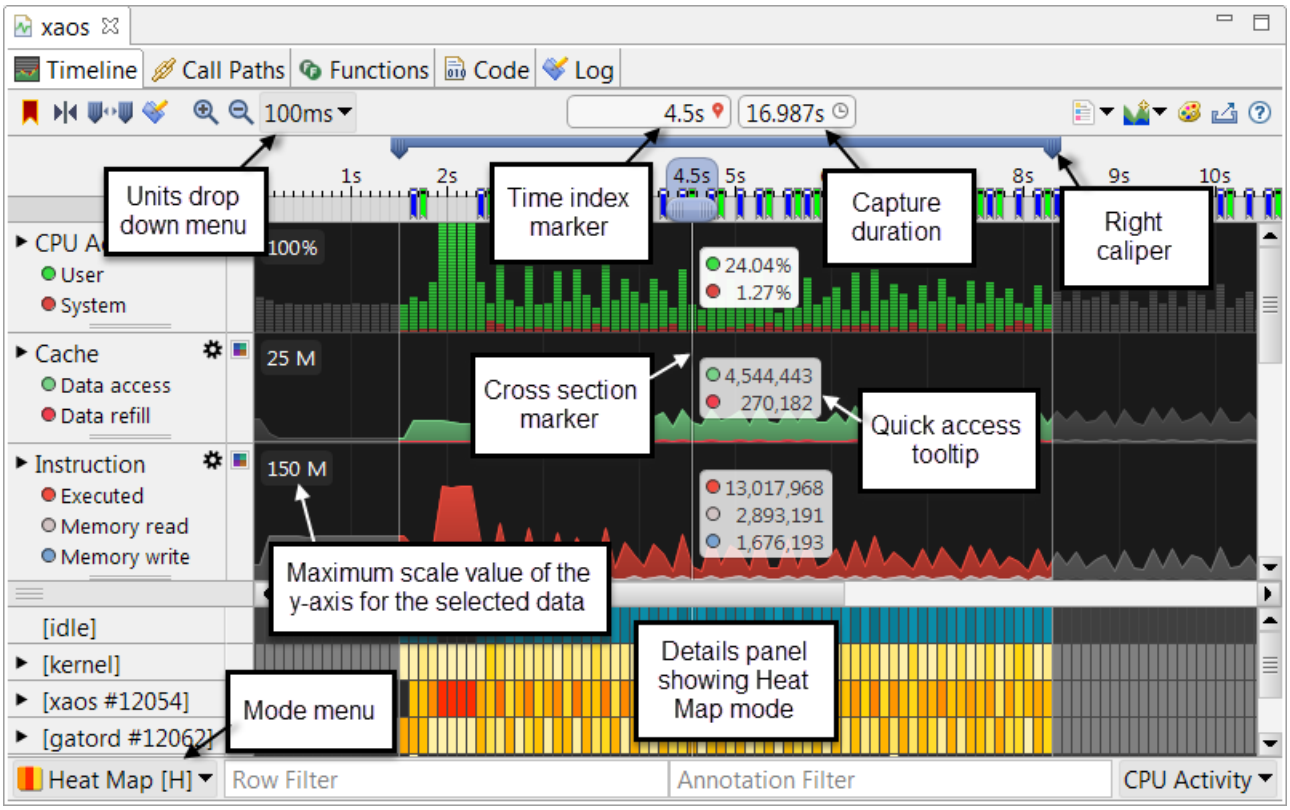


Figure 6-3 Timeline view

The **Timeline** view has two main sections, which are separated by a horizontal scrollbar. Charts appear in the upper section and the details panel appears in the lower section. The information in the details panel is dependent on the current selection in the mode menu, located in the lower left of the **Timeline** view.

The **Timeline** view breaks up its data into bins, a unit of time defined by the units drop down menu at the top of the view. For example, if 50ms is selected in the menu, every color-coded bin in the details panel represents data captured during a 50ms window. The charts scale according to the filter applied to the bins in the chart configuration series options. For example, if you select **Average**, the y-axis scales to the maximum average value for the bins in the selected range.

Related concepts

6.3 Charts on page 6-81.
6.6 Details panel in the Timeline view on page 6-97.

6.3 Charts

The charts that are displayed in the top half of the **Live** view and **Timeline** view depend on the counters that you have defined using the **Counter Configuration** dialog box and any customizations you have made using the chart configuration controls.

6.3.1 Default charts

Streamline collects data for charts from the hardware and software performance counters that you have selected. The target hardware determines which counters are available for selection.

Here are some of the default charts in the **Live** and **Timeline** views:

CPU Activity
: The percentage of the CPU time that is spent in system or user code, the remainder being idle time.

Cache
: The number of memory reads or writes that cause a cache access or a cache refill of at least the level of data or unified cache closest to the processor.

Clock
: The number of cycles that are used by each core.

Disk I/O
: The number of bytes read from or written to disk.

Instruction
: An approximate count of the total number of instructions that each core executes, and the number of instructions that read from or write to memory.

Interrupts
: Maps the amount of both soft IRQs and standard, hardware IRQs. Soft IRQs are similar to IRQs, but are handled in software. Soft IRQs are usually delivered at a time that is relatively convenient for the kernel code.

Memory
: Charts the available system memory over the time of the execution.

Related concepts

5.2 Counter Configuration dialog box structure on page 5-69.
6.4.1 Chart configuration panel on page 6-87.

Related references

6.8 Toolbar options in the Live and Timeline views on page 6-107.

6.3.2 Charts that are specific to ARM® Mali™-based targets

If you have a Mali-based target that is configured to support Mali GPU-specific profiling, and is running a version of gator that supports the target GPU, the **Live** and **Timeline** views provide GPU charts that are specific to Mali-based targets.

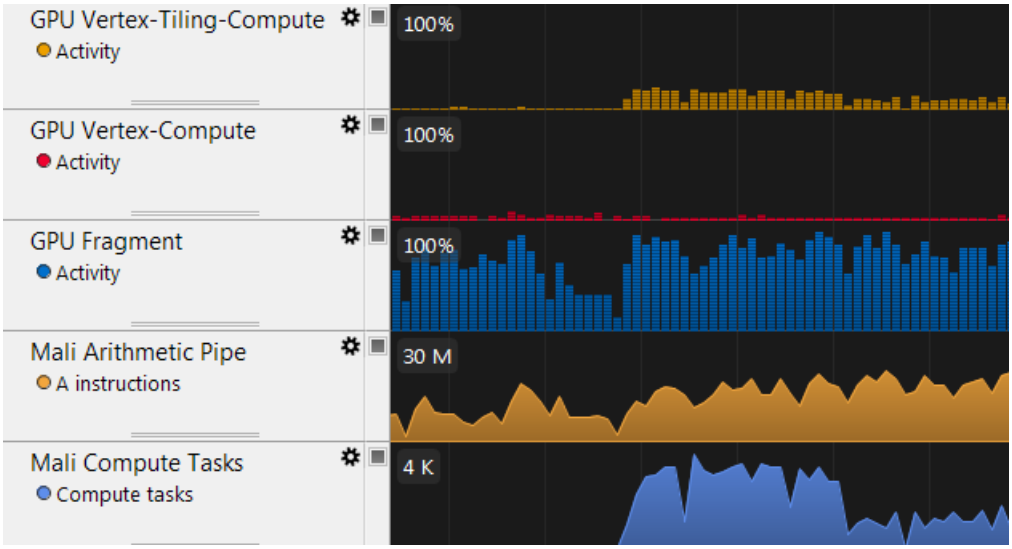


Figure 6-4 Charts specific to a Mali Midgard-based target

If you have run the capture session on either a Mali-400 or Mali-450 based target, the following charts are added to the default set of charts:

GPU Vertex chart
: Streamline reports whether the status of the Mali vertex processor is idle or active. The load on the vertex processor is proportional to the number of vertices and the complexity of the shader that is used to transform their coordinates.

GPU Fragment chart
: Streamline reports whether the status of the Mali fragment processor is idle or active. The load on the fragment processor is proportional to the number of pixels to be rendered and the complexity of the shader that is used to determine the final pixel color. Pixels that are rendered include on and off screen pixels and the additional pixels that are required for super-sampling.

If you have run the capture session on a Mali Midgard-based target, Streamline includes the following Mali-specific chart types:

GPU Fragment
: Reports whether job slot 0 is occupied or idle. This chart is exclusively for fragment processing.

GPU Vertex-Tiling-Compute
: Reports whether job slot 1 is occupied or idle. This chart generally, though not exclusively, corresponds to vertex processing.

GPU Vertex-Compute
: Reports whether job slot 2 is occupied or idle. This chart generally, though not exclusively, corresponds to compute work.

These counters are included in the capture by default, but you can exclude them using the **Counter Configuration** dialog box.

6.3.3 Moving and re-sizing charts

Each chart in the **Live** or **Timeline** view has a box on the left that shows the chart title, the names of the series in the chart, and a color-coded key for each series.

When you click within a box, it becomes a handle which you can use to drag and drop charts into a preferred order.

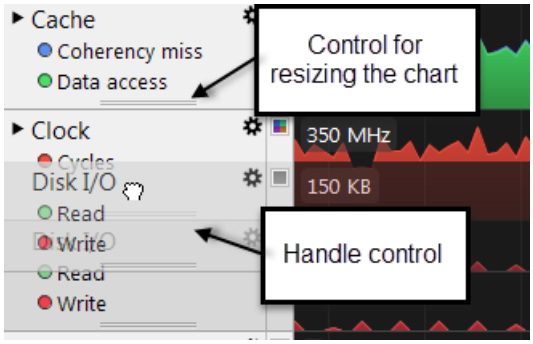


Figure 6-5 Moving a chart using the handle control

To re-order the charts, click and drag the handle control, then release it where you want it placed. To hide a chart, drag it to the bottom of the charts and drag the divider bar up until it is hidden.

You can also re-size any chart in the **Live** or **Timeline** view using a control on the bottom edge of the chart handle control. All series expand to fill the new height. Increasing the size of a chart provides a higher level of graphical detail, highlighting the variance in values.

You can also customize the width of the chart handle. To do so, click and drag the right edge of any handle to your desired width.

6.3.4 Chart disclosure control

The chart disclosure control appears in the upper left corner of a chart handle in the **Live** or **Timeline** view where per-core data is present.

By default, the control points right, to show chart data for an aggregate of all cores. If you click the control, it points downwards and the chart breaks down into multiple sections, one for each core on your target.

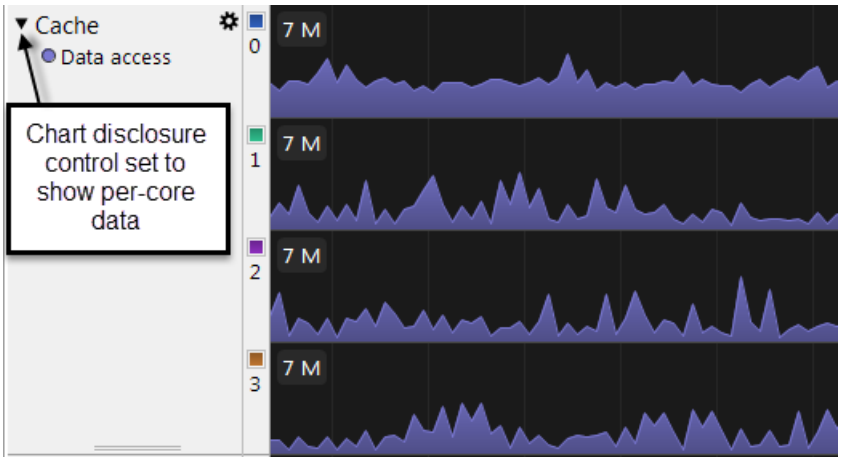


Figure 6-6 Using the chart disclosure control to show per-core data

Hover over a core number to see a tooltip that shows the name of the core.

Click the button again and the arrow returns to its default state.

6.3.5 Per-cluster charts

For systems with multiple clusters, for example ARM big.LITTLE™ systems, the **Live** and **Timeline** views display cluster-specific data in per-cluster charts.

For example, **CPU Activity** and **Clock Frequency** are per-core counters. In a big.LITTLE system, they are shown in per-cluster charts, with the cluster name shown in the chart title:

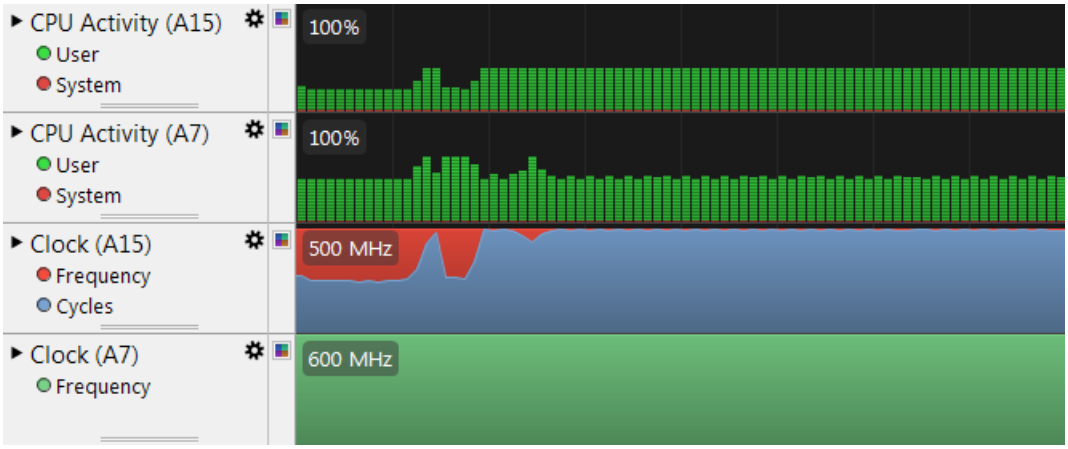


Figure 6-7 Per-cluster charts

Clicking the chart disclosure control for a per-cluster chart displays charts for each core in that cluster only.

Note

To capture complete cluster information, you must use gator version 24 or later and the following Linux kernel versions:

- Version 4.2 or later for ARMv7 targets.
- Version 4.4 or later for ARMv8 targets.

With incomplete cluster information, Streamline displays all cores when you click the chart disclosure control.

Related concepts

6.3.4 Chart disclosure control on page 6-83.

6.3.6 Quick access tooltips

Hover over any of the charts in the **Live** or **Timeline** views and a tooltip appears, displaying values and key colors specific to that chart.

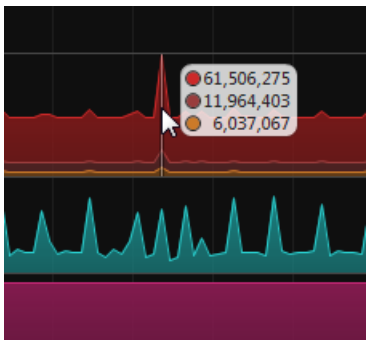


Figure 6-8 Quick Access tooltip

Clicking on a chart either displays or moves the Cross Section Marker, which shows values and key colors for all of the charts in the **Live** or **Timeline** view.

6.3.7 Cross Section Marker

The Cross Section Marker is a versatile tool for looking at specific ranges of data in the **Live** and **Timeline** views. It starts one bin wide but can be re-sized using the handles on both sides or moved left

and right using the middle of the marker. An overlay shows you data pertinent to the current range covered by the Cross Section Marker.

By default, the Cross Section Marker is inactive. To activate or move it, click anywhere in the charts or in the graphic portion of the details panel in the **Timeline** view. The Cross Section Marker appears where you clicked and provides data specific to the bin where you placed it. The position of the Cross Section Marker in **Live** view is preserved in **Timeline** view.

You can also stretch the Cross Section Marker using the handle that is located above the charts in the **Live** and **Timeline** views. Click and drag on either side of the handle to expand it. Once expanded, you can move the Cross Section Marker left and right by clicking and dragging it. The information contained in the details panel in **Processes** and **Samples** modes in the **Timeline** view relates only to the window of time defined by the Cross Section Marker.

Note

Unlike the filter controls, moving and expanding the Cross Section Marker does not have an effect on the data in the other report views.

In cases where you set a Cross Section Marker border, then change to a different level of magnification where the Cross Section Marker border would not sit precisely, the border is displayed as a blurred line. This indicates that the offset of the Cross Section Marker is not perfectly aligned with the bin at the current magnification level, but is fractionally within it.

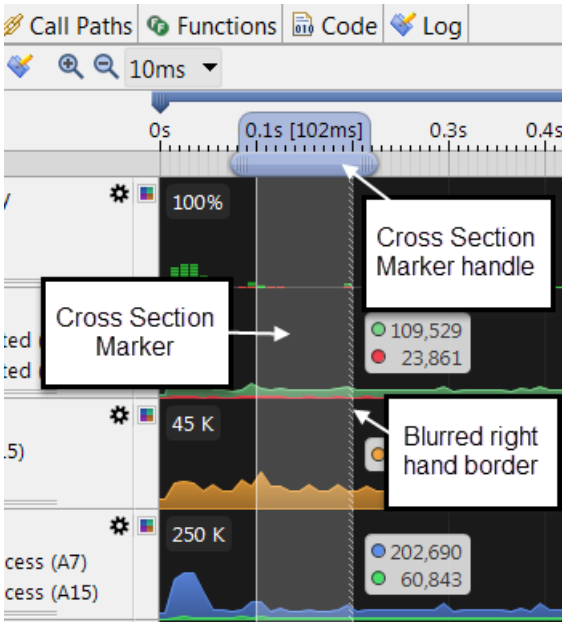


Figure 6-9 Cross Section Marker blurred border

6.3.8 Filtering using the caliper controls

The **Live** and **Timeline** views contain calipers that you can use to specify a window of time on which you want to focus. Streamline updates each of the report views based on the position of the calipers.

To set filtering using the caliper controls, follow these steps:

Procedure

1. Set the left caliper in either of the following ways:
   - Drag the left caliper control to a location. The caliper control is blue and is located in the **Live** or **Timeline** ruler.
   - Right-click anywhere in the **Live** or **Timeline** view and select **Set Left Caliper** from the contextual menu.

2. Repeat the process for the right caliper.

The caliper controls narrow the focus of the report. Only data relevant to this interval appears in the other views, so the **Call Paths**, **Functions**, and **Code** views update when you move the calipers.

Note

The position of the calipers in the **Live** view is preserved in the **Timeline** view.

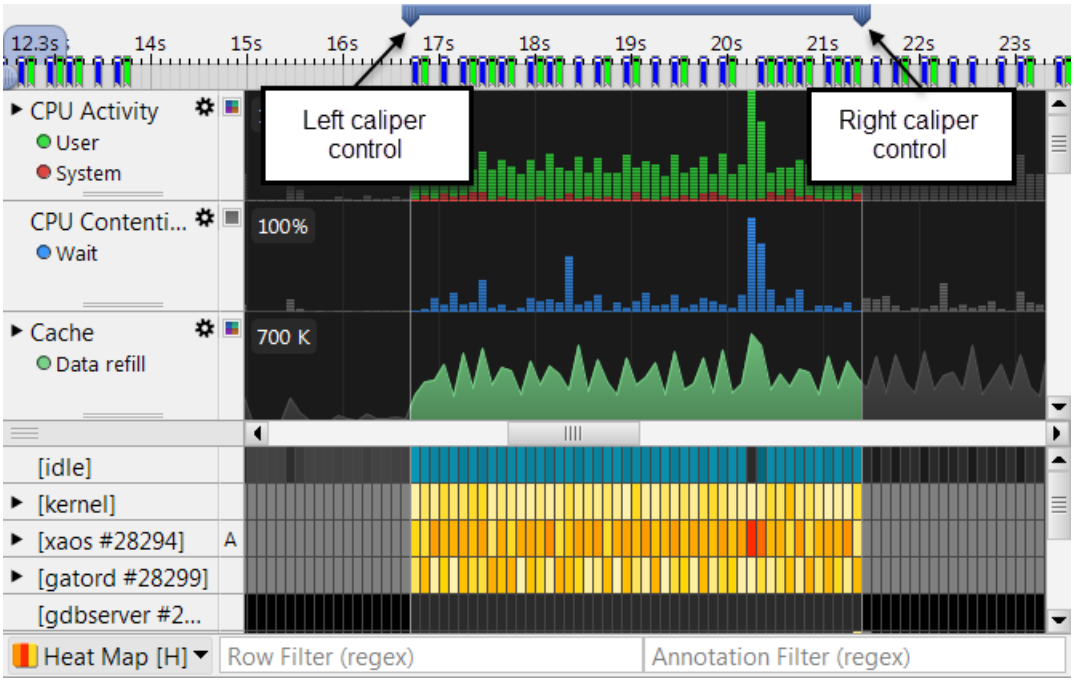


Figure 6-10 Using the calipers to filter

Related references

6.8 Toolbar options in the Live and Timeline views on page 6-107.
6.9 Contextual menu options in the Live and Timeline views on page 6-111.
6.10 Keyboard shortcuts in the Live and Timeline views on page 6-113.

6.4 Chart configuration

Streamline allows you to configure many aspects of the charts displayed in the **Live** and **Timeline** views, including the colors, titles, and data sets used by each series.

6.4.1 Chart configuration panel

Many of the charts in the **Live** and **Timeline** views have a button, located near the top right of the chart handle, which opens and closes the chart configuration panel.

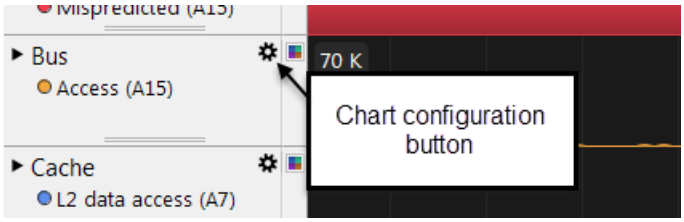


Figure 6-11 Chart configuration button

The chart configuration panel, shown below, contains the following sections:

- A toolbar section, which contains controls that apply to the chart as a whole.
- A series section, which contains controls that apply to the individual series in the chart.

For example, the toolbar section shown below defines the chart title as **Bus**, and the chart type as Stacked. The series section defines the **Access (A15)** series.

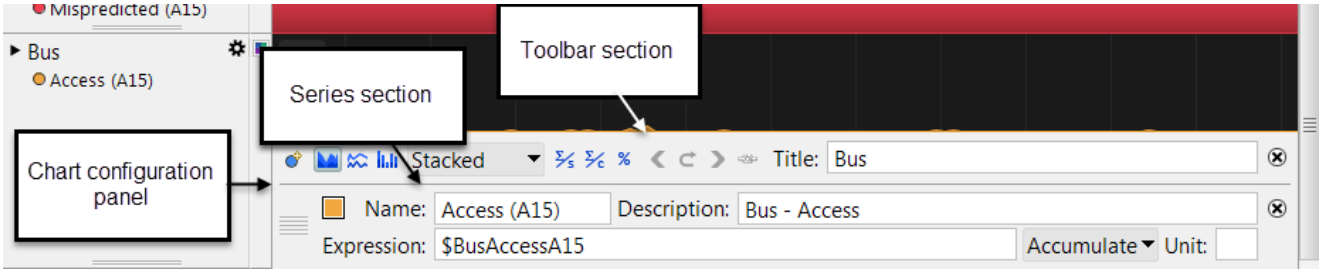


Figure 6-12 Chart configuration panel

Any updates you make to the chart configuration in the **Live** view are preserved and displayed in the **Timeline** view.

6.4.2 Chart configuration toolbar options

The toolbar of the **Chart Configuration** panel defines options that apply to all series in a chart.

It has the following options:

Create a new series
: Adds an empty series to the chart.

Chart Type
: Use the Type buttons on the left side of the toolbar to choose between one of the following chart types:

Filled

In a filled chart, each series is displayed as an area filled with the color specified in the chart configuration.

Figure 6-13 Filled chart

Line

In a line chart, each series is displayed as a colored line.

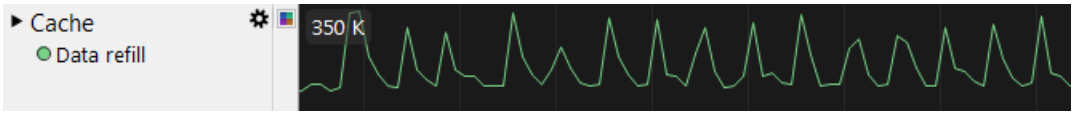


Figure 6-14 Line chart

Bar

In a bar chart, each series is displayed as a colored bar. Each bar in the chart represents a time bin.

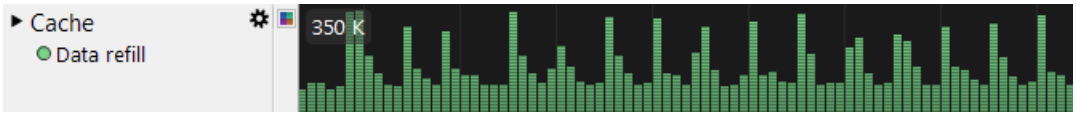


Figure 6-15 Bar chart

Series composition

Use the drop-down menu to select one of the following options:

Stacked

In a stacked chart, the data for different series are stacked on top of each other. So, the highest point of a stacked chart is an aggregate of data from all of the series contained in the chart. For example, if the first value of series A is three and the first value of series B is five, the first data point in the stacked chart that contains these series is eight.

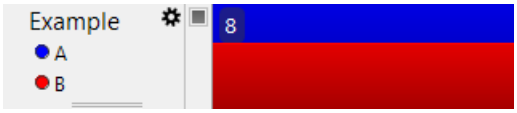


Figure 6-16 Stacked chart

Stacked charts are appropriate when the events are counted in exactly one of the series in a chart. For example, this is useful in a case where a chart contains both Data Read Hits and Data Read Misses.

Overlay

In an overlay chart, the different series overlap each other. The front-to-back ordering is determined by their position in the chart control. To prevent data from being obscured by other series, as shown in the following figure, ensure series with larger values are placed above series with lower values in the chart control. Drag and drop series controls using the chart configuration panel to reorder them.

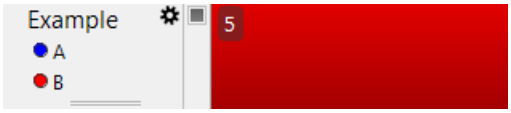


Figure 6-17 Data from series A obstructed by data from series B

Overlay charts are appropriate when some of the events are counted in more than one series in the same chart. Data Read Requests and Data Read Hits are a good example of this.

Logarithmic

Shows all data points on a log10(y) scale where y is each Y-axis data point in the chart. This Y-axis modifier is useful when there is a huge difference between the minimum and maximum values in a chart. In charts with these huge deltas, it can be hard to differentiate between the lower value data points in the chart. Setting the Y-axis modifier to Logarithmic can give you a better visual representation of your data in these cases. A logarithmic chart displays with a series of horizontal lines in Streamline, each line represents an increase in value by a power of ten, so that if data point A sits exactly one line higher than data point B, A is ten times higher than B.

Note

If the actual value of a data point in a Streamline logarithmic chart is less than one, it appears in the Streamline chart like a zero value. This is because the logarithmic value of such a data point would be a negative number.

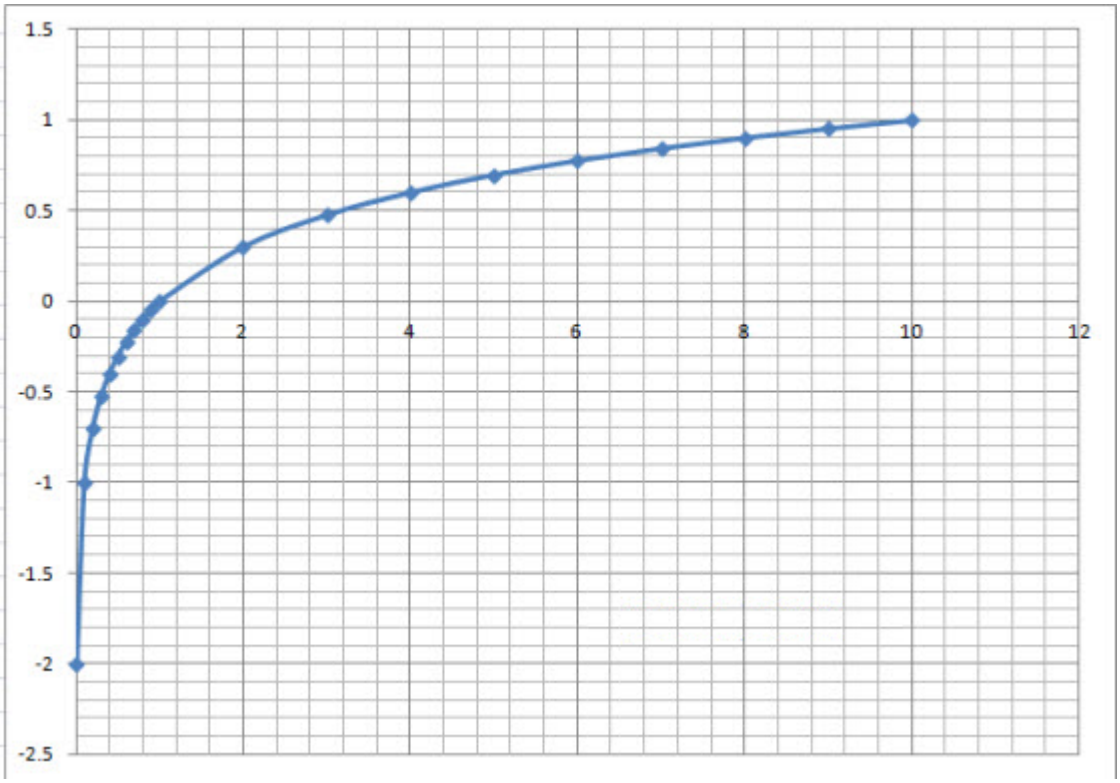


Figure 6-18 Logarithmic scale

Average Selection
If you select this option, the Cross Section Marker overlay shows the average value of all bins included in the selection. If not selected, the overlay shows the total value of all bins in the selection.

Average Cores
Select this option to plot values in a multi-core chart as the average of all cores, unless you have used the multi-core disclosure control to show metrics per core. If not selected, the multi-core chart shows the total for all cores.

Percentage
Select this option to plot values as a percentage of the largest value in the chart.

The following energy offset buttons allow you to align energy data with the other data in the view.

Note

These buttons are disabled if there is no energy data in the chart. To capture energy data you must have either an Energy Probe or a supported NI DAQ device.

Offset energy chart data to the left
Manually adjusts the energy data to the left.

Reset energy chart offset to zero
Returns the energy data to its original position.

Offset energy chart data to the right
Manually adjusts the energy data to the right.

Link adjustments to other charts
Links the adjustments made to the current energy chart to the other energy charts in the capture.

Title
Use this field to give the chart a title. The title appears at the top of the chart handle.

Remove Chart
Removes the chart from the view. If you have saved the chart in a chart configuration template, you can add it back to the view later using the **Switch and manage templates** button.

Related concepts

11.1 Energy Probe overview on page 11-152.

6.4.3 Chart configuration series options

Each chart in the **Live** or **Timeline** view contains one or more series or sets of data. Each series has a set of options, specific to that series that enable you to customize both the data used to plot the chart and its look and feel.

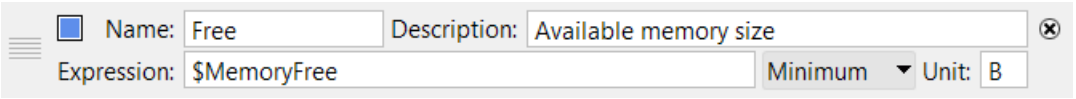


Figure 6-19 Series options

You can use the handle on the left side of each series control to reorder the series in the chart. You can also use it to move or copy the series to another chart. To move the series, drag and drop it. To copy the series to another chart, hold down the **Ctrl** key while dragging it. When moving the series to another chart, make sure the chart configuration panel is open for the destination chart.

Each series in a chart contains the following options:

Color
To change the color of a series, click on the color box in its series control. This opens a **Color** dialog box.

Name
Enter a name for the series. This name appears next to the chart color in the chart key.

Description

Enter a description for the series. When you hover over the series title or color, a tooltip appears, containing the description defined here.

Expression

Use this field to define the data set that the series uses. Press **Ctrl + Space** or the `$` symbol to activate a drop-menu that shows you a list of counters. You can select a counter in this **Content Assist** list to see its description. Click on a counter to add it to the **Expression** field. You can create an expression using more than one counter by using a combination of counter names and any of the following operators: `>, <, >=, <=, ==, !=, ||, !, &&, %, *, /, +, -`. You can use parentheticals to define the order of operation.

In addition to the mathematical and comparative operators, you can use the following functions in the **Expression** field:

`if`

Evaluates whether a condition is true or false before applying an effect. Usage: `if(x, y, z)`, where `x` is the expression to be analyzed. The result is `y` if `x` is non-zero or `z` if `x` is zero. Only one of `y` or `z` is evaluated.

`abs`

Returns the absolute value of the numeric expression specified as the parameter. Usage: `abs(x)`, where `x` is a numeric expression.

`ceil`

Returns the smallest integer that is greater than or equal to the numeric expression given as a parameter. Usage: `ceil(x)`, where `x` is a numeric expression.

`floor`

Returns the largest integer that is less than or equal to a numeric expression given as a parameter. Usage: `floor(x)`, where `x` is a numeric expression.

`max`

Compares the arguments and returns the greater value. Usage: `max(x,y)`, where `x` and `y` are numeric expressions.

`min`

Compares the arguments and returns the lesser value. Usage: `min(x,y)`, where `x` and `y` are numeric expressions.

`round`

Returns a numerical value rounded to an integer. Usage: `round(x)`, where `x` is a numeric expression.

——— **Note** ———

When used independently from source data, entering constants in the **Expression** field can yield inconsistent results.

———————————

Filters drop-down menu

Depending on the class of the counter selected, you can select one of the following filters to apply to all the values in the series using the drop-down menu in the series options panel:

Average

Works the same way as `Minimum`, except that it displays an average value for each time bin.

Accumulate

Displays the accumulated value of all the samples in the time bin.

Hertz

Converts the counter to a rate. It takes the value for each time bin and divides it by the unit of time represented by the time bin, then converts it up to seconds. You can use it to convert a cycles count into cycles per second.

Maximum

Works the same way as `Minimum`, except that it displays a maximum value for each time bin.

Minimum

Displays the minimum values for the counter for each time bin in the current zoom level of the **Timeline** view. So, if the current zoom level of the **Timeline** view is one second, `Minimum` displays the lowest value recorded for any millisecond within that second.

Note

In most cases, `Minimum` provides the lowest value for each millisecond within any time bin. If you have **High Resolution Timeline** enabled, `Minimum` provides the lowest value per microsecond, if the given counter provides that level of detail.

Unit

Enter the unit type for the series. The value you enter in this field appears when you use the **Cross Section Marker** to select one or more bins.

Remove Series

Removes the current series from the chart.

Related concepts

6.4.1 Chart configuration panel on page 6-87.
6.12 Counter classes on page 6-115.

Related references

6.4.2 Chart configuration toolbar options on page 6-87.

6.4.4 Chart configuration templates

A chart configuration template is a set of pre-configured charts that you can apply to the report that is displayed in the **Live** and **Timeline** views.

To create a template, configure the charts in the report as required, then click **Switch and manage templates** () and select **Save as…**.

You can apply templates to existing reports. You can also apply templates to new captures using the **Counter Configuration** dialog. This feature ensures that the counters required by the charts in the template are included in the capture.

Note

atrace counters and visual annotations are not supported in templates, although applying a template does not remove a visual annotation chart from the display.

To apply a template to an existing report, select the template from the drop-down list using the **Switch and manage templates** button in the **Live** and **Timeline** views. To revert the report to its default chart configuration, select **Default Template**.

If the capture does not include a counter that a chart in the template requires, a warning icon is shown in the chart handle. To see which counter was missing, hover over the icon.

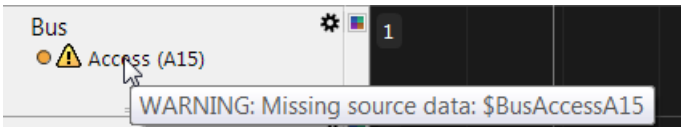


Figure 6-20 Warning message for a missing counter

Templates are stored by default in the following locations:

- On Windows, in `C:\Users\<username>\Documents\Streamline\Templates`.
- On Linux and Macintosh, in `~/Documents/Streamline/Templates`.

To add a template to the list from a different location, click **Switch and manage templates** then select **Install and Load…**.

Related references

5.5 Counter Configuration dialog box settings on page 5-73.
6.8 Toolbar options in the Live and Timeline views on page 6-107.

6.5 Image download

Download process images from the target device at the end of a live capture session using the image download functionality.

To trigger the image download when the session ends, select the **Download process images from target** checkbox at the bottom of the **Live** view.

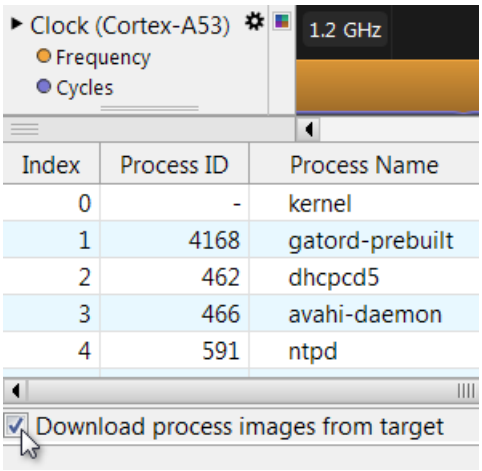


Figure 6-21 Select Download process images from target.

When you stop the capture, the **Image Download** dialog opens.

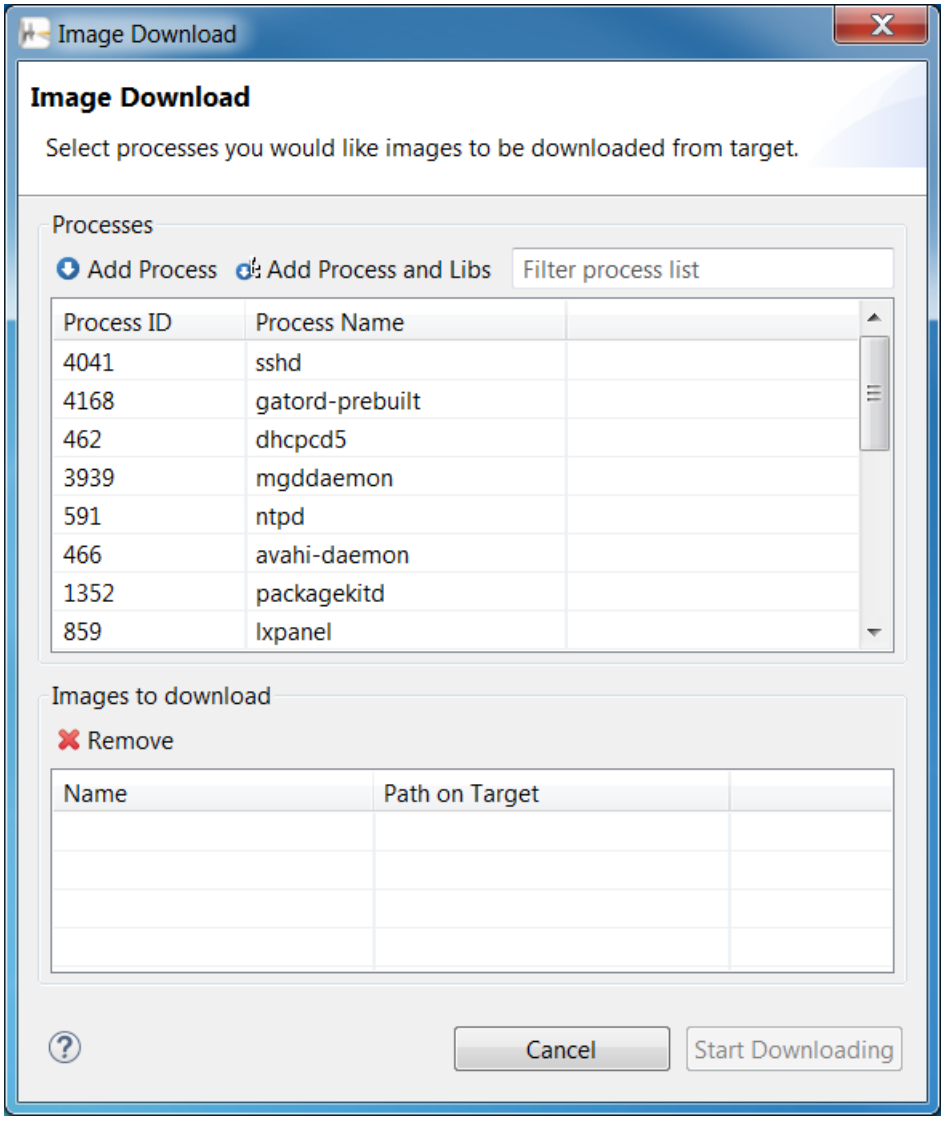


Figure 6-22 Image Download dialog

The upper part of this dialog lists the processes that are running on the target. The lower part lists the process images that you have selected for download.

Select a process then click **Add Process** to add it to the list of images to download. To remove an image from this list, select the image then click **Remove**.

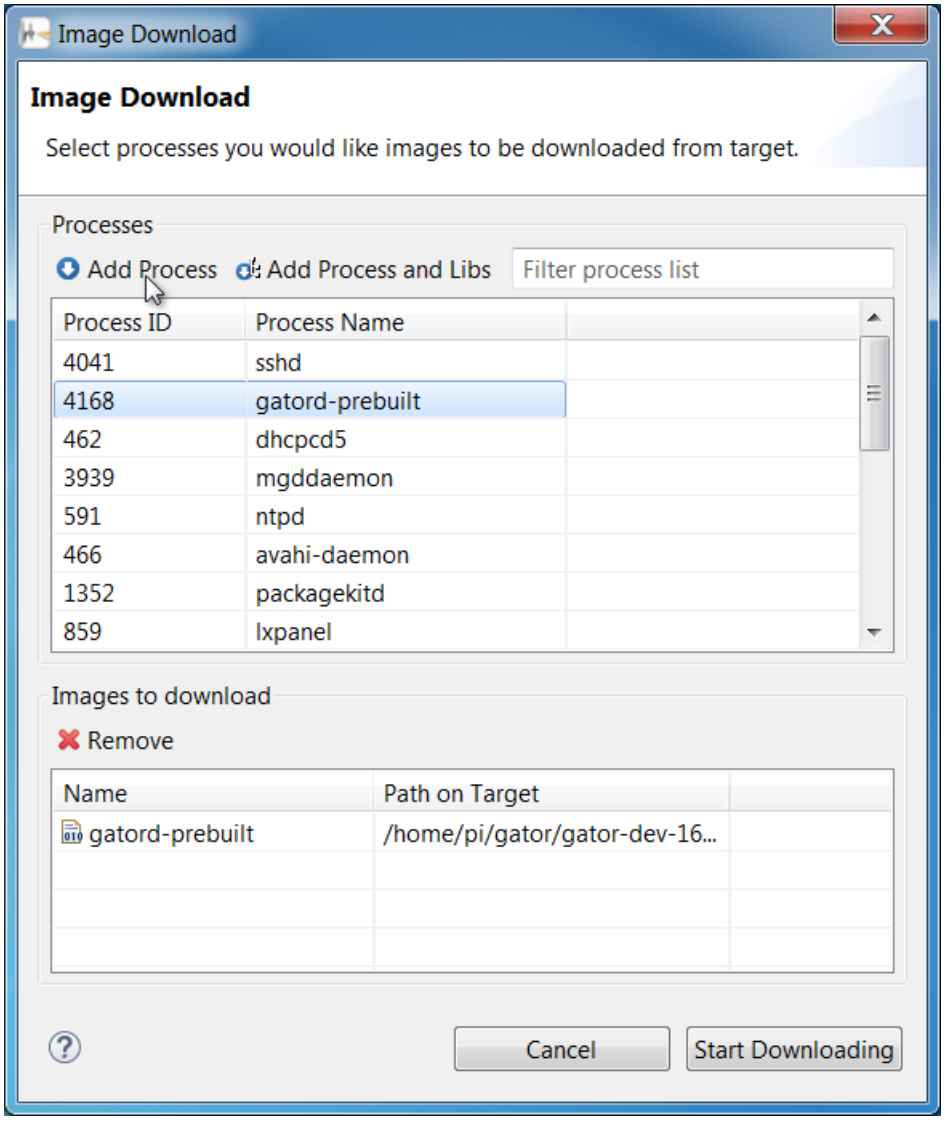


Figure 6-23 Adding a process

Add Process and Libs also adds any libraries on which the selected process depends.

To download the selected images, click **Start Downloading**.

6.6 Details panel in the Timeline view

The details panel of the **Timeline** view enables you to switch between different modes using the menu in its bottom left corner. Each mode displays a different set of data to supplement the charts.

6.6.1 Details panel modes

The details panel has the following modes:

Heat Map mode
The **Heat Map** shows you a list of threads and processes that were active during the capture session in each time bin, alongside a color-coded heat map. Colors range from white to red, and the darker the color, the more activity caused by the thread or process in that bin.

Core Map mode
The **Core Map** mode is similar to **Heat Map** mode. It shows a list of threads and processes, but instead of a color-coded heat map, it provides you a colored activity map based on which core was responsible for the majority of the activity for each thread or process. This mode is not available for single core hardware targets.

Cluster Map mode
Identical to **Core Map** mode, except that **Cluster Map** mode provides a color-coded activity map based on clusters. This mode is only available for targets that have multiple core clusters.

Samples mode
Samples mode lists the functions with samples in the currently selected cross-section. Double-click on a function to jump to the relevant row in the **Functions** view.

Processes mode
Processes mode provides a list of all processes alongside a process ID, the average percentage of CPU used and the maximum amount of memory used. Like **Samples** mode, the data shown is dependent on the current selection of the cross-section marker.

OpenCL mode
This mode displays the OpenCL commands being executed on each thread over the course of a capture session and shows dependencies between commands. It is only available for Mali Midgard targets.

Images mode
This mode is only available if the capture contains visual annotations. It displays an enlarged version of the selected image in a visual annotation chart.

The map modes and **OpenCL** mode have a filter field. Enter a regular expression in the field to filter the data in the details panel. For example, the map modes show only the threads and processes whose name matches the expression. Regular expression strings are not case sensitive.

Entries in the filter field in one of the map modes affect the other map modes only.

Related concepts

6.2 Timeline view overview on page 6-80.
6.13 Visual Annotation in the Timeline view on page 6-117.

Related references

6.10 Keyboard shortcuts in the Live and Timeline views on page 6-113.

6.6.2 Heat Map mode

Heat Map mode in the **Timeline** view shows you a list of processes and threads that were active during the capture session. The entries are derived from process and thread trace data from the Linux kernel scheduler. Weighted colors reflect the number of samples in each process or thread.

Open **Heat Map** mode using the mode menu in the bottom left of the **Timeline** view.

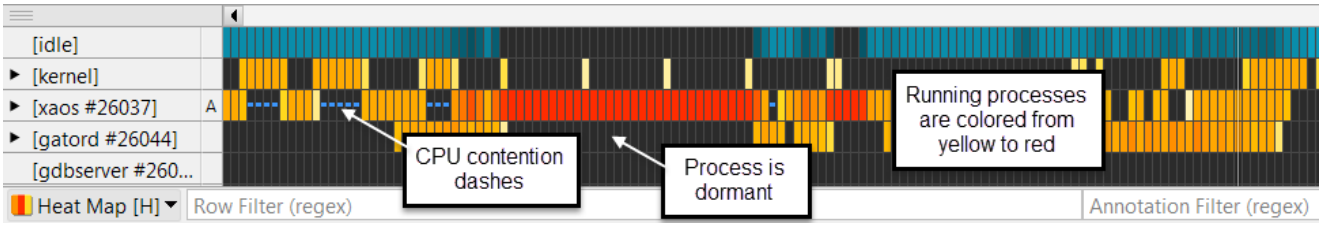


Figure 6-24 Heat Map mode

Here is what each of the colored bins in the **Heat Map** represent:

White or black, depending on the theme
: The process is not running.

Light gray or dark gray, depending on the theme
: The process has started, but is dormant. It could be sleeping, waiting on user input, or waiting for some other process to finish.

Yellow to red
: The process is responsible for a percentage of total instructions during this bin. Red indicates a higher percentage.

Note

The [idle] process is color-coded differently to the other processes in the **Timeline** view. When the system is fully idle, it is bright blue. When it is partially idle it is a lighter shade of blue, and when the system is fully active, it is gray.

Blue dashes
: CPU contention caused a delay. This can happen if there are too many processes and not enough cores to handle them.

Red dashes
: An I/O operation caused a delay. The process stopped while a read or a write to disk occurred.

If you select one or more processes or threads, the filterable chart series in the **Timeline** view update to show only activity caused by the selected processes and threads. Other chart series remain unchanged.

Each of the multi-threaded or annotated processes in the list have a disclosure control. Use the control to show each of the threads and annotations for that process. Annotations shown here can be hierarchical, with annotation groups each containing a set of channels, as defined by the macros inserted in your code.

Below the **Heat Map** are two filter fields. Add a regex to the row filter to filter the list of processes and threads. Add a regex to the annotation filter to filter the string annotations displayed in the **Heat Map**.

Figure 6-25 Filtering annotations

Click to the right of the filter fields to export the **Heat Map** data to a text file. This function exports all data that is displayed in the **Heat Map**, selected by any applicable filters, and within the calipers. The exported data is similar to the exported **Timeline** view data. Each process or thread is a column and each time bin is a row. The **Heat Map** values are exported as percentages, or the following characters:

-
: Process not present. Equivalent to white or black time bins.

I
: Process is idle.

C
: Code contention. Equivalent to blue dashes in the time bins.

IO
: Process is waiting for I/O. Equivalent to red dashes in the time bins.

Note

Annotations are not exported.

You can export the **Heat Map** for any activity source.

Related concepts

6.6.5 Selecting the activity source on page 6-100.

Related references

16.4 Exporting the Heat Map from the command-line on page 16-221.

6.6.3 Core Map and Cluster Map modes

Core Map and **Cluster Map** modes in the **Timeline** view map threads and processes to processor cores or clusters.

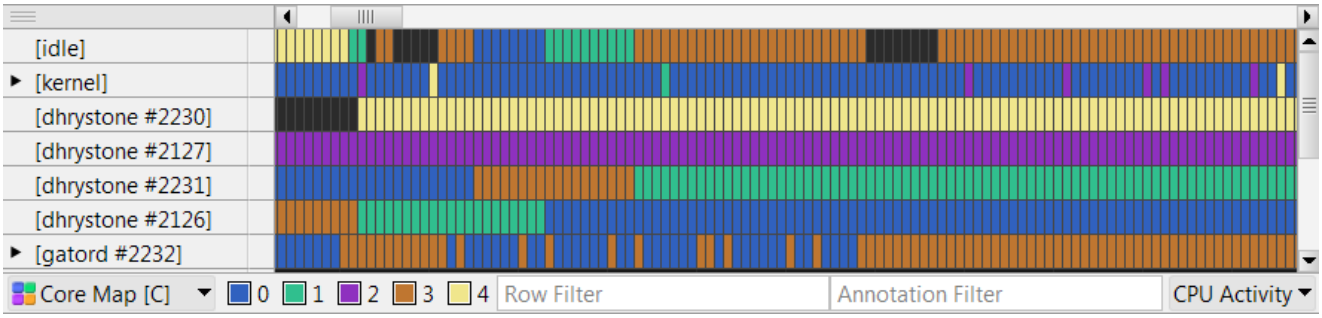


Figure 6-26 Core Map mode with five cores

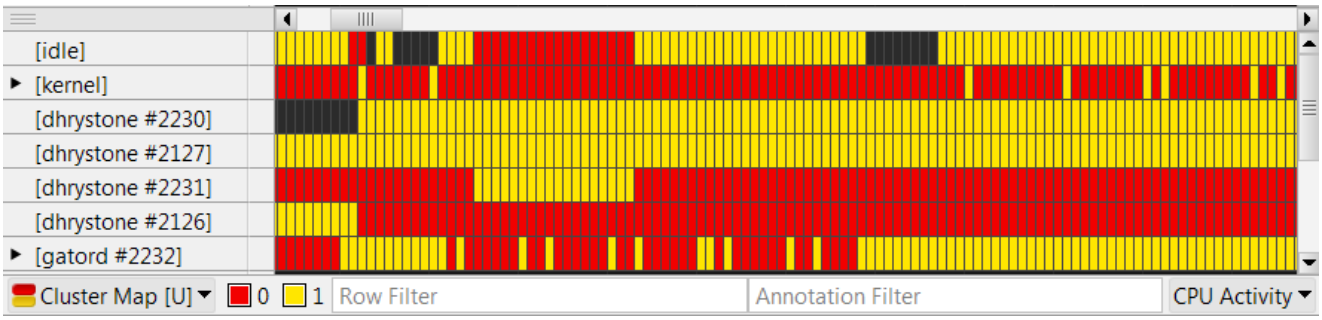


Figure 6-27 Cluster Map mode with two clusters

Note

Core Map mode is supported for captures using SMP systems and **Cluster Map** mode is only supported by hardware targets where there is more than one cluster of cores. These modes do not appear in the mode menu for captures that do not support them.

6.6.4 Filtering by threads or processes

Many chart series in the **Timeline** view can be filtered, based on the threads or processes that are selected in the **Heat Map**, **Core Map**, **Cluster Map**, or in the process list in **Processes mode**.

When there is an active selection, the affected chart series show activity that is caused by the selected processes or threads only. Filtered series are identified by the letter **F** beside the series name. Streamline still displays the total activity in dark gray so that you can visually compare the activity values of the selected processes or threads to the total.

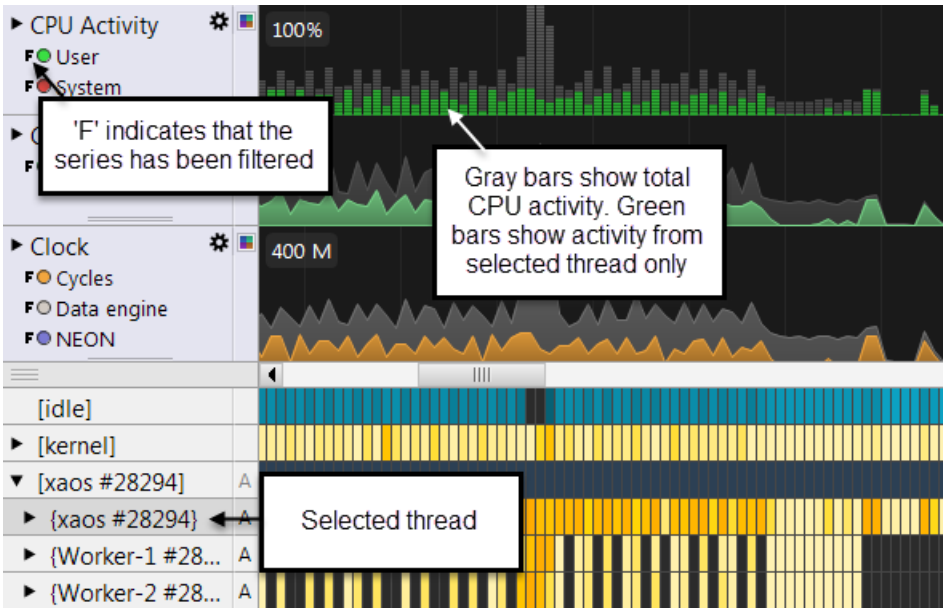


Figure 6-28 CPU Activity chart with a thread selected in the Heat Map

Like the CPU Activity chart, the GPU Vertex and Fragment charts display only activity that is initiated by the selected processes or threads. This allows you to differentiate between GPU activity that is caused by your application and activity resulting from other applications or system services.

6.6.5 Selecting the activity source

You can select an activity source from the drop-down menu in the bottom right corner of the **Timeline** view as the focus of the **Heat Map**, **Core Map**, or **Cluster Map**.

By default, **CPU Activity** is selected.

If you select a different activity source to focus on, the **Heat Map**, **Core Map**, or **Cluster Map** in the details panel updates to show a map of threads and processes for the newly selected source. For example, if you select **GPU Fragment**, the map updates to show activity for the GPU fragment processor only. The GPU activity sources are only available if your ARM Mali-based target is configured to support Mali GPU-specific profiling.

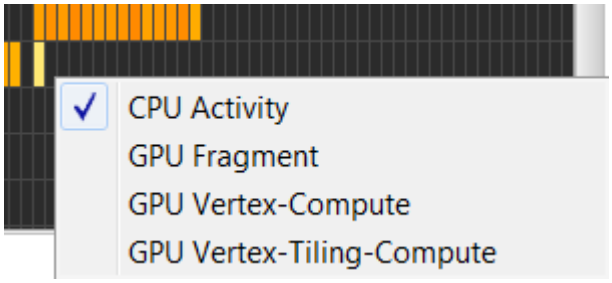


Figure 6-29 Processes focus menu

6.6.6 Samples mode

Samples mode in the **Timeline** view lists all functions in which one or more samples occurred in the time window that is covered by the Cross Section Marker.

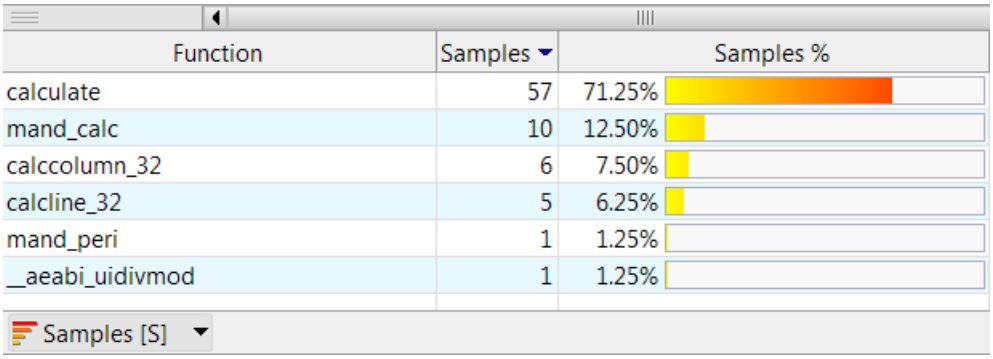


Figure 6-30 Samples mode

The details panel has the following columns when it is in **Samples** mode:

Function
The name of the function.

Samples
The number of samples that occurred in all instances of the function in the time window that is covered by the Cross Section Marker.

Samples %
The **Samples** figure as a percentage of all samples that were taken across the selected time window.

Select one or more rows in **Samples** mode and right-click to open a contextual menu, with the following options:

Select Process/Thread in Timeline
Switches to **Heat Map** mode and highlights the processes and threads for the selected functions in the Processes section.

Select in Call Paths
Opens the **Call Paths** view and highlights the function instances that relate to the selection.

Select in Functions
Opens the **Functions** view and highlights the selected functions.

Select in Code
Opens the **Code** view and highlights the source code for the selected functions.

Edit Source
Opens the source files for the selected functions in your default code editor.

Double click on a function to jump into the relevant row in the **Functions** view.

6.6.7 Processes mode

Processes mode in the **Timeline** view provides a similar set of data to the output of the `top` command in your Linux shell. For every time bin in the **Timeline** view, it provides a list of processes along with usage information.

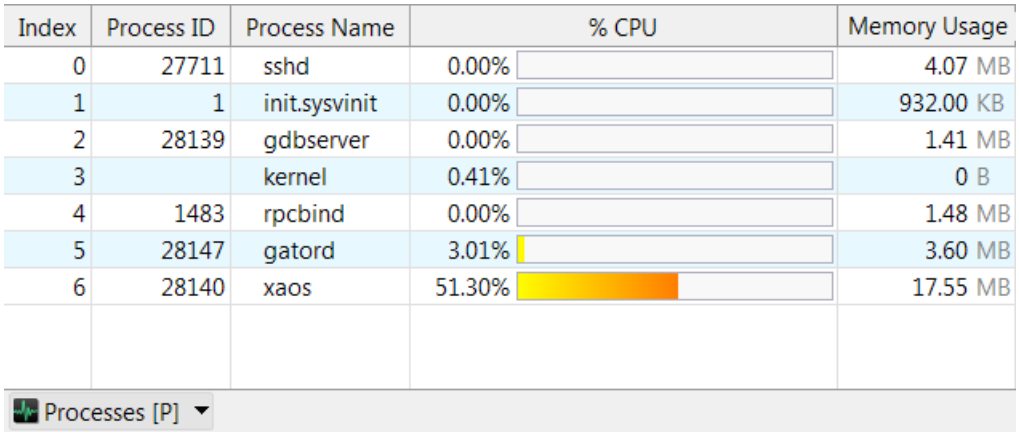


Figure 6-31 Processes mode

The details panel has the following columns when it is in **Processes** mode:

Index
The order in which rows were added to the table. This column is the default sort order.

Process ID
The PID that Linux assigned to the process.

Process Name
The name of the process.

% CPU
The average percentage of the CPU activity that this process used over the range that was selected by the Cross-Section Marker.

Memory Usage
The maximum amount of memory used by the process over the range that was selected by the Cross-Section Marker.

——— **Note** ———

The **Memory Usage** column is only displayed if you used kernel space gator and if the capture included memory counters. It is not available with user space gator.

———————————

You can sort by any of the columns in **Processes** mode. Click for a descending sort and click again to reverse the sort.

6.6.8 Images mode

Images mode in the **Timeline** view displays an enlarged version of the image that is selected in a visual annotation chart.

Images mode opens automatically if you click on an image in a visual annotation chart. If you left-click and drag the mouse left or right within the chart, **Images** mode displays each selected image in turn, creating an animation effect.

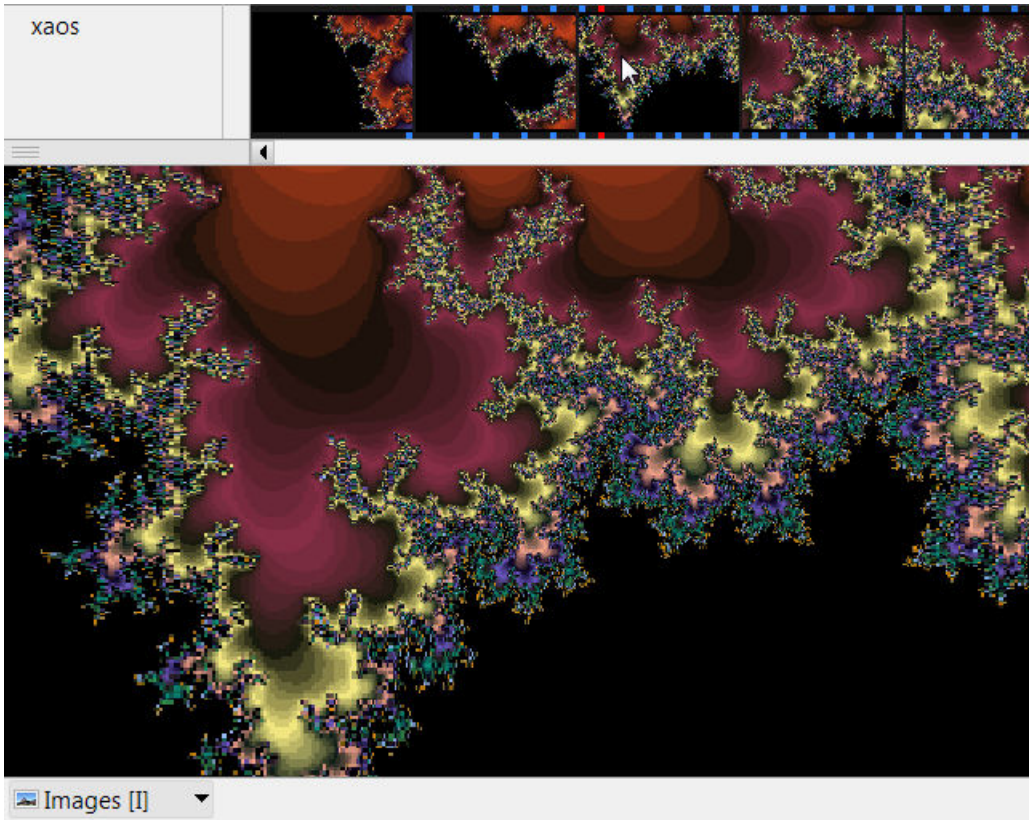


Figure 6-32 Images mode

6.6.9 OpenCL mode

OpenCL mode provides a visual representation of OpenCL code running on Mali Midgard devices. It shows which command is being run on each thread over the course of the capture session and provides mechanisms to explore command dependencies.

Note

- **OpenCL** mode is an early access feature available to Mali licensees only. Contact your support team for more information.
- **OpenCL** mode is supported by gator version 21 and later and by Mali Midgard DDK version r6p0.

The Open Computing Language, or OpenCL, is a framework for parallel execution of jobs or kernels using task-based and data-based parallelism.

To enable **OpenCL** mode, you must create an instrumentation configuration file. For details of the options it must contain, see the documentation that is supplied with the ARM Driver Development Kit (DDK) for Mali Midgard devices.

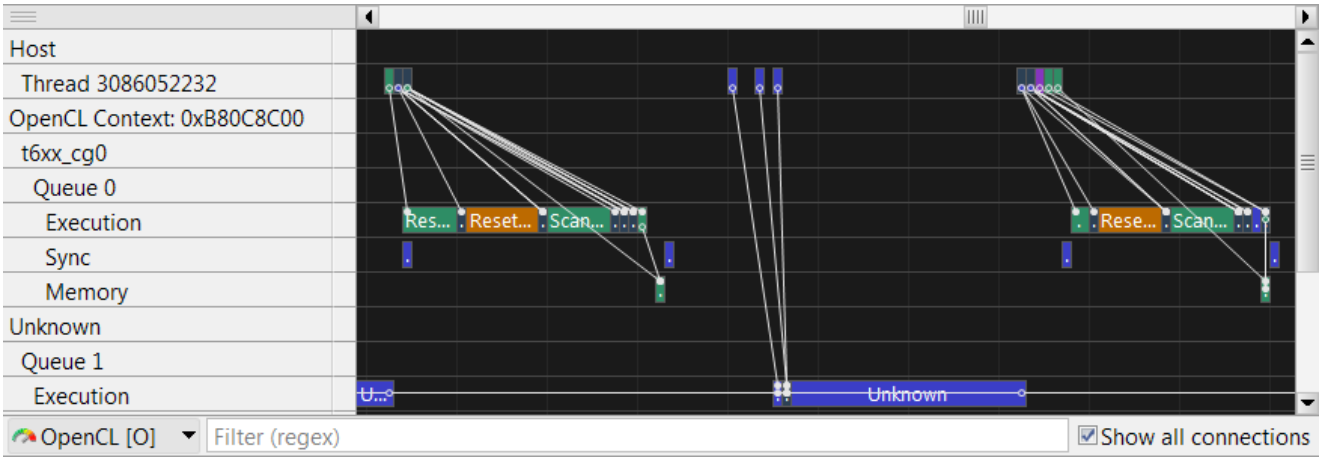


Figure 6-33 OpenCL mode

In OpenCL, commands are added to queues, then execute in parallel on one of the available hardware devices, usually a CPU, GPGPU, or DSP. Commands in a queue execute in series, but commands can also depend on the completion of commands in other queues.

Command names are shown inside colored areas, representing the duration of their execution. If there are two or more commands in a bin, and there is enough room, the number of commands in the bin is shown in light gray.

Dependencies between commands are shown using connecting lines. These have circles at each end to indicate the direction of the dependency. A command that is shown with a closed circle depends on a command that is shown with an open circle. Hover over the line connecting two commands to see a tooltip that shows the time delta between them.

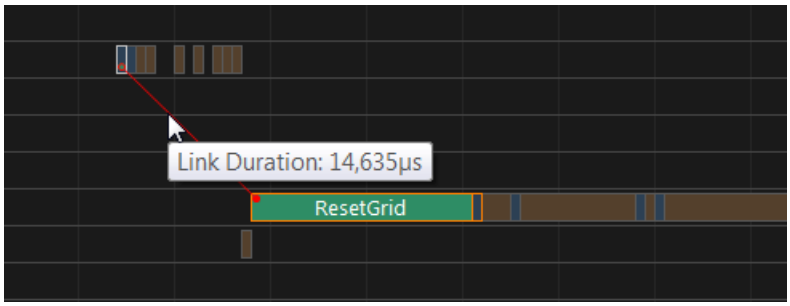


Figure 6-34 Time delta between connected commands

Streamline allows you to click a command to give it focus, hide any non-relevant information, and show its dependency connections. The following figure shows a selected command that is highlighted with a yellow border. A yellow line shows where it entered the queue.

Figure 6-35 Selecting a command in OpenCL mode

Zoom out to see more commands and the points at which they were enqueued. Click the **Show all connections** option to display connections for all commands.

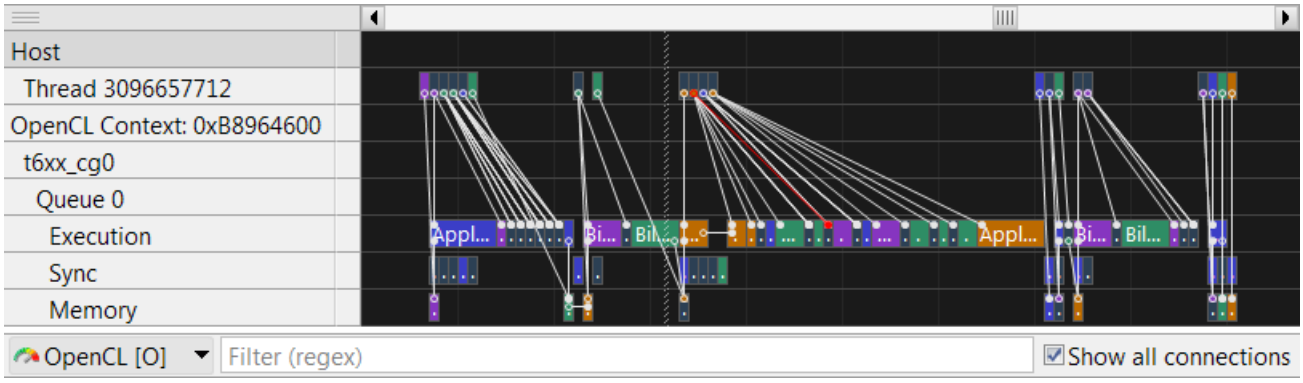


Figure 6-36 Zooming out in OpenCL mode

Hover over a command to display a tooltip that shows the command name, the time that it was initiated, and its duration.

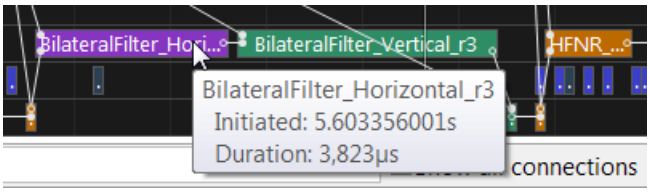


Figure 6-37 OpenCL mode tooltip

At the bottom of the chart in **OpenCL** mode is a filter field. Enter a regex in the field and **OpenCL** mode updates to show only matching commands.

Figure 6-38 Filtering in OpenCL mode

Related tasks

2.14 Setting up Streamline to support an ARM® Mali™-based device on page 2-41.

6.7 Mali Graphics Debugger (MGD) Mode in Live view

Switch to MGD mode while profiling an application in Streamline to analyze spikes in graphics activity.

To be able to use MGD mode, install Mali Graphics Debugger (MGD) and set up the target with `mgddaemon`. Instructions for setting up the target for MGD are found in the Mali Graphics Debugger User Guide, which comes with your installation of MGD.

In **Live** view, click the **Analyze in MGD** icon in the toolbar to launch MGD from the directory address that is provided in the **Capture & Analysis Options** dialog.

Note

The functionality that is described here is provided for MGD version 4.0.0.

The following steps then take place:

1. Streamline locks the **Live** view.
2. MGD launches or, if it is already open, starts a new trace in the open application, ensuring that only a single instance of MGD is running at a time.
3. The MGD application connects to `mgddaemon` on the target device, using a TCP/IP connection, and begins retrieving trace details from all graphics API applications running on the target.
4. Processes are then displayed in MGD charts. If there are no graphics API applications running on the target, MGD displays an empty window on entering MGD mode.

Streamline continues to profile and collect data while the **Live** view is locked. To continue displaying this data in **Live** view in Streamline alongside MGD, select the **Continue live capture after MGD is activated** option in the **Capture & Analysis Options** dialog. If this option is selected, a bookmark annotation appears in **Live** view to indicate when the first function is traced by MGD.

Streamline can detect the version number of `mgddaemon` through the TCP/IP connection. The version number allows Streamline to determine whether the Midstream Trace feature is available or whether a capture must be started before the program to be profiled is run.

To continue using Streamline as normal, close the MGD application.

MGD mode has the following limitations:

- Streamline cannot guarantee that `mgddaemon` is in the required state on the target device for MGD mode to be entered.
- Streamline is not aware if there have been or currently are any graphics API calls that will show a meaningful picture in MGD when it is launched.
- There is some visible distortion of performance in **Live** view as some Streamline performance indicators drop temporarily while `mgddaemon` pauses some of the processes that use graphics APIs.
- Running `mgddaemon` has minimal performance impact, however when MGD launches and begins collecting data from `mgddaemon` there is a significant drop in overall system performance.
- Continuous running of the target device with two agents can lead to high memory consumption on the host side after a while.

For further information about the Mali Graphics Debugger, see the Mali Graphics Debugger v4.0.0 User Guide, or the Mali Developer site, *http://www.malideveloper.com*.

Related references

4.1 Capture & Analysis Options dialog box settings on page 4-61.
6.8 Toolbar options in the Live and Timeline views on page 6-107.
15.3 Troubleshooting MGD Mode issues on page 15-211.

6.8 Toolbar options in the Live and Timeline views

ARM Streamline provides easy ways to navigate and modify the **Live** and **Timeline** views using the toolbar.

The toolbar controls in the **Live** and **Timeline** views are:

Tags

If a special condition applies to the capture, one or more of the following tags appear on the left side of the toolbar:


The capture uses event-based sampling.


The capture uses user space gator.


One or more warnings has occurred, as indicated by the number.


You are using DS-5 Community Edition.

When you hover over a tag, a tooltip appears, giving more information about what the tag means.

Stop capture and analyze - *Live view only*


Stops the capture session and creates an APC report. The report is displayed in the **Timeline** view. Clicking this button has the same effect as clicking the **Stop data capture from the target** button in the **Streamline Data** view.

Stop capture and discard - *Live view only*


Terminates the current capture session and discards all data. If you choose this option, Streamline does not create an APC report.

Toggle bookmark markers


Activates or deactivates the bookmark markers in the view. If selected, Streamline places vertical lines that stretch across the charts under each of your bookmarks.

Center the display on the Cross Section Marker


Centers the display on the position of the **Cross Section Marker**. This control is inactive if the **Cross Section Marker** is parked.

Reset Calipers


Resets the calipers. Calipers are the blue arrow controls at the top of the view that you can use to limit the statistics in the reports to an area of interest.

Select Annotation log... - *Timeline view only*


Opens the **Log** view and selects the closest **Annotation**-generated message to the current position of the **Cross Section Marker**. This option is not applicable if you did not include `ANNOTATION` statements in your code.

Zoom level - *Timeline view only*

Cycles through the levels of zoom. To increase or decrease the unit of time that is used to represent one bin in the **Timeline** view, use the plus and minus buttons. If you zoom in beyond the sampling frequency, the **Timeline** view shows interpolated data. Alternatively, use the drop-down list to specify the zoom level.

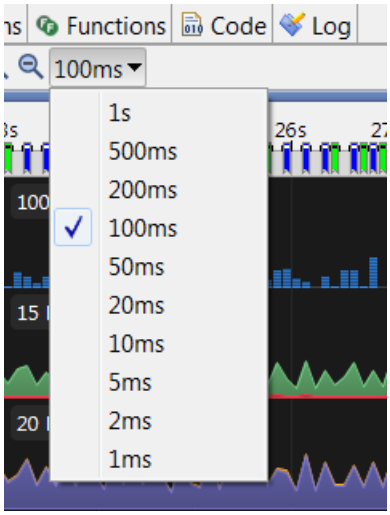


Figure 6-39 Zoom level drop-down list

Time index marker

Shows the time index of the current location of the mouse cursor.

Capture duration

The capture duration in seconds.

Go to beginning - *Live view only*

Moves the scrollbar to the beginning of the charts in the **Live** view and holds focus there.

Halt scrolling - *Live view only*

Causes the **Live** view to stop scrolling so that you can focus on the chart data currently in view. The charts of the **Live** view continue to expand as the capture session continues, but the **Live** view only moves if you move the horizontal scrollbar.

Go to live position - *Live view only*

Clicking **Go to live position** returns the **Live** view to the default scrolling behavior. The view scrolls with the growing charts and the live streaming data remains in focus.

Analyze in MGD - *Live view only*

Launches the Mali Graphics Debugger from the address that is specified in the **Capture & Analysis Options** dialog box, and locks the **Live** view in Streamline. If MGD is already running in another window, Streamline connects to MGD and a new trace begins. If the option to **Continue live capture after MGD is activated** has been selected in the **Capture & Analysis Options**, Streamline continues to display the live capture alongside tracing in MGD. If live capture is continued, a bookmark annotation appears on the **Live** view to indicate when the first function is traced by MGD. MGD retrieves trace details from all graphics API applications running on the target device from the time MGD mode started.

Switch and manage templates

Enables you to select a chart configuration template to apply to the current display and to manage saved templates and to create and load templates.

To append the charts in the selected template to the currently displayed charts, rather than replacing them, hold down the **Shift** key while clicking this button.

Add charts with default settings...

Enables you to add a chart to the **Live** or **Timeline** view from the drop-down list. Either select one of the charts with default settings, or add an empty chart and configure it yourself. The contents of the drop-down list depends on the data that was collected during the capture session.

Cycle through themes

Switches between the dark and light color schemes.

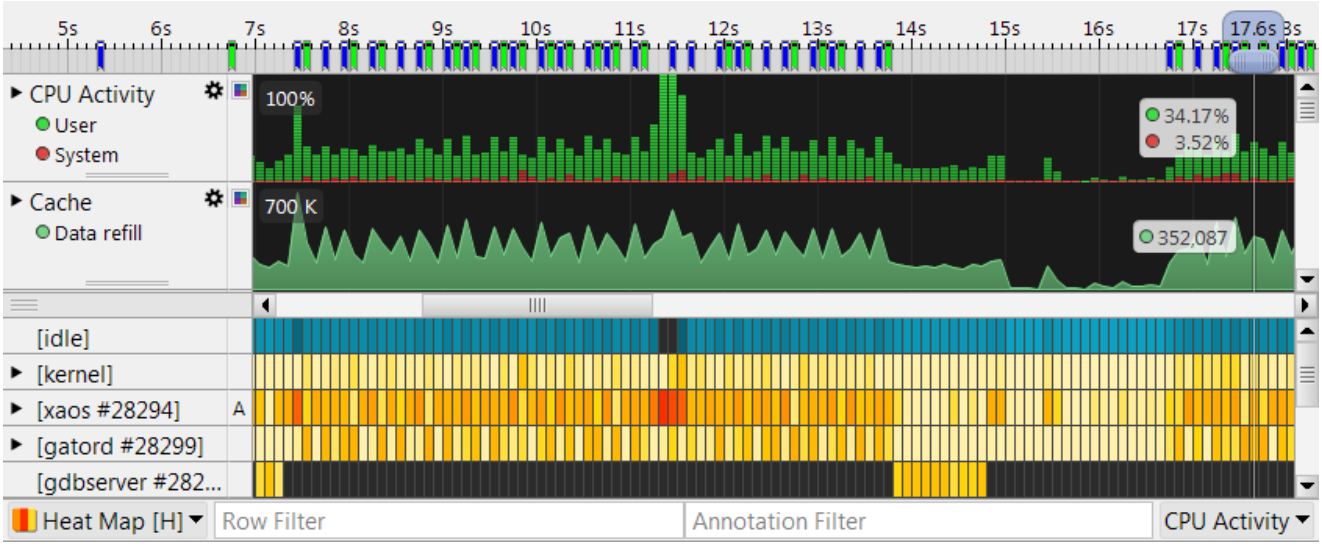


Figure 6-40 Dark color scheme

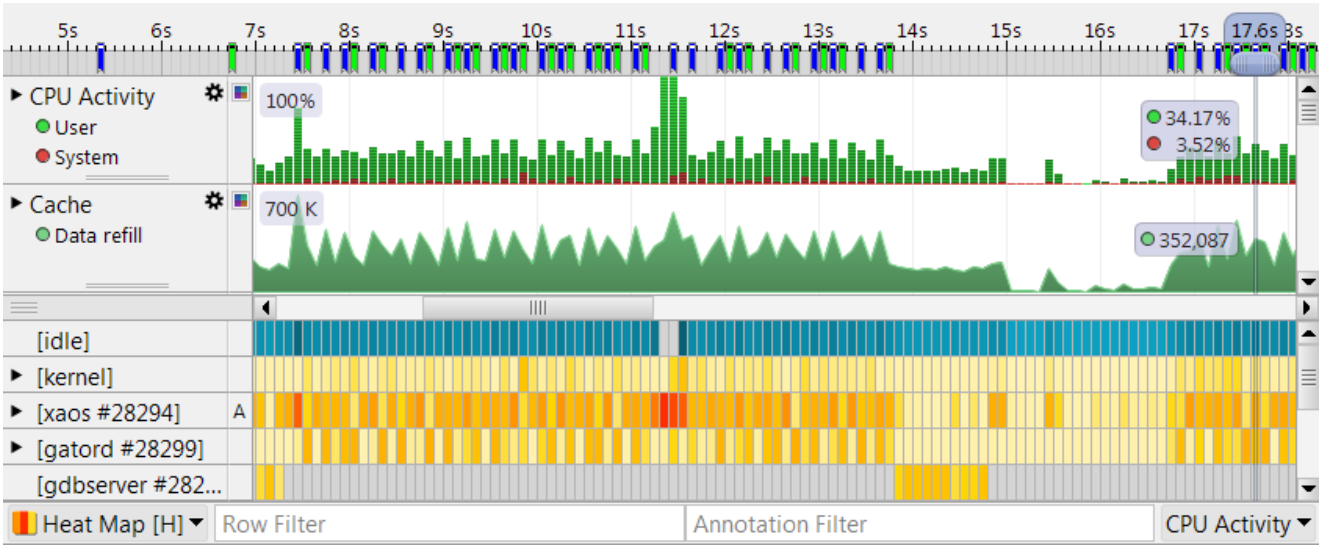


Figure 6-41 Light color scheme

Export - *Timeline view only*

Opens the **Export** dialog box, enabling you to export the data for the current zoom level from the **Timeline** view to a text file. You can choose to separate values with spaces, commas, or tab delimiters, making it easy to save data as a separate file or to open it in a spreadsheet application.

Help

Displays contextual help.

Related concepts

6.2 Timeline view overview on page 6-80.
6.4.4 Chart configuration templates on page 6-92.
*6.7 Mali Graphics Debugger (MGD) Mode in **Live** view* on page 6-106.
9.1 Annotate overview on page 9-135.

Related tasks

6.3.8 Filtering using the caliper controls on page 6-85.

Related references

6.9 Contextual menu options in the Live and Timeline views on page 6-111.
6.10 Keyboard shortcuts in the Live and Timeline views on page 6-113.

6.9 Contextual menu options in the Live and Timeline views

Right-click anywhere in the charts or in the details panel of the **Timeline** view when in one of the map modes to open a contextual menu that enables you to add bookmarks and to control the calipers and the cross section marker.

The contextual menu options are:

Create Bookmark at...
Creates a bookmark at the time index of the current position of the mouse cursor. The bookmark is displayed in the **Live** or **Timeline** view rule as a colored mark.

Bookmarks enable you to label and quickly return to points of interest in the view.

Bookmarks that are added in the **Live** view are preserved in the **Timeline** view and the **Log** view. Bookmarks are also preserved if you re-analyze a capture, or if you export then import a capture.

Hovering over a bookmark displays an overlay that contains the timestamp and text string of the bookmark. To change the text or the color selector, click them.

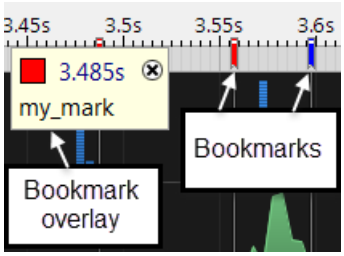


Figure 6-42 Bookmarks

Set Left Caliper
Sets the left caliper control to the current position of the mouse cursor. Streamline filters out all data before the left caliper from all views.

Set Right Caliper
Sets the right caliper control to the current position of the mouse cursor. Streamline filters out all data after the right caliper location from all views.

Reset Calipers
Resets the calipers to their default locations. The left caliper returns to the start of the capture session and the right caliper to the end.

Set Cross Section Marker To...
Moves the **Cross Section Marker** to the specified time index, by default the current position of the mouse cursor.

Set Cross Section Marker Start
Sets the left-hand boundary of the **Cross Section Marker**. If it is parked, the **Cross Section Marker** is moved to this location. This option is not available if the selected location is to the right of the **Cross Section Marker** end.

Set Cross Section Marker End
Sets the right-hand boundary of the **Cross Section Marker**. If it is parked, the **Cross Section Marker** is moved to this location. This option is not available if the selected location is to the left of the **Cross Section Marker** start.

Park the Cross Section Marker
Resets the **Cross Section Marker** to its default, inactive position.

Related concepts

6.3.7 Cross Section Marker on page 6-84.
6.6 Details panel in the Timeline view on page 6-97.

Related tasks

6.3.8 Filtering using the caliper controls on page 6-85.

Related references

6.8 Toolbar options in the Live and Timeline views on page 6-107.
6.10 Keyboard shortcuts in the Live and Timeline views on page 6-113.

6.10 Keyboard shortcuts in the Live and Timeline views

While you can navigate every report in Streamline using the mouse, keyboard shortcuts are often a faster way to accomplish common tasks.

The keyboard shortcuts available for both the **Live** and **Timeline** views are:

Left arrow
: Moves the Cross Section Marker one bin to the left.

Right arrow
: Moves the Cross Section Marker one bin to the right.

Shift+left arrow
: Contracts the width of the Cross Section Marker if it is wider than one bin.

Shift+right arrow
: Expands the width of the Cross Section Marker.

B
: Toggles the vertical markers for bookmarks on and off.

The following shortcuts apply to the **Timeline** view only:

L
: Opens the Log view and highlights any annotations covered by the Cross Section Marker.

+
: Zooms the **Timeline** view in one level.

-
: Zooms the **Timeline** view out one level.

Additionally, various shortcut keys can be used to switch between the available **Timeline** view modes. These keys are displayed in square brackets after the mode name in the mode menu, for example, **Processes [P]**.

Related concepts

6.6.1 Details panel modes on page 6-97.
6.3.7 Cross Section Marker on page 6-84.

Related references

6.9 Contextual menu options in the Live and Timeline views on page 6-111.
Chapter 10 Log View on page 10-146.

6.11 Warnings tag

The Warnings tag appears in the **Live** and **Timeline** views if there is a problem with the reports. Clicking on the tag displays a list of error messages, alerting you to the type and severity of problems.

The tag indicates the number of warnings that occurred. The tag is not displayed if there are no issues.

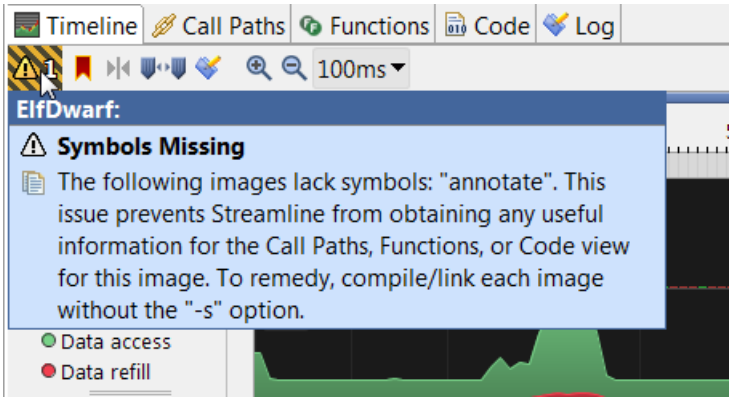


Figure 6-43 Warnings tag

The warnings are sorted by type and each is given a color-coded warning icon that tells you the severity of the issue:

Red
The issue is critical. For example, a connection timeout caused the termination of the capture, leaving you with incomplete data.

Yellow
The issue is medium severity. For example, Streamline was unable to successfully correlate power data from the Energy Probe.

White
The issue is minor. For example, a chart exists in the report but there is no counter data from the capture session to populate it.

6.12 Counter classes

Every counter belongs to a counter class. The class determines which filters are available for the counter.

Streamline supports the following basic classes of counters:

Absolute
: Absolute counters, for example **Memory: Free**, report the current, absolute value. Use the `average`, `maximum`, and `minimum` filters with absolute counters.

Delta
: Delta counters, for example **Clock: Cycles**, report the number of occurrences since the last measurement. The exact time when the data occurs is unknown, so data is interpolated between timestamps. Use the `accumulate` and `hertz` filters with delta counters.

Incident
: Incident counters, for example **Kmem: kmalloc**, are the same as delta counters, except the exact time is known when the data occurs, so no interpolation is calculated.

Activity
: Activity counters, for example **Contention: Wait**, report changes in processor activity or state. Use the `average` filter with activity counters.

For counters other than Activity counters, the data is calculated for 1ms resolution even in a high-resolution report. The high resolution zoom levels for these counters show interpolated values based on the 1ms data.

The following figure illustrates how the same data received from gator appears differently, depending on the counter class. In each case, the value 10 occurs at the 4.999ms timestamp, and the value 6 occurs at 6.999ms. The red lines shows the counter value at 1ms time intervals.

Note

In the delta counter chart, the value of 10 at 4.999ms is amortized from 5ms back to 0ms, because there is no other value, so its value is 2 for that period. The value of 6 at 6.999ms is amortized from 7ms back to 5ms, which is when the last value was received, so its value is 3 for that period.

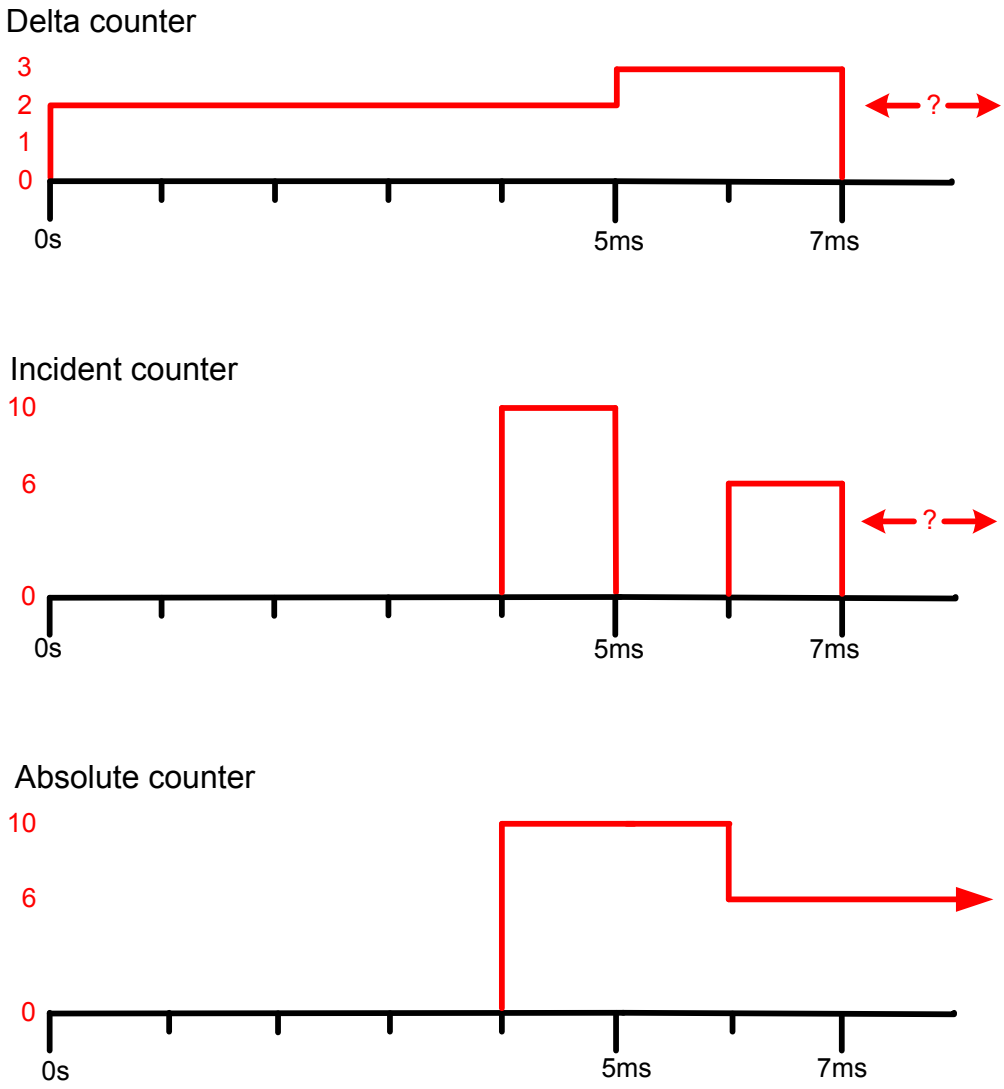


Figure 6-44 Counter classes

Related references

6.4.3 Chart configuration series options on page 6-90.

6.13 Visual Annotation in the Timeline view

If you use the Visual Annotate feature to add images to the capture data, those images appear as a chart in the **Timeline** view so that you can track the visuals of your application with the chart data.

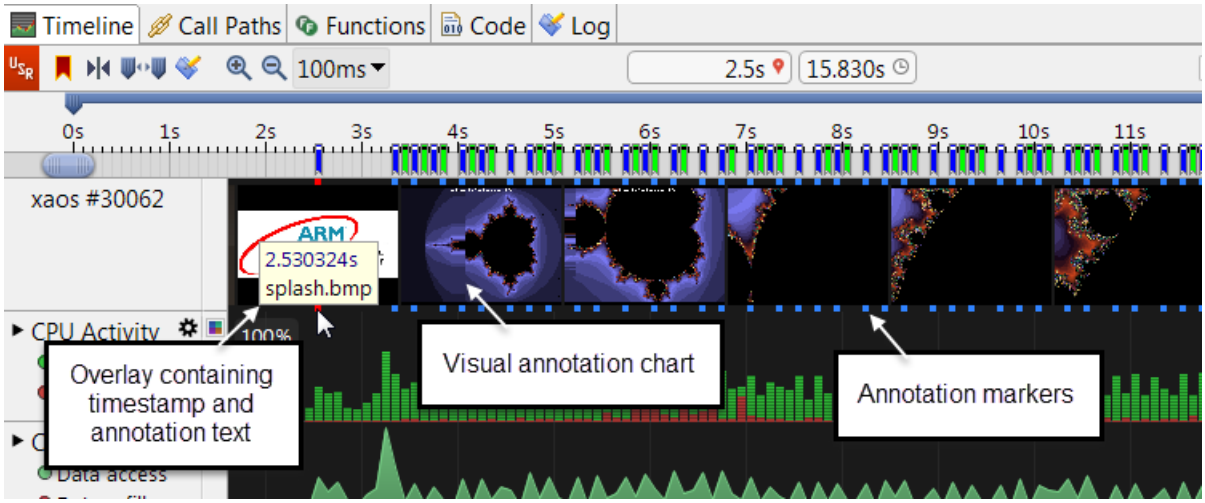


Figure 6-45 Visual annotation in the Timeline view

Visual annotation charts can be re-ordered in the same way as the other charts, but they have a few unique properties:

- If there is more than one image in the time period covered by a thumbnail in the chart, Streamline displays the first image from the range. Hovering your mouse over the thumbnail displays the image at the current mouse position. Moving the mouse over the thumbnail reveals the other annotated images in that range.
- Yellow or blue markers at the top and bottom of the chart identify the time bins in which your code produced each image. Yellow markers indicate that the annotation contains text in addition to the image. Blue markers indicate that there is no additional text. Hover over an image in the **Timeline** view and the markers at the cursor position turn red to identify the time bin of the image currently being displayed.
- Left-click while hovering over an image and the Details panel switches to **Images** mode to show an enlarged version of the image. Left click and drag the mouse left or right to view an animation of the images in the Details panel.
- Hover over a red marker to display an overlay that contains the timestamp and, if present, the text annotation associated with the image.

Related concepts

9.3 Visual Annotate overview on page 9-138.

Related references

9.6 Annotate macros on page 9-142.

Chapter 7
Table Views: Call Paths and Functions

Describes the **Call Paths** and **Functions** views, which provide tabular data about the capture.

It contains the following sections:

7.1 Toolbar options in the table views

The toolbar in the table views provides options for help, export, and editing source files.

The following toolbar options are available:

Fully expand all rows - ***Call Paths*** *view only*

Shows the entire hierarchy. It has the effect of opening disclosure controls to reveal all processes, threads, and functions.

Fully collapse all rows - ***Call Paths*** *view only*

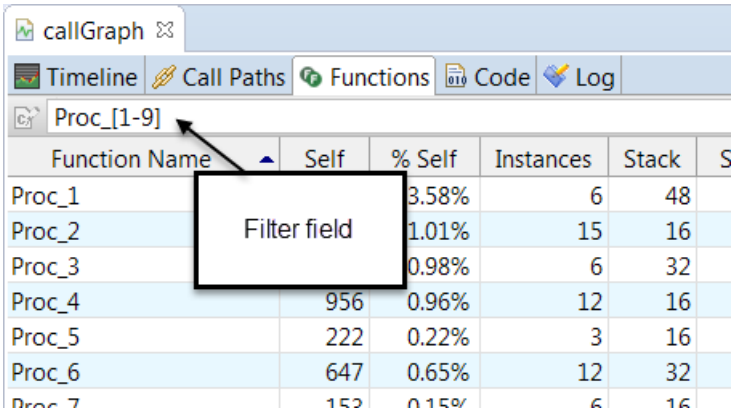
Hides all children in the entire hierarchy. It has the effect of closing all disclosure controls.

Edit Source

Opens the source file for the selected row in your default code editor. This button is enabled if source is available for the image in which the selected function is located, and the image contains debug symbols.

Filter

Enter a regular expression in this field to filter the rows displayed in the view. Only processes, threads, and functions whose name matches the regular expression are displayed.



Export table to a text file

Opens the export dialog box, enabling you to export the data from the table view. Values can be separated by spaces, commas, or tab delimiters, making it easy to save data as a separate file or to open it in your favorite spreadsheet application.

Show Help

Opens contextual help.

7.2 Call Paths view contextual menu options

Right-click on any row in the table report in the **Call Paths** view to open a contextual menu.

The following menu options are available:

Fully Expand Rows
: Expands the call path hierarchy for the selected functions.

Fully Collapse Rows
: Collapses the call path hierarchy for the selected functions.

Expand Selection to All Matching Functions
: Updates the current selection to include every instance of the selected functions, wherever they exist in the hierarchy. It expands the hierarchy to expose currently hidden instances of the function.

Expand Unselected Rows
: Expands every row in the view that is not part of the current selection. This is only available if there are unselected rows that are collapsed.

Collapse Unselected Rows
: Collapses every row in the view that is not part of the current selection. This is only available if there are unselected rows that are expanded.

Select Process/Thread in Timeline
: Opens the **Timeline** view with the currently selected processes, threads, and functions selected in the Processes section. Handles for the selected processes appear highlighted.

Select in Functions
: Opens the **Functions** view. All functions related to the selection in the current report are selected in the **Functions** view.

Select in Code
: Opens the **Code** view. All lines of code for the current selection are selected in the **Code** view.

Edit Source
: Opens the source file for the selected function in your default code editor.

7.3 Functions view contextual menu options

Right-click on any row in the table report in the **Functions** view to open a contextual menu.

The following menu options are available:

Top Call Paths
: Lists the processes and threads that the selected function was called from. The list is ordered by the number of samples collected in each call path instance. Only the top ten instances are listed.

Select Process/Thread in Timeline
: Opens the **Timeline** view with the processes and threads for the currently selected functions selected in the Processes section. Handles for selected processes appear highlighted.

Select in Call Paths
: Opens the **Call Paths** view. All instances related to the selection are selected in the **Call Paths** view. This option is only displayed if the selected functions have instances in the **Call Paths** view.

Select in Code
: Opens the **Code** view. All lines of code for the current selection are selected in the **Code** view.

Edit Source
: Opens the source file for the selected function in your default code editor.

7.4 Keyboard shortcuts in the table views

While you can navigate every table report in ARM Streamline using the mouse, you can use keyboard shortcuts to perform common tasks quickly.

The following keyboard shortcuts are available for the table views:

Up arrow
: Moves the current selection up one row.

Shift + Up Arrow
: Adds the previous row to the current selection.

Down arrow
: Moves the current selection down one row.

Shift + Down Arrow
: Adds the next row to the current selection.

Home
: Selects the first row in the active table report.

End
: Selects the last row in the active table report.

Page Up
: Moves up in the current report one page. A page is defined by the range of rows currently displayed in the table report.

Page Down
: Moves down one page.

Right Arrow - ***Call Paths*** *view only*
: Discloses the subordinate rows for the currently selected process, thread, or function. Has the same effect as clicking on the disclosure control to the left of the process, thread, or function's title.

Left Arrow - ***Call Paths*** *view only*
: Hides the subordinate rows for the currently selected process, thread, or functions.

Shift + Right Arrow - ***Call Paths*** *view only*
: Discloses all of the subordinate rows for the currently selected process, thread, or functions. The entire hierarchy below the selected links is revealed.

Shift + Left Arrow - ***Call Paths*** *view only*
: Hides all of the subordinate call chain links. On the surface, it has the same effect as pressing the left arrow by itself, but when the subordinate process, thread, and functions are again revealed, this command ensures that only their immediate subordinates appear.

7.5 Sorting table reports

Streamline supports multi-level search in all of the table reports, enabling you to bring functions to the top of table reports based on your specifications.

To perform a multi-level sort in any table report, follow these steps:

Procedure

1. Click on any of the column headers.
   The data in the table views is reordered based on the data contained in that column. Repeat clicks toggle the sort order between ascending and descending. The default numerical and alphabetical sorting behavior varies from column to column, but an upwards arrow in the column header always indicates an ascending sort, while a downward arrow indicates a descending sort.

   Note

   If an element is selected in the table, a re-sort attempts to keep the selected element in view.

2. Hold down the shift key and click on a different column header.
   This provides a secondary sort. Clicking on the same column again with the shift key held down reverses the sort order. This can be repeated on any number of columns to provide multiple levels of sorting.

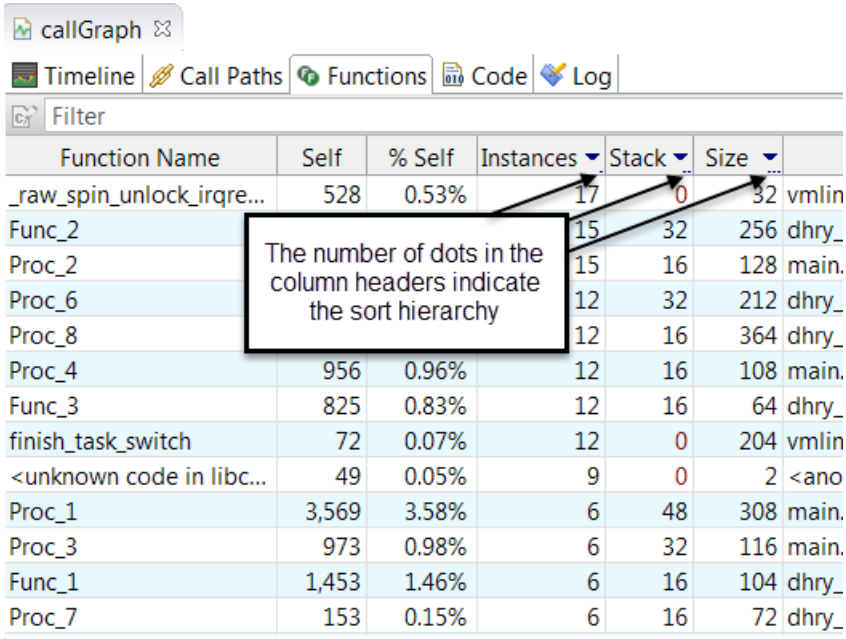


Figure 7-1 Multi-level sort

Note

The dots in the lower right of the column headers indicate ordering, with the number of dots indicating the position of the column in the sort hierarchy. For example, one dot means the marked column is the primary sort column, while two dots indicates it is the secondary sort.

7.6 Call Paths view column headers

The **Call Paths** view presents its data hierarchically, showing called functions and threads as subordinate in the hierarchy to their calling function, thread, or process. Every function, thread, or process that called another has a disclosure control that you can use to see more of the hierarchy.

It has the following column headers:

Process/Thread/Code
The name of the process, thread, or function. Every [process] appears in the list enclosed in square brackets, while {threads} are enclosed in braces.

Note

If call stack unwinding is not available, for example because it was disabled in the **Capture & Analysis Options** dialog box, the sampled functions all appear directly under the threads in the **Call Paths** view.

Total
The **Process** figure as a percentage of the total number of samples collected in all processes.

Self
The number of samples collected in this function, within this call path only. This figure excludes samples collected in functions called by this function.

% Self
The **Self** figure as a percentage of the total number of samples collected within the process.

Process
The number of samples collected in this process, thread, or function. For a function, this figure includes samples collected in any functions it calls.

% Process
The **Process** figure as a percentage of the total number of samples collected within the process.

Stack
The number of bytes used by the stack at this point in the call path. The value is a dark red color if the stack usage could not be determined for one or more functions in the call path.

Location
The location of the function, listing both the file name and line number of the declaration.

Note

All data in the **Call Paths** view is dependent on the text entered in the filter field in the toolbar, as well as the filtering selection in the **Timeline** view. If you have used the caliper controls to filter data in the **Timeline** view, the data in the **Call Paths** view reflects this selection.

Related tasks

6.3.8 Filtering using the caliper controls on page 6-85.

Related references

15.5 Troubleshooting report issues on page 15-214.
4.1 Capture & Analysis Options dialog box settings on page 4-61.

7.7 Functions view column headers

The **Functions** view provides a list of all functions called during the capture session alongside usage data.

It has the following column headers:

Function Name
The name of the function.

Self
The number of samples collected in all instances of the function. This figure excludes samples collected in any functions called by this function.

% Self
The **Self** figure as a percentage of the total number of samples collected in all functions.

Instances
The number of times the function appears in the **Call Paths** view.

Stack
The number of bytes used by the stack in this function. If the stack usage cannot be precisely determined, the value is colored dark red.

Size
The size of the function in bytes.

Location
The location of the function, listing both the file name and line number of the declaration.

Image
The image file that contains the function.

Note

All data in the **Functions** view is dependent on the text entered in the filter field in the toolbar, as well as the filtering selection in the **Timeline** view. If you have used the caliper controls to filter data in the **Timeline** view, the data in the **Functions** view reflects this selection.

Related tasks

6.3.8 Filtering using the caliper controls on page 6-85.

Chapter 8
Code View

Describes the **Code** view, which provides statistics for lines of source code and for disassembled instructions.

It contains the following sections:

8.1 Code view basics

The **Code** view helps with the discovery of function-level hot spots. It flattens statistics and displays them at the source and disassembly levels.

By default, the **Code** view shows the source code next to color-coded statistics. To toggle between viewing the source code only and both source and disassembly, click the disassembly view button.

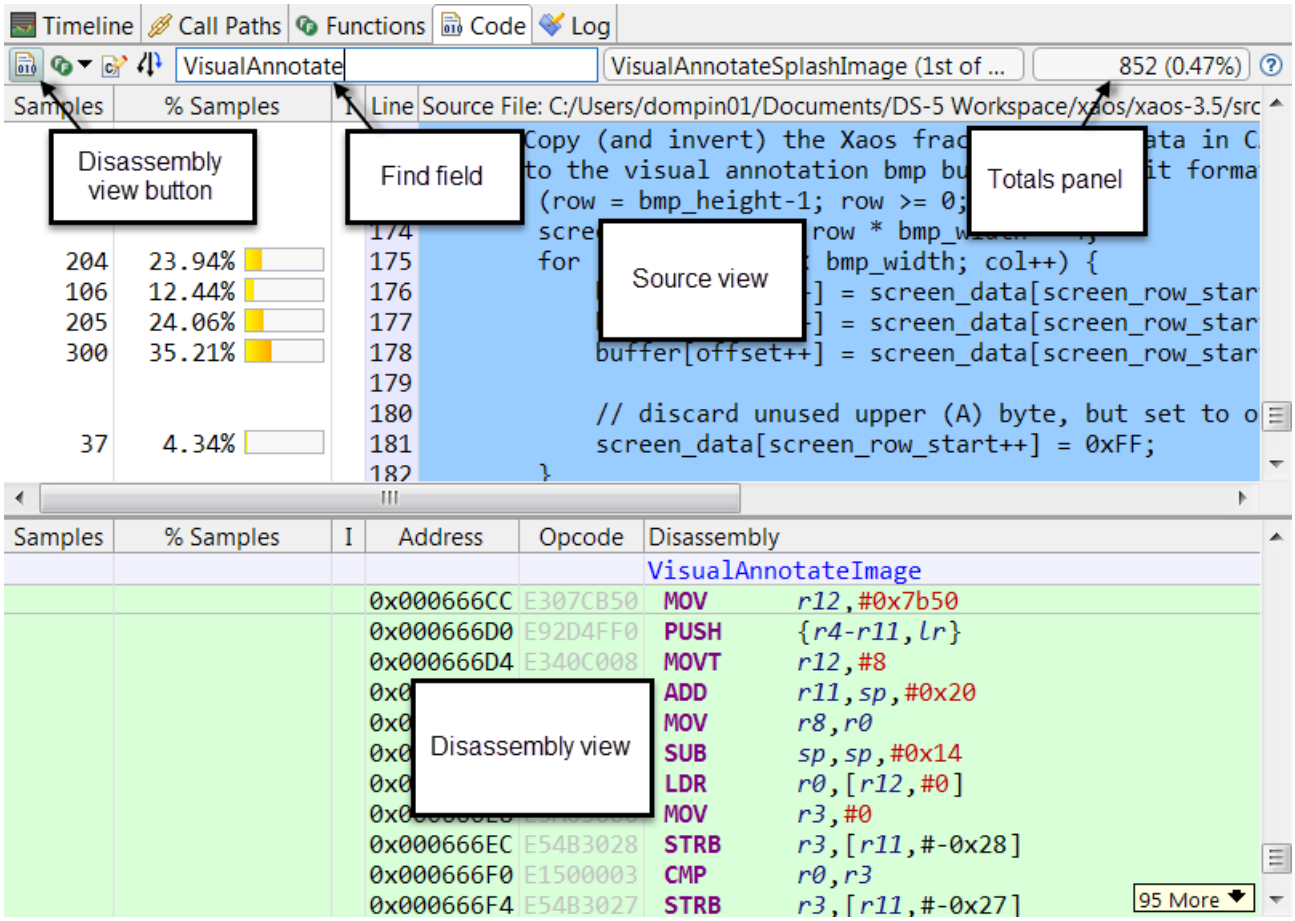


Figure 8-1 Code view

The totals panel displays the number of samples collected in the selected code, and this value as a percentage of the total number of samples collected.

Note

All of the sampling data in the **Code** view is dependent on the filtering selection in the **Timeline** view. If you have used the caliper controls to filter data in the **Timeline** view, the sampling data in the **Code** view reflects this selection.

Related tasks

6.3.8 Filtering using the caliper controls on page 6-85.

Related references

*8.5 **Code** view toolbar options* on page 8-132.

*8.6 **Code** view keyboard shortcuts* on page 8-133.

8.2 Selecting rows in the Code view

To select a single row of code in the **Code** view, simply click on it. If you want the selection to span a range of code lines, you have a few options.

To select multiple rows in the **Code** view, follow these steps:

Procedure

1. Click on the code row that is the start of your range.
2. There are multiple ways to define the range:
   - Hold the mouse after clicking on the first row and drag it across a range of rows.
   - Hold down the shift key and select the last row of the series to select the entire sequence of rows.
   - Hold down the control key if you want to select additional rows without selecting all of the rows in between.

   Selecting code in the **Code** view highlights related instructions in the disassembly panel.

   Note

   - The selection behavior in the disassembly section is slightly different than that of the source code section. Selecting a single line of disassembly selects all of the instructions that relate to a single line of source code, and you can use function tags to highlight all instruction lines associated with a single function.
   - If selected source code lines or disassembly instructions contain too many rows to fit in the bounds of the current window, small selection indicators appears on the right hand side of the **Code** view. If there are more selected rows than can fit in the view, the indicators show you how many more are present off screen. Click on the **More** indicator to see additional selected rows.

8.3 Path prefix substitution in the Code view

Streamline automatically locates and displays the source code in the **Code** view. If, however, the source files are not located in the same directory they were in during compilation, the **Code** view is not populated and you must set up path substitutions so that Streamline can find the code.

In cases where Streamline cannot locate the source, it displays a missing source file message in the source section of the **Code** view.

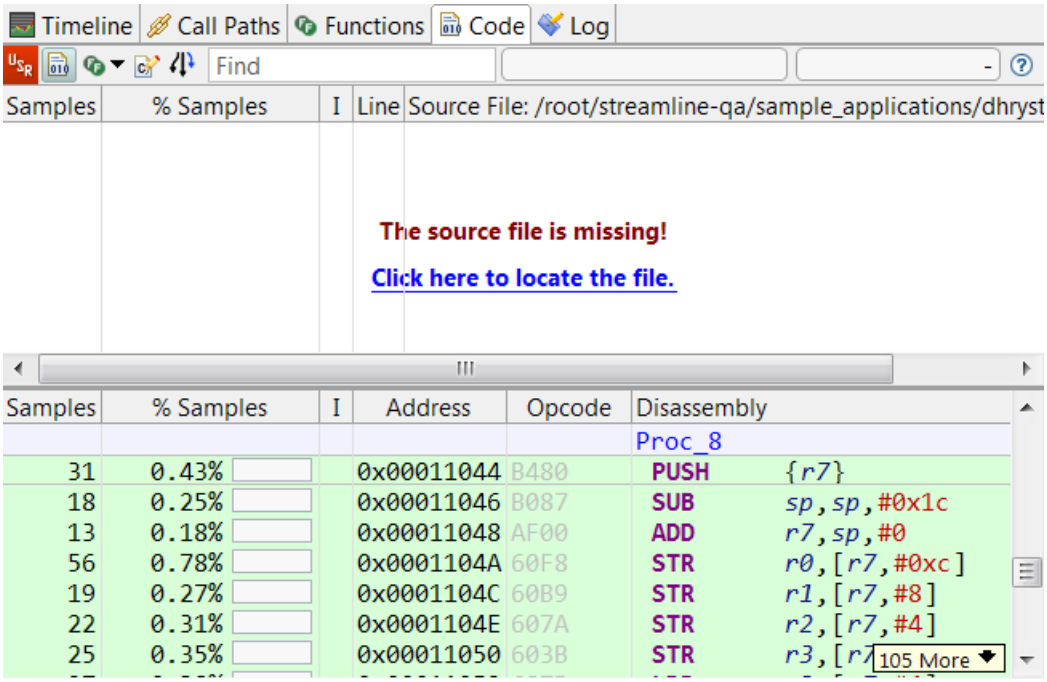


Figure 8-2 Missing source file

Follow these steps to set up path prefix substitutions:

Procedure

1. There are two ways to start:
   - Click the link under the missing file message. This displays a standard file dialog which allows you to locate the file. Doing this creates an entry in the **Path Prefix Substitutions** dialog to appropriately map the file.
   - Click the **Set Path Prefix Substitutions** button in the toolbar. This opens the **Path Prefix Substitutions** dialog box.

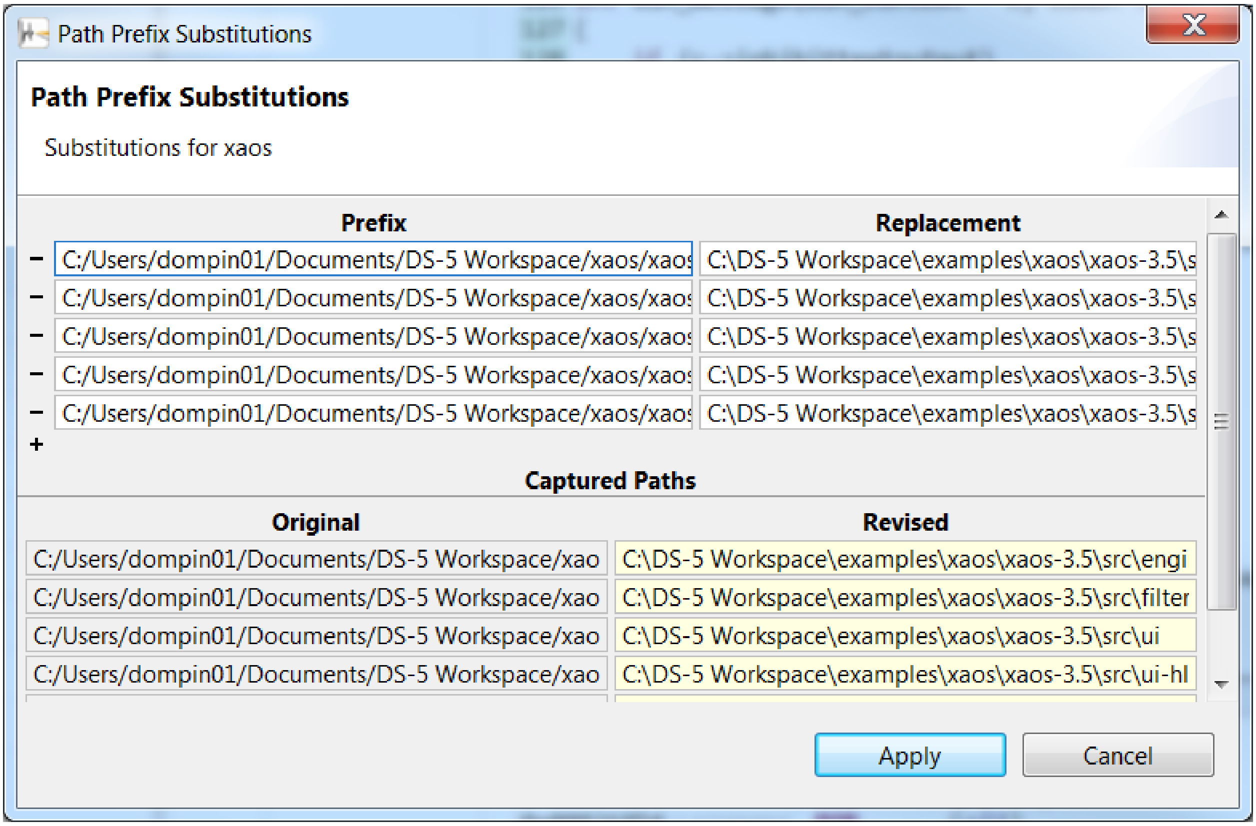


Figure 8-3 Path Prefix Substitutions dialog box

2. Click the plus symbol to add a new path prefix substitution.
3. Enter valid paths in the Prefix and Replacement fields.
4. Click the **Apply** button.
   If the path given in the Replacement field contains code that lines up with the code used in the capture, Streamline populates the view with your source code and the statistical overlay.

8.4 Using the Find field in the Code view

To search your code and instructions for a function name or an instruction address, use the Find field, located just below the toolbar in the **Code** view.

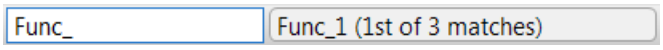


Figure 8-4 Find field

To find a specific function or hexadecimal instruction address, follow these steps:

Procedure

1. Enter a string in the Find field.
   The box to the right of the Find field updates to show any current matches.
2. Press the Enter key to go to the first match in the code.
3. Press the Enter key again to cycle through all available matches.

Related concepts

*8.1 **Code** view basics* on page 8-127.

8.5 Code view toolbar options

The toolbar of the **Code** view contains the following buttons:

Disassembly View

 Opens the disassembly panel. The disassembly panel takes up the bottom section of the **Code** view and shows the assembly language instructions associated with the source code.

Recently Selected Functions

 Opens a drop-down menu that enables you to choose a recently selected function from a list. This selects the function in the **Code** view.

Edit Source

 Opens the source file in your preferred editor.

Set Path Prefix Substitutions

 Opens the **Path Prefix Substitutions** dialog box. This enables you to set up path substitutions so that Streamline can find code that is not located in the same directory that it was in during compilation.

Help

 Opens contextual help.

Related concepts

*8.1 **Code** view basics* on page 8-127.

Related tasks

*8.3 Path prefix substitution in the **Code** view* on page 8-129.

8.6 Code view keyboard shortcuts

The keyboard shortcuts in the **Code** view help you to quickly navigate through your source code and disassembly instructions.

The keyboard shortcuts available for the table views are:

Up arrow
Moves the current selection up one row.

Shift + Up Arrow
Adds the previous row to the current selection.

Down arrow
Moves the current selection down one row.

Shift + Down Arrow
Adds the next row to the current selection.

Home
Takes you to the top of the function that contains the currently selected row. If a line of code is selected in the source view that does not have any instructions associated with it, the home key takes you top of the source file.

End
Takes you to the bottom of the function that contains the currently selected row. Like the home key, if the selected line of source does not have any instructions associated with it, the end key takes you to the bottom of the file.

Page Up
Moves up one page. A page is defined by the range of rows currently displayed in either the source or disassembly view.

Page Down
Moves down one page.

Related concepts

*8.1 **Code** view basics* on page 8-127.

Chapter 9
Streamline Annotate

Describes the Streamline Annotate feature. It enables you to add annotations to your code, which are propagated into the **Timeline** and **Log** views.

It contains the following sections:

9.1 Annotate overview

While ARM Streamline provides a large variety of target information, sometimes you might require extra context. Streamline lets you instrument your source code by adding annotations to it.

When the user space application writes to the gator annotate socket, the gator driver integrates the recorded annotate-driven output into the Streamline sample and trace capture report. The annotated text is marked with a thread identifier, which keeps the data uncluttered and eliminates the need for user mutexes.

You can add the following types of annotations:

———— **Note** ————

Build the examples in *DS-5_install_directory*/sw/streamline/examples/annotations/ as follows:

- Build natively on an ARM target by running `make`.
- Cross compile by running `make CROSS_COMPILE=`*`filename`*. The compiled binaries appear in a folder called `bin`.
- Build for Android by running `ndk-build`.

————————————

String

String annotations work in a similar way to `printf()` statements. However, instead of console output, they populate the **Log** view and place framing overlays directly in the **Timeline** view.

See *DS-5_install_directory*/sw/streamline/examples/annotations/text.c for examples of how to use some of the string annotation macros.

Visual

Visual annotations add images to the visual annotation chart in the **Timeline** view. The images are also displayed in the **Log** view.

See *DS-5_install_directory*/sw/streamline/examples/annotations/visual.c for an example of how to use the `ANNOTATE_VISUAL()` macro.

Marker

Marker annotations add bookmarks to the **Timeline** and **Log** views, optionally with a text string, to identify time points of interest.

See *DS-5_install_directory*/sw/streamline/examples/annotations/text.c for examples of how to use the various marker annotation macros.

Custom counters

These annotations are counters that you dynamically define in user space code. After you have run a capture session, the **Timeline** view displays a chart for each custom counter you have created, showing the numeric values plotted over time.

See *DS-5_install_directory*/sw/streamline/examples/annotations/absolute.c for examples of how to use custom absolute counters and *DS-5_install_directory*/sw/streamline/examples/annotations/delta.c for examples of how to use custom delta counters.

Groups and channels

This set of macros enables you to break down threads into multiple channels of activity and assign each channel to a group. A number identifies each group and channel, and each one has a name for display in the **Timeline** and **Log** views. Groups and channels are defined per thread. Therefore although each channel number must be unique within the thread, channels in different threads can have the same number.

See *DS-5_install_directory*/sw/streamline/examples/annotations/text.c for examples of how to use the group and channel annotation macros.

Custom activity maps

These annotations allow you to define and visualize a complex dependency chain of jobs. Each *custom activity map* (CAM) view contains one or more tracks and each track contains one or more jobs.

Jobs might have dependencies on other jobs. Dependencies between jobs are shown using connecting lines with circles at each end to indicate the direction of the dependency. A job with a closed circle depends on a job with an open circle.

See `DS-5_install_directory/sw/streamline/examples/annotations/cam.c` for examples of how to use the CAM annotation macros.

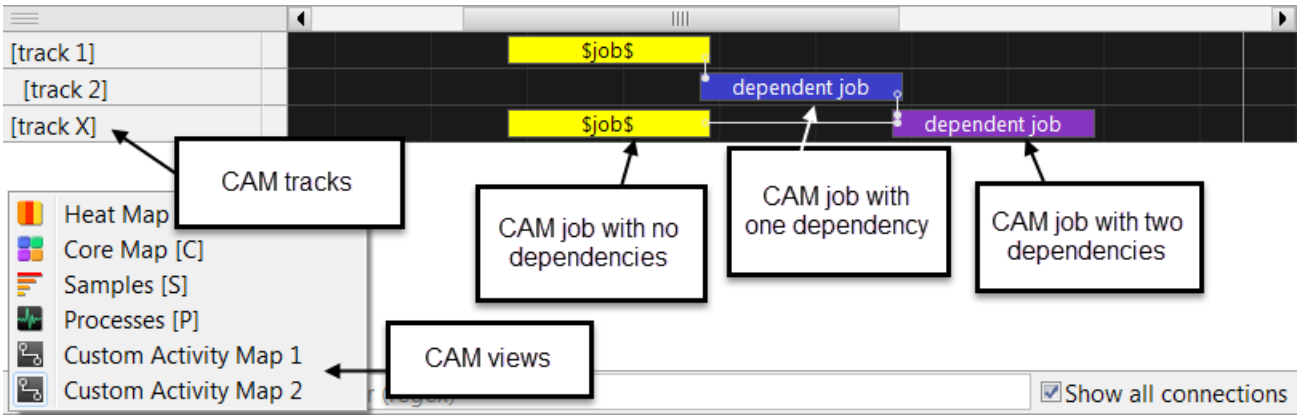


Figure 9-1 Custom Activity Maps

Annotation macros are defined in `streamline_annotate.h`. `StreamlineAnnotate.java` provides equivalent methods for string, visual, and marker annotations, and for groups and channels.

In addition to the basic reference examples for each annotation type in `DS-5_install_directory/sw/streamline/examples/annotations/`, some of the DS-5 Linux examples use annotations. `Streamline_annotate` uses string, visual, and marker annotations, and groups and channels, and `xaos` uses string, visual, and marker annotations. Both are located in `DS-5_install_directory/examples/Linux_examples.zip`. To use these examples, import the Linux application example projects into Eclipse for DS-5. Refer to the `readme.html` for each example for more details.

Note

In DS-5 version 5.20 and later, applications that use user space gator, in addition to ones that use kernel space gator, can emit annotations. Applications that are built using the annotation implementation in earlier versions of DS-5 continue to work in version 5.20 and later. However they only work with kernel space gator. If they use user space gator, you must rebuild them using the new annotation implementation.

Related tasks

9.2 Adding string annotations to your code on page 9-137.
9.7 Importing the Streamline_annotate example on page 9-145.

Related references

9.6 Annotate macros on page 9-142.

9.2 Adding string annotations to your code

String annotations work in a similar way to `printf()` statements but instead of console output, they populate the **Log** view and place framing overlays directly in the **Timeline** view.

To add string annotations to your code, follow these steps:

Procedure

1. Include `streamline_annotate.h` and `streamline_annotate.c` in your project.

   If you are working in Java, `StreamlineAnnotate.java` provides the same functionality as the macros in `streamline_annotate.h` for string, visual, and marker annotations, and for groups and channels.
2. Add the `ANNOTATE_SETUP` macro to your code. You must call this before any other annotate macros.
3. Add annotations to your code using the macros defined in `streamline_annotate.h`. For example, write a null-terminated string from any thread and set its color using the `ANNOTATE_COLOR(color, string)` macro. Either choose a color constant from those defined in `streamline_annotate.h` or send the ASCII escape code followed by a 3-byte RGB value.
4. Optional: Use `ANNOTATE_END()` to clear the annotation message for the thread.
5. Ensure `gatord` is running.
6. Run your application and exercise the area of the code that emits the annotations.

String annotations are displayed as text overlays inside the relevant channels in the details panel of the **Timeline** view, for example inside **Channel 0** in the following screenshot. The letter **A** is displayed in the process list to indicate the presence of annotations. String annotations are also displayed in the Message column in the **Log** view.

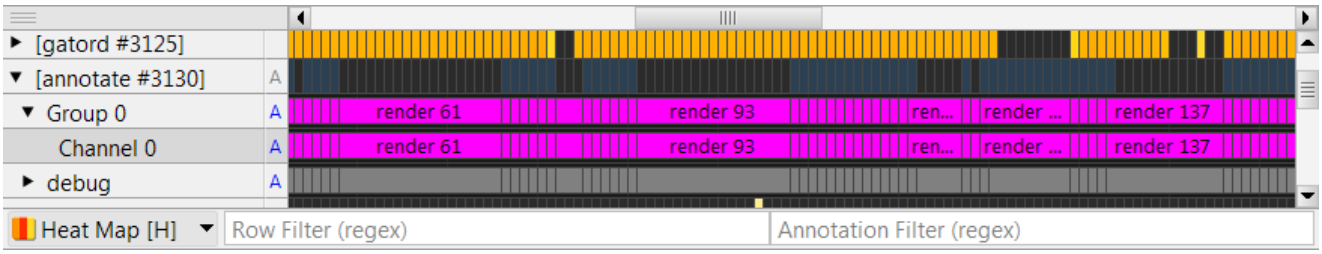


Figure 9-2 String annotation overlays

Related concepts

9.1 Annotate overview on page 9-135.

Related tasks

9.7 Importing the Streamline_annotate example on page 9-145.

Related references

9.6 Annotate macros on page 9-142.

9.3 Visual Annotate overview

In addition to simple text annotations, Streamline supports annotations that contain images, providing further application-level context to the **Timeline** view.

In the same way as string annotations, the application writes to the gator annotate socket and Streamline integrates the image data and its timestamp into the sample and trace capture report.

Visual annotations are displayed in the visual annotation chart, shown in the following screenshot:

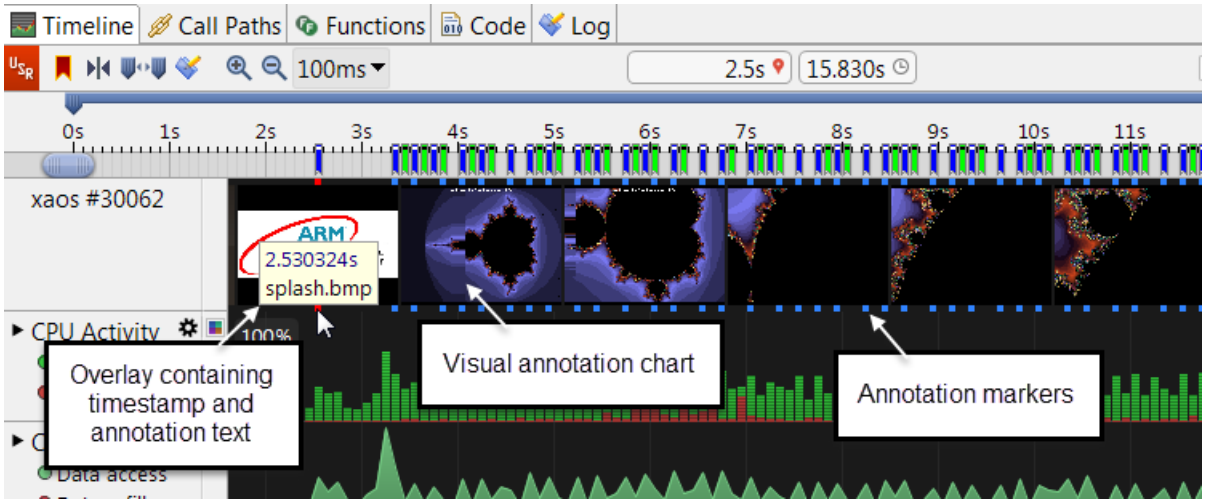


Figure 9-3 Visual Annotate in the Timeline view

To include images in the data sent to the host during a capture session, use the `ANNOTATE_VISUAL()` macro in your source code. `ANNOTATE_VISUAL()` provides a parameter for image data.

Visual Annotate supports images in the following formats:

- GIF.
- PNG.
- JPEG.
- TIFF.
- ICO.
- BMP +RLE.

There is no limit to the image size but the larger the image, the greater the impact on system performance. Increasing the amount of data sent to the host in this way increases the probe effect for the applications you are profiling.

The following example function is from the `Streamline_annotate` example project, included in the `Linux_examples` archive in the `examples` directory of your DS-5 installation:

```
void displayImage() {
    char filename[32];
    char* image;
    unsigned int size;

    // Supported formats include gif, png, jpeg, tiff, ico, bmp (+rle)
    strcpy(filename, "splash.bmp");
    image = readFromDisk(filename, &size);
    if (image == NULL) {
        printf("error loading image %s\n", filename);
        exit(1);
    }

    // Add text along with the image annotation
    ANNOTATE_VISUAL(image, size, filename);
    free(image);
}
```

You can see the effects of Visual Annotate in the **Timeline** and **Log** views of your Streamline Analysis Reports. The **Timeline** view includes a chart that displays the images. In the **Log** view, any annotation event that includes an image has a camera icon in the message field. Select a row containing a camera icon to see the image.

Note

A Mali GPU automatically emits visual annotations when you select a Filmstrip counter.

Related concepts

6.13 Visual Annotation in the Timeline view on page 6-117.

Related tasks

2.14 Setting up Streamline to support an ARM® Mali™-based device on page 2-41.

Related references

9.6 Annotate macros on page 9-142.

9.4 Adding Visual Annotate to your code

Streamline provides a set of macros that enable you to add images to the **Timeline** and **Log** views in your generated Analysis Reports.

To add visual annotations to your code, follow these steps:

Procedure

1. Include the files `streamline_annotate.h` and `streamline_annotate.c` in your project.
2. Add the `ANNOTATE_SETUP` macro to your code. It must be called before any other annotate macros are called.
3. Insert the `ANNOTATE_VISUAL(data, length, str)` macro into your code. Replace `data` with your image, `length` with the size of the data being written to the annotate file, and, optionally, `str` with a descriptive string to be included with the image.
4. Build your code.
5. Ensure that `gatord` is running then run your application and exercise the area of the code that emits the annotations.

9.5 Kernel annotations

You can insert annotation macros in either user space code or kernel space code. There are a few important considerations when using them in kernel space code or in a module.

Note

As the annotation macros might block, ARM recommends that you do not add them to kernel space code in an interrupt context. For more information, see `gator_annotate.c` in the gator driver source code.

For string, visual, and marker annotations, and channels and groups, the same macros are defined for both user space and kernel space using the `_KERNEL_` preprocessor conditional. Insert these macros in kernel space code or in a module in the same way you would in user space code, with the following exceptions:

- Do not call `ANNOTATE_SETUP` in kernel space code. This macro is a prerequisite for other annotation macros in user space code, but not in kernel space code.
- You do not need to include `streamline_annotate.c` in the project.

You can only use the custom counter macros and custom activity map macros in user space code. See the `gator_events_mmapped.c` example in the gator driver source code for an example of adding custom counters to kernel space code.

Related tasks

12.4 Using the gator_events_mmapped.c custom counters example on page 12-168.

Related references

9.6 Annotate macros on page 9-142.

9.6 Annotate macros

Streamline provides a variety of macros to mark up your code, whether you want color markup, macros that include channel information, or macros that insert bookmarks into the **Timeline** and **Log** views.

These macros are defined in *DS-5_install_directory*/sw/streamline/gator/annotate/streamline_annotate.h.

Macro	Description
ANNOTATE_DEFINE	You do not need to call ANNOTATE_DEFINE. It is defined in streamline_annotate.h to avoid compilation errors in legacy code.
ANNOTATE_SETUP	Call this macro to set up annotation, before calling any other annotate macros. **Note** When you annotate within the kernel or a module, do not use ANNOTATE_SETUP.
ANNOTATE(string)	Adds a string annotation. This macro does not define a specific channel, so the annotation is added to Channel 0 by default.
ANNOTATE_CHANNEL(channel, string)	Adds a string annotation to a channel defined by the numeric identifier passed in the channel parameter. **Note** Annotation channels and groups are used to organize annotations within the threads and processes section of the **Timeline** view. Each annotation channel appears in its own row under the thread. Channels can also be grouped and displayed under a group name, using the ANNOTATE_NAME_GROUP macro.
ANNOTATE_COLOR(color, string)	Works in the same way as the basic ANNOTATE macro, except the color parameter defines an interface display color for the annotation string. See streamline_annotate.h for the defined colors.
ANNOTATE_CHANNEL_COLOR(channel, color, string)	Defines an annotation string with a display color, and assigns it to a channel.
ANNOTATE_END()	Terminates the annotation. Because the ANNOTATE_END macro does not define a specific channel, it defaults to channel 0.
ANNOTATE_CHANNEL_END(channel)	Terminates the annotation for the given channel.
ANNOTATE_NAME_CHANNEL(channel, group, string)	Defines a channel and attaches it to an existing group. The channel number must be unique within the thread.
ANNOTATE_NAME_GROUP(group, string)	Defines an annotation group. The group identifier, group, must be unique within the thread.
ANNOTATE_VISUAL(data, length, str)	Records an annotation that includes an image in one of the following formats: • GIF. • PNG. • JPEG. • TIFF. • ICO. • BMP +RLE. Specify the image in data, the amount of data being written to the annotate file in length, and optionally a descriptive string to be included with the image in str.

(continued)

Macro	Description
`ANNOTATE_MARKER()`	Adds a default bookmark to the **Timeline** and **Log** views.
`ANNOTATE_MARKER_STR(string)`	Adds a bookmark with a string to the **Timeline** and **Log** views. The string is displayed in the **Timeline** view when you hover over the bookmark and in the **Message** column in the **Log** view.
`ANNOTATE_MARKER_COLOR(color)`	Adds a bookmark with a color to the **Timeline** and **Log** views.
`ANNOTATE_MARKER_COLOR_STR(color, string)`	Adds a bookmark with a string and a color. The bookmark appears in the **Timeline** and **Log** views.
`ANNOTATE_DELTA_COUNTER(id, title, name)`	Defines a custom delta counter. Specify an integer in `id` to uniquely identify the counter, and a string for the chart title in `title` and for the series name in `name`. Use this macro in user space code only.
`ANNOTATE_ABSOLUTE_COUNTER(id, title, name)`	Defines a custom absolute counter. The parameters have the same meanings as for `ANNOTATE_DELTA_COUNTER`. Use this macro in user space code only.
`ANNOTATE_COUNTER_VALUE(id, value)`	Emits a value for a custom delta or absolute counter. Identify the counter using `id` and specify the integer value using `value`. Use this macro in user space code only.
`CAM_TRACK(view_uid, track_uid, parent_track, name)`	Adds a track to a CAM view. To make the track a child of another track, specify the parent track ID in `parent_track`, or specify -1 to make it a root track. `track_uid` must be unique within the view.
`CAM_JOB(view_uid, job_uid, name, track, start_time, duration, color)`	Adds a job with a start time and duration to a CAM track. `job_uid` must be unique within the view. The job is displayed in the **Timeline** view as a colored bar representing its duration. The job name is displayed inside the bar. The start time and duration of the job are specified in nanoseconds. Use `gator_get_time()`, declared in `streamline_annotate.h` to get the current timestamp in nanoseconds.
`CAM_JOB_DEP(view_uid, job_uid, name, track, start_time, duration, color, dependency)`	The single-dependency version of `CAM_JOB_DEPS`.
`CAM_JOB_DEPS(view_uid, job_uid, name, track, start_time, duration, color, dependency_count, dependencies)`	Adds a job with dependencies on other jobs to a CAM track. `track` is the track to add this job to, `dependencies` are the jobs that this job depends on, and `dependency_count` is the number of dependencies.
`CAM_JOB_START(view_uid, job_uid, name, track, time, color)`	Adds a job with a start time to a CAM track. End the job using `CAM_JOB_STOP`.
`CAM_JOB_SET_DEP(view_uid, job_uid, time, dependency)`	The single-dependency version of `CAM_JOB_SET_DEPS`.
`CAM_JOB_SET_DEPS(view_uid, job_uid, time, dependency_count, dependencies)`	Sets the dependencies of a job that was previously started using `CAM_JOB_START`. If you call this macro multiple times, Streamline uses the dependencies with the latest timestamp.
`CAM_JOB_STOP(view_uid, job_uid, time)`	Ends a CAM job that was previously started using `CAM_JOB_START`.
`CAM_VIEW_NAME(view_uid, name)`	Creates a named *Custom Activity Map* (CAM) view. `view_uid` must be unique.

Related concepts

9.1 Annotate overview on page 9-135.
9.5 Kernel annotations on page 9-141.

Related tasks

9.2 Adding string annotations to your code on page 9-137.
9.7 Importing the Streamline_annotate example on page 9-145.

9.7 Importing the Streamline_annotate example

The best way to get started with **Streamline Annotate** is to look at the `Streamline_annotate` example. It contains examples of text, visual, and bookmark annotations, as well as groups and channels.

To import the `Streamline_annotate` example into Eclipse for DS-5, follow these steps.

Procedure

1. Select **File** > **Import...** in Eclipse for DS-5.
2. Use the **Disclosure control** to open the **General Tab** in the **Import dialog** box.
3. Select **Existing Projects into Workspace**.
4. Click **Next**.
5. Use the radio button to activate the **Select archive file** field.
6. Click the **Browse** button next to the **Select archive file** field.
7. Navigate to `../examples` in your DS-5 installation directory.
8. Select the `Linux_examples` archive.
9. Click **Open**.
10. Make sure the `Streamline_annotate` example is checked. Select any other projects that you want to import.
11. Click **Finish**.
    When completed, the `Streamline_annotate` example appears in your **Project Explorer** view along with any other examples you imported. See the `readme.html` for more information about the example.

Related concepts

9.1 Annotate overview on page 9-135.

Related tasks

9.2 Adding string annotations to your code on page 9-137.

Related references

9.6 Annotate macros on page 9-142.

Chapter 10
Log View

Describes the **Log** view, which lists the annotations generated in your code along with information about them.

It contains the following sections:

10.1 Log view column headers

The **Log** view lists all annotations, except CAM annotations, that were generated during your capture session.

Note

To populate the **Log** view, insert `ANNOTATE` statements in your code or add bookmarks to the capture in **Live** view.

The **Log** view contains the following column headers:

When
The number of seconds since the start of the capture session when the message was generated. All messages appear in the **Log** view in chronological order.

Duration
The duration in seconds of the annotation. For bookmark annotations, this value is zero.

Message
The contents of the annotation message. For bookmark annotations, whether they are generated on the target by an annotation or added manually in **Live** view, a bookmark icon ( ) precedes the message text. For visual annotations, a camera icon is displayed. To see the image, select a row with a camera icon.

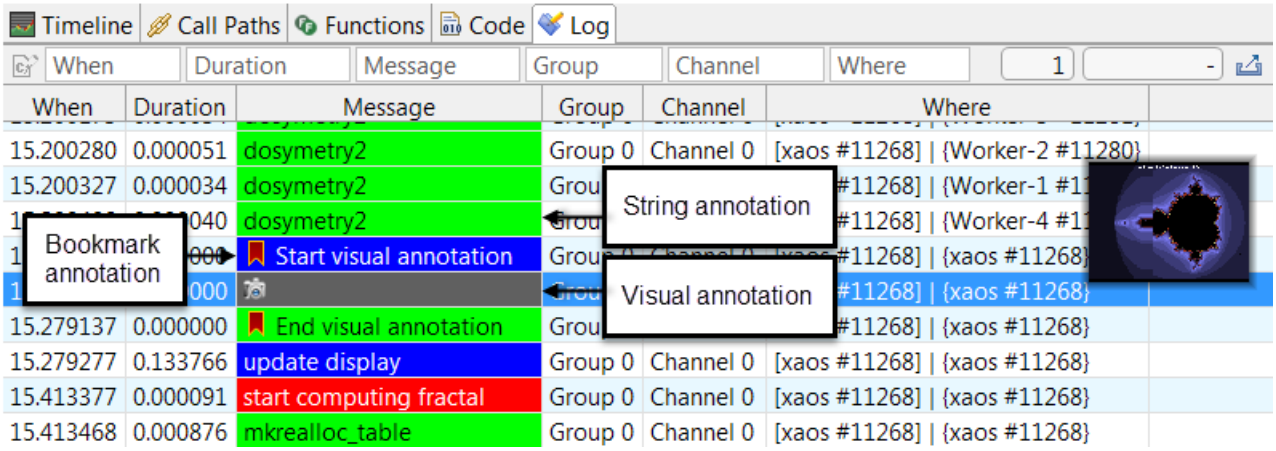


Figure 10-1 Annotations in the Log view

Group
The name of the group to which the annotation channel belongs. **Group 0** is displayed if the channel does not belong to a group.

Channel
The name of the channel to which the annotation belongs. **Channel 0** is displayed if the annotation does not belong to a channel.

Where
The process and thread that generated the message.

Note

Unlike the other table reports in Streamline, you cannot sort the data in the **Log** view.

Related concepts

9.1 Annotate overview on page 9-135.

Related references

9.6 Annotate macros on page 9-142.

10.2 Log view filter fields and toolbar options

The **Log** view provides several filter fields above the table data which enable you to filter the messages displayed.

You can enter a regular expression in any of the filter fields listed below, except the **When** and **Duration** fields. Only messages that match the pattern appear in the list, sorted in chronological order. The **Log** view contains the following filter fields:

When
: Filters the **Log** view based on when Streamline captured the annotation. To define a range, use a dash to separate the lower and upper limits. For example, `4-5` filters the **Log** view to only show annotations captured between seconds 4 and 5 inclusive. To define a range up to a certain number, enter a dash and then a single number. To define a range that includes a number and everything above it, enter a number followed by a dash. For example, to filter for annotations that occurred at the thirty second mark or later in the capture, enter `30-`.

Duration
: Filters the **Log** view based on the duration of the annotation. As with the **When** field, you can enter a number or a range of numbers.

Message
: Shows only messages that match the regex.

Group
: Shows only messages that belong to a particular group.

Channel
: Shows only messages assigned to a particular channel.

Where
: Filters the view based on the name of the process or thread that logged the message.

In addition, a totals panel gives additional information when you select multiple messages in the view. It contains the following fields:

Log Entries
: The total number of entries you have selected in the **Log** view.

Delta
: The time difference in seconds between the selected entries that were logged first and last. To select multiple entries, click on one, then hold down the **Shift** or **Ctrl** key and click on another.

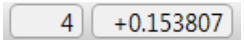


Figure 10-2 Log view totals panel

The **Log** view has the following toolbar options:

Export table to a text file
: Opens the **Export** dialog box, which enables you to export the data from the **Log** view to a text file.

Show Help
: Opens contextual help.

10.3 Log view contextual menu options

Right-click anywhere in the table to open a contextual menu that provides you with options to navigate to selected annotations in either the **Timeline** view or the **Call Paths** view.

The menu contains the following options:

Select Time Range in Timeline
: Opens the **Timeline** view with the Cross Section Marker set to include all of the selected entries.

Select Process/Thread in Timeline
: Opens the **Timeline** view with the Cross Section Marker moved to the location of the selected annotation message.

Select in Call Paths
: Opens the **Call Paths** view. All functions related to the selection in the **Log** view are selected in the **Call Paths** view.

Related concepts

6.3.7 Cross Section Marker on page 6-84.

Related references

*7.6 **Call Paths** view column headers* on page 7-124.

Chapter 11
Capturing Energy Data

Describes how to set up and use the ARM Energy Probe with Streamline to view the power metrics of code running on target hardware.

It contains the following sections:

11.1 Energy Probe overview

To capture energy data for Streamline you must have either an Energy Probe or a supported NI DAQ device.

The Energy Probe is designed to be a low impact, inexpensive solution to give application and system software developers quick feedback on the impact of their code on the system energy footprint. It is intended to provide a better understanding of the static and dynamic behavior of the target system for the purposes of debugging, profiling, and analysis. It is not intended to be a high-precision data acquisition instrument for power benchmarking.

The Energy Probe has three power connectors, each of which is designed to connect to a 2-pin header for measuring the power. This allows the energy probe to provide three independent power, current, or voltage measurements.

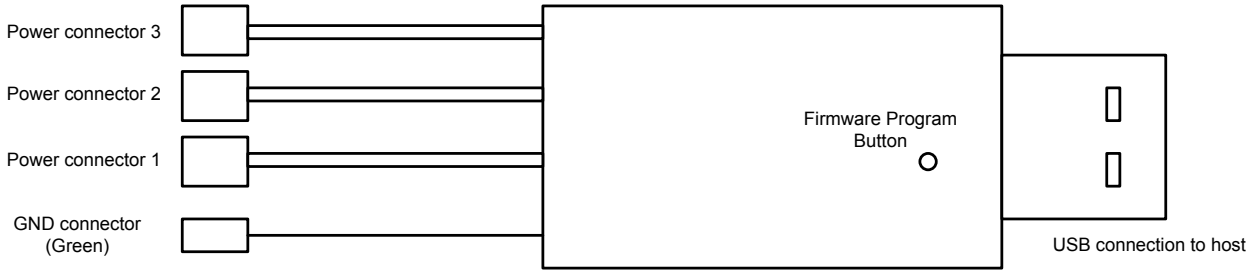


Figure 11-1 Energy Probe schematic

In addition to the three power connectors, the Energy Probe has a single pin GND connector that must be connected to ground on your target board. It provides a ground connection for the Energy Probe.

Each of the three power connectors measures:

- The current flow through a shunt resistor of a known value on your target system
- The voltage at the positive terminals of the Energy Probe.

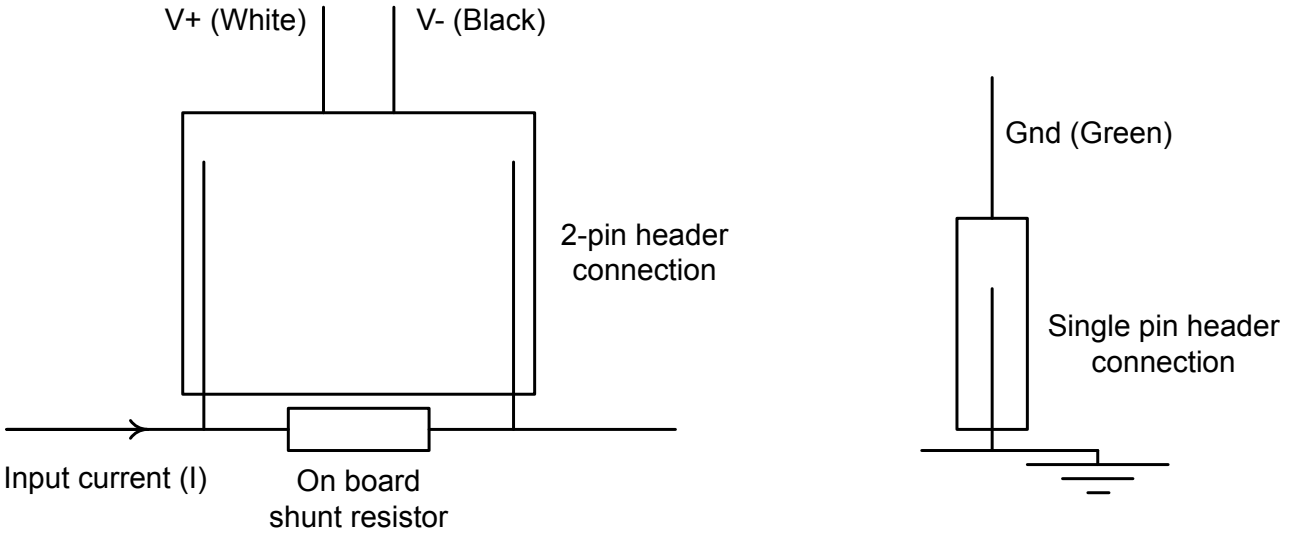


Figure 11-2 Energy Probe electrical connection example

Caution

The Energy Probe has flying leads and must be carefully connected to your target. Do not plug the green ground wire into anything but the target ground. This includes I/O pins and power pins. Doing so could cause damage to your Energy Probe or the power supply of your target.

Related concepts

11.7 Energy Probe data in Streamline on page 11-160.

Related tasks

11.4 Setting up Energy Probe on page 11-156.
11.5 Adding the caiman application to Streamline on page 11-158.

Related references

11.2 Energy Probe requirements on page 11-154.

11.2 Energy Probe requirements

Using the Energy Probe with Streamline requires the Energy Probe hardware, a DS-5 installation, and a target with a shunt resistor that meets the specifications required by the Energy Probe.

The Energy Probe has the following requirements:

- An installation of ARM DS-5 Community, Professional, or Ultimate Edition.
- A suitable DS-5 license.
- A Streamline-enabled target running Linux kernel version 3.4 or later.
- An Energy Probe unit.
- A USB extension cable.
- USB drivers for the Energy Probe.
- A target that has 2-pin IDC 0.1" power measurement headers. The target also requires a shunt resistor with a supply voltage less than 15V and rated at least 0.5W. The shunt resistor needs a 1-pin IDC ground terminal and must not drop more than 165mV.

Note

You can tell if your target has the right power management headers by visual inspection. For more information about whether or not your target meets the other requirements for Energy Probe, see the documentation for your target.

Related concepts

11.1 Energy Probe overview on page 11-152.
11.7 Energy Probe data in Streamline on page 11-160.

Related tasks

11.4 Setting up Energy Probe on page 11-156.
11.5 Adding the caiman application to Streamline on page 11-158.

11.3 Shunt resistor selection for Energy Probe

With 20x amplification, Energy Probe requires the correct selection of a shunt resistor to provide the best possible dynamic range in power measurement, while avoiding saturation of the input of Energy Probe.

A shunt resistor with a value that is too low reduces measurement dynamic range, resulting in less resolution in the power data. A shunt resistor with a value that is too high can cause the input stage of the Energy Probe to saturate, which causes a flat line in the charts that are related to Energy Probe in the **Timeline** view.

To avoid input saturation, the drop across the shunt resistor must never be more than 165mV. You can also use the following equation to determine whether your shunt resistor is appropriate:

```
RShunt(max) = 165 x Vsupply / ( 1000 x Power )
```

`Vsupply` is the input/core voltage. `Power(max)` is the maximum power that the Energy Probe measures. `Rshunt(max)` is the maximum value of the shunt resistor. A shunt resistor value that is greater than `Rshunt(max)` might cause input saturation.

This equation provides the absolute maximum value for the shunt resistor. Use a value that is more than five percent lower than this to allow for component tolerances.

Note

When connecting the Energy Probe, consider the following:

- The black and white probe closest to the green wire is Channel 0.
- For best results, attach Channel 0 to the power source which best represents the CPU load. Streamline aligns the power data with the software activity by maximizing the correlation of Channel 0 with the CPU load.
- The probe white wire is V+. The black wire is V-.

Examples

- 5V power supply, 8W(max), `Rshunt(max)` = 100 milliohms.
- 1V core voltage, 2.5W(max), `Rshunt(max)` = 50 milliohms.
- 1.5V core voltage, 0.4W(max), `Rshunt(max)` = 500 milliohms.

11.4 Setting up Energy Probe

It is critical that you set up your Energy Probe correctly. Failure to do so can result in missing power data in Streamline, and, in some cases, damage to your target.

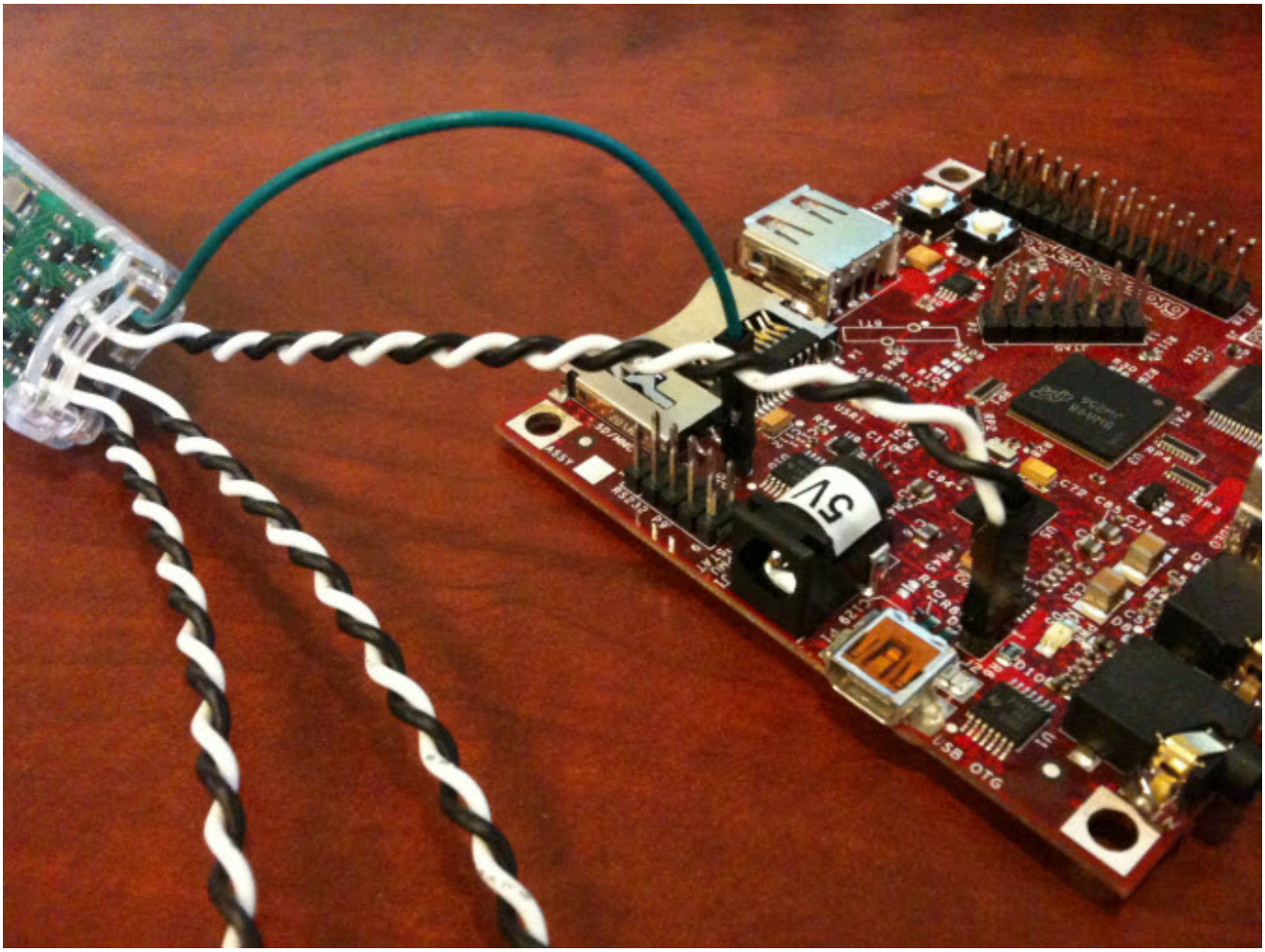


Figure 11-3 Connection to target

To set up Energy Probe, follow these steps:

Procedure

1. Connect your target to your host through a USB port.
2. If necessary, install drivers using the standard Windows driver installation.

   When first plugging in the Energy Probe on Windows, the LED flashes RED at approximately one second intervals. This indicates the USB enumeration has failed and that you must install the driver.
3. If prompted to locate the driver file, select the `ARM_EnergyProbe.inf` configuration file, which is located in *`DS-5_install_directory`*`/sw/streamline/energy_meter/energy_probe/`.
   When you have installed the drivers, and Energy Probe is operating correctly, the LED remains green.
4. To set it up on Linux Ubuntu, enter the following command:

   ```
   sudo apt-get install libudev-dev
   ```
5. On Linux Ubuntu, you must do either of the following for Energy Probe to successfully use the device:
   - Change the permissions of the tty device. To do this, enter the following command:

```
chmod 777 tty_location
```

- Add yourself to the same group as the tty device. To do this, enter the following command:

```
usermod -a -G Group_Name User_Name
```

`dialout` is the usual name of the group for the `tty` device.

If you attach a probe to the target with the channel wire connector the wrong way around, this causes the LED to turn red and blink. This does not damage your target or the Energy Probe but it does not provide useful data. If this happens, disconnect all Energy Probes and wait until the LED changes to green, then reattach them one by one and make sure you reverse any probe that caused the LED to change to red. When you have attached the Energy Probe correctly, the LED remains green.

Note

In case of low consumption, it is possible that the LED remains red even if the connection is correct. This might happen if the target device is idle.

6. If your target has headers available for power or energy metering, attach the ground wire, then attach the probes.
7. Attach the probes.

   Make sure that the probe polarity is correct by observing whether the LED changes from green to red. The probe white wire is V+. The black wire is V-.

Related concepts

11.1 Energy Probe overview on page 11-152.
11.7 Energy Probe data in Streamline on page 11-160.

Related tasks

11.5 Adding the caiman application to Streamline on page 11-158.
11.6 Updating your firmware on page 11-159.

Related references

11.2 Energy Probe requirements on page 11-154.

11.5 Adding the caiman application to Streamline

For Streamline to communicate with the Energy Probe, you must give the location of the `caiman` application using the **Tool Path** field in the **Capture & Analysis Options** dialog box.

To configure Streamline for the Energy Probe, follow these steps:

Procedure

1. Select the ARM Energy Probe option from the **Energy Capture** drop-down menu.
2. Click the **Select the energy capture tool** button in the **Tool Path** field.
3. Locate `caiman.exe` in your Streamline installation directory, select it, and click **Open**.

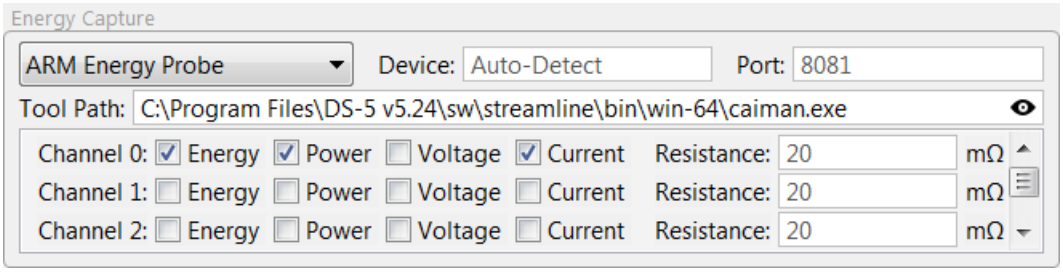


Figure 11-4 Tool Path field

Note

- It might be necessary to specify the tty device when using the ARM Energy Probe on Linux. To do so, enter the following in the **Device** field: `/dev/ttyACM0`. When using NI DAQ, the device name is usually `Dev1`.
- On Windows, you can leave the **Device** field empty. Streamline auto-detects your energy probe.

Related concepts

11.1 Energy Probe overview on page 11-152.
11.7 Energy Probe data in Streamline on page 11-160.

Related tasks

11.4 Setting up Energy Probe on page 11-156.

Related references

11.2 Energy Probe requirements on page 11-154.

11.6 Updating your firmware

Necessary updates occasionally become available for the Energy Probe firmware.

Follow these instructions to update your firmware:

Procedure

1. Detach the Energy Probe probes and ground from the target.
2. Plug the Energy Probe into a USB port on your Windows host machine.

   Because of a limitation with the USB bootloader from the MCU vendor, updating firmware on a Linux host is not supported.
3. To enable firmware programming mode, insert a paperclip into the small hole on the top near the LED.
4. Press and hold down the button for three seconds.
5. Wait for the LED to turn red, and then off. Firmware programming mode is now active.
6. When the drive folder opens, delete the `firmware.bin` file. This file is a placeholder.
7. Drag and drop the new `emeter_firmware.bin` file onto the drive folder. This file is located in *`DS-5_install_directory`*`/sw/streamline/energy_meter/energy_probe/`.
8. Safely remove the Energy Probe drive from the host machine using the Windows **Safely Remove Hardware** icon.
9. Unplug the Energy Probe from the host machine.
10. Wait for one second and plug it back in.

Related tasks

11.4 Setting up Energy Probe on page 11-156.

11.7 Energy Probe data in Streamline

After you have successfully set up the Energy Probe and have run a capture session, you can see the Energy Probe data aligned to the software activity in the **Timeline** view.

If your target meets the requirements, and you have attached a power probe point that closely resembles the activity of the CPU cores using Channel 0, the Streamline correlation algorithm aligns the data for you. Notes that appear when you click the warnings tag on the left side of the toolbar communicate any limitations or failures in this area.

When you have set up everything correctly, run a standard report capture and analysis. A large variance between idle and activity ensures that the correlation algorithm has something to lock onto, so set your workloads accordingly. If you have connected and configured your Energy Probe correctly, a power chart appears in the **Timeline** view.

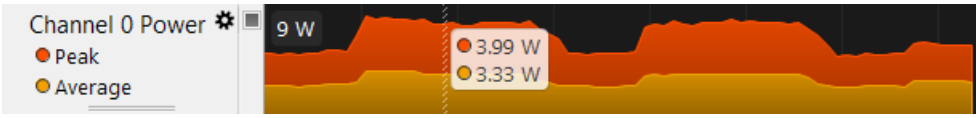


Figure 11-5 Energy Probe data in the Timeline view

Related concepts

11.1 Energy Probe overview on page 11-152.

Related tasks

11.4 Setting up Energy Probe on page 11-156.

11.5 Adding the caiman application to Streamline on page 11-158.

Related references

11.2 Energy Probe requirements on page 11-154.

11.8 Setting up National Instrument Multifunction Data Acquisition devices (NI DAQ) to capture energy data

As an alternative to the Energy Probe, you can use a NI DAQ device to gather energy data from your target and view the results in Streamline.

To collect power statistics using a NI DAQ device, you must have the following:

- NI DAQ hardware.
- NI-DAQmx software installed on your host machine, so `caiman` can communicate with the NI DAQ device. This software package includes drivers for the NI DAQ device. You must use the NI-DAQmx Base software on Linux.

In addition, you must set the location of the appropriate `caiman` application using the Tool Path field in the **Capture & Analysis Options** dialog box.

Note

National Instruments only distribute 32-bit versions of their libraries. Therefore only the 32-bit version of `caiman` works with the NI-DAQ device, even on 64-bit platforms. For example, a Windows 64-bit installation of DS-5 contains a 32-bit version of `caiman`.

If the pre-built `caiman` executable that is distributed with Streamline is insufficient, or you want to change some options, you can re-build `caiman` from source as follows:

1. Extract the source.
2. Open `CMakeLists.txt`.
3. Modify the settings at the beginning of the file as required. If necessary, modify `include_directories` and `target_link_libraries` to add other dependencies, such as NI-DAQ.
4. Use CMake to generate either a Makefile or a Visual Studio project.
5. Build the project as normal.

Note

`caiman` uses CMake so that one configuration can generate both Visual Studio and Makefile projects. See *http://www.cmake.org* for more information about CMake.

For example, to build a NI DAQ enabled version of `caiman` for Red Hat Enterprise Linux 6, set the following values in `CMakeLists.txt`:

- `SUPPORT_UDEV 0`.
- `SUPPORT_DAQ 1`.
- `NI_RUNTIME_LINK 0`.

Also, verify the NI DAQ install paths within `CMakeLists.txt`.

See the National Instruments website, *http://www.ni.com* for the list of operating systems that NI DAQ supports.

To set up your NI DAQ device, follow these steps:

Procedure

1. Connect the Ai1 connections on the NI DAQ device to go across the shunt resistor on your target.
2. Connect Ai0 negative to ground.
3. Connect Ai GND to ground.
4. Loop Ai0 positive to Ai1 negative.

   The Ai0 connectors are measured across the load, which is used to derive the voltage.

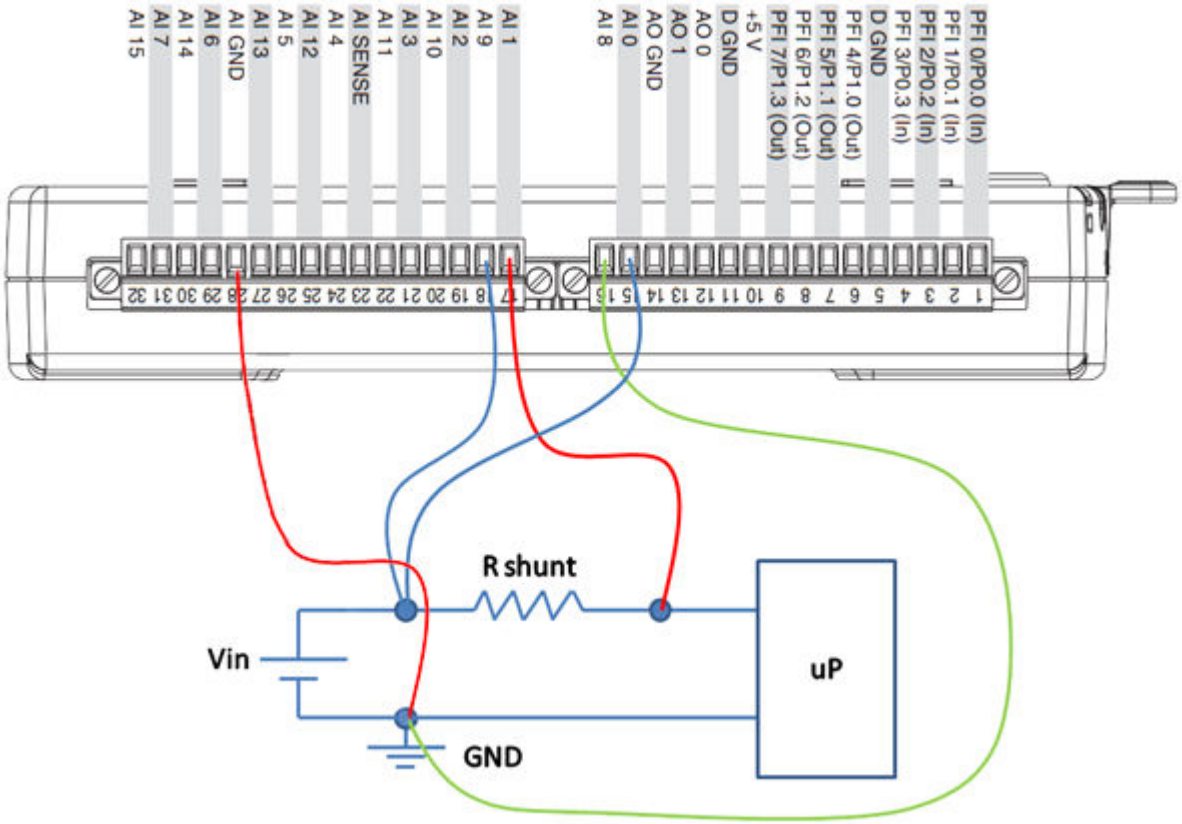


Figure 11-6 Connections for NI USB-621x

5. Repeat steps 1-3 using the other connectors on the NI DAQ device to measure additional channels.
6. In the **Streamline Data** view, click **Capture & Analysis Options**.
7. Select the **NI DAQ** option from the **Energy Capture** drop-down menu.
8. Enter a valid system name in the **Device** field.

   To get the system name of your National Instruments device, you must run the `NI-DAQmx Base List Devices` application, which is installed as part of the NI-DAQmx software package.
9. Click the **Select the energy capture tool** button in the `Tool Path` field.
10. Locate either the pre-built `caiman` in your DS-5 installation directory, or the modified version that you built from source, and select it.
11. Click **Open**.

Note

The NI DAQ device takes a while to initialize with the NI-DAQmx Base drivers, therefore the first 3-8 seconds of power data is not captured.

Related references

4.1 Capture & Analysis Options dialog box settings on page 4-61.

Chapter 12
Advanced gator Customizations

Describes how to customize the more advanced collection and reporting features of Streamline.

It contains the following sections:

12.1 Capturing data on your target without an Ethernet connection

Typically, ARM Streamline uses an active network connection to send capture data from the target to the host. If this is not possible because of limitations with the target, you can save the data to local storage on the target and then manually transfer it to the host.

To capture data locally, follow these steps:

Procedure

1. Create a `session.xml` file. To do so, use the **Capture & Analysis Options** dialog box to set your capture options, then click the **Export...** button to save the new `session.xml` file.

   Alternatively, copy a `session.xml` file from an existing capture.
2. Create a `configuration.xml` file. To do so, define the target name or IP address, then open the **Counter Configuration** dialog box. Set up the counters you want to collect, then click **Save**. A new `configuration.xml` file is created in the directory where `gatord` is running. This file is automatically used when you run `gatord`. To select a different configuration file, use the `-c` option when you run `gatord`.

   If your target is not TCP/IP capable, you must create a `configuration.xml` file and manually transfer it to your target. For instructions on how to do this, see *12.2 Creating a configuration.xml file* on page 12-165.
3. Enter the following command:

   ```
   ./gatord -s session.xml -o <CaptureName>.apc &
   ```

   The `-s` option defines the location of `session.xml` and the `-o` option defines the name and location of the output capture directory.
4. Stop the capture session. You can do so in a number of different ways:
   - Specify a maximum duration in seconds using the `duration` attribute in `session.xml`. Alternatively, use the **Duration** field in the **Capture & Analysis Options** dialog.
   - In the **Capture & Analysis Options** dialog box, set the value of **Buffer Mode** to something other than streaming. Use one of the following values: **Large**, **Normal**, or **Small**. A **Large** store-and-forward buffer is 16MB, while **Normal** is 4MB, and **Small** is 1MB. The profiling session terminates automatically when it reaches the set buffer size.
   - Press **Ctrl+C** on the console to interrupt `gatord`. `gatord` must be running in the foreground.
   - Determine the process id of `gatord` and enter the kill command: `kill process_ID`.

   When the capture stops, Streamline creates an `.apc` directory on the target containing the data and `.xml` files.
5. Transfer the newly created `.apc` directory to your host.
6. Click the **Edit Locations...** button () in the **Streamline Data** view.
7. Choose the directory that contains the `.apc` directory that you transferred from your target.
8. If you want to define image files for better data in the report views, double-click on the capture in the **Streamline Data** view to open the **Analyze** dialog box.
9. Define your image files using the **Analyze** dialog box. Image files must match the images that you ran on your target during the local capture session.
10. Click **Analyze**.

Related tasks

12.2 Creating a configuration.xml file on page 12-165.

12.2 Creating a configuration.xml file

A `configuration.xml` file defines the set of counters and their attributes that you want Streamline to collect from the target.

You normally configure counters using the **Counter Configuration** dialog, but if you have no network connection to your target, then you can manually create a `configuration.xml` file before running a capture session.

Note

Streamline defines a default set of counters. If this is sufficient, you do not need to create a `configuration.xml` file.

Procedure

1. Start `gatord`, then run `ls /dev/gator/events` to list all the counters that your target supports. The list might look like this:

```
ARMv7_Cortex_A9_ccnt  Linux_block_rq_wr          Linux_power_cpu_freq
ARMv7_Cortex_A9_cnt0  Linux_cpu_wait_contention  Linux_proc_statm_data
ARMv7_Cortex_A9_cnt1  Linux_cpu_wait_io          Linux_proc_statm_share
ARMv7_Cortex_A9_cnt2  Linux_irq_irq              Linux_proc_statm_size
ARMv7_Cortex_A9_cnt3  Linux_irq_softirq          Linux_proc_statm_text
ARMv7_Cortex_A9_cnt4  Linux_meminfo_bufferram    Linux_sched_switch
ARMv7_Cortex_A9_cnt5  Linux_meminfo_memfree      mmapped_cnt0
L2C-310_cnt0          Linux_meminfo_memused      mmapped_cnt1
L2C-310_cnt1          Linux_net_rx               mmapped_cnt2
Linux_block_rq_rd     Linux_net_tx
```

2. Counters whose name does not end in `_cnt<n>`, for example `Linux_block_rq_rd`, support a single event only. To find out which event these counters support, search for the counter name in `events-*.xml` in the gator daemon source code, for example:

```
$ grep Linux_block_rq_rd events-*.xml
events-Linux.xml:    <event counter="Linux_block_rq_rd" title="Disk I/O" name="Read" units="B"
description="Disk I/O Bytes Read"/>
```

   This shows that the `Linux_block_rq_rd` counter is associated with the Disk IO: Read event.

3. Counters whose name ends in `_cnt<n>` usually support more than one event. To find out which events these counters support, search for the `events-*.xml` file they are defined in.
   For example, to search for the file that defines the `ARMv7_Cortex_A9` counters, use the following command:

```
$ grep -l ARMv7_Cortex_A9 events-*.xml
events-Cortex-A9.xml
```

   Next, view the contents of `events-Cortex-A9.xml` to find out which events the `ARMv7_Cortex_A9` counters correspond to, for example:

```
$ cat events-Cortex-A9.xml
  <counter_set name="ARMv7_Cortex_A9_cnt" count="6"/>
  <category name="Cortex-A9" counter_set="ARMv7_Cortex_A9_cnt" per_cpu="yes"
supports_event_based_sampling="yes">
    <event counter="ARMv7_Cortex_A9_ccnt" event="0xff" title="Clock" name="Cycles" display="hertz" units="Hz"
average_selection="yes" average_cores="yes" description="The number of core clock cycles"/>
    <event event="0x00" title="Software" name="Increment" description="Incremented only on writes to the
Software Increment Register"/>
    <event event="0x01" title="Cache" name="Instruction refill" description="Instruction fetch that causes a
refill of at least the level of instruction or unified cache closest to the processor"/>
    <event event="0x02" title="Cache" name="Inst TLB refill" description="Instruction fetch that causes a TLB
refill of at least the level of TLB closest to the processor"/>
...
```

   This shows that the `ARMv7_Cortex_A9_ccnt` counter is the Clock: Cycles event. It also shows that the `ARMv7_Cortex_A9_cnt<n>` counters can be associated with a large number of events, including Software: Increment, Cache: Instruction refill, and Cache: Inst TLB refill.

   After following this step, you should know which events are available for your target.

4. Create an empty `configuration.xml` file. Copy and paste into it the counter, event (if any), title, and name attributes from the `<event .../>` nodes for the events you are interested in.
   For example, to add the Disk IO: Read event, copy the required attributes from `events-Linux.xml`. Your `configuration.xml` file should look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<configurations revision="3">
        <configuration counter="Linux_block_rq_rd" title="Disk I/O" name="Read"/>
</configurations>
```

Note

The `Linux_block_rq_rd` counter does not have an event attribute.

To add another event, for example Clock: Cycles, copy its attributes, including the event attribute, into a new `<configuration>` node:

```
<?xml version="1.0" encoding="UTF-8"?>
<configurations revision="3">
        <configuration counter="Linux_block_rq_rd" title="Disk I/O" name="Read"/>
        <configuration counter="ARMv7_Cortex_A9_ccnt" event="0xff" title="Clock" name="Cycles"/>
</configurations>
```

To add an event that does not have a counter attribute, for example Cache: Instruction refill, you must choose a counter value from the category for the event. For example, the category for the Cache: Instruction refill event uses `counter_set="ARMv7_Cortex_A9_cnt"`. This means you can choose one of the unused `ARMv7_Cortex_A9_cnt<n>` counters to associate with the Cache: Instruction refill event, for example `ARMv7_Cortex_A9_cnt0`. Your `configuration.xml` file should look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<configurations revision="3">
        <configuration counter="Linux_block_rq_rd" title="Disk I/O" name="Read"/>
        <configuration counter="ARMv7_Cortex_A9_ccnt" event="0xff" title="Clock" name="Cycles"/>
        <configuration counter="ARMv7_Cortex_A9_cnt0" event="0x01" title="Cache" name="Instruction refill"/>
</configurations>
```

Note

You can only associate each `_cnt<n>` counter with a single event at the same time, so you cannot include all possible events in this `configuration.xml` file.

These steps should enable you to create a `configuration.xml` file for a specific target to capture the events you are interested in.

12.3 Capturing energy data locally and importing it to a capture

If you want to run a capture session locally and want the resulting report to include energy data, it is possible to run `caiman` and `gatord` simultaneously and then merge the data from the two captures into a single report.

To import energy data into a capture, follow these steps:

Procedure

1. Open the **Capture & Analysis Options** dialog box.
2. Enable energy capture by selecting **ARM Energy Probe** or **NI DAQ** from the Energy Capture drop-down menu, then enter the settings for each channel.
3. Enter all other options as normal.
4. Export a `session.xml` file using the **Export...** option.
5. On the host, start a `caiman` local capture where the channel/resistance pairs match what you configured in the exported `session.xml`. For example:

   ```
   caiman -l -r 0:20 -r 1:30
   ```

6. At the same time, trigger a gator local capture using the `session.xml` previously exported:

   ```
   gatord -s session.xml -o <YourFileName>.apc
   ```

7. When ready, stop the gator capture.
8. Immediately stop the `caiman` capture.
9. On your host machine, create a sub-folder in the .apc folder of the gator capture. Name the new folder `energy`.
10. Copy the following `caiman` local capture files to the newly created `energy` folder: `0000000000`, `captured.xml`, and, if it exists, `warnings.xml`.
11. In the **Streamline Data** view, right-click on the merged capture and select **Analyze...** from the contextual menu.
12. Use the **Offset energy chart data** buttons in the Chart Configuration panel in the **Timeline** view to match the energy data with other capture data.

12.4 Using the gator_events_mmapped.c custom counters example

The file `gator_events_mmapped.c` is provided as an example of how to add custom counters to code that uses kernel space gator. It is located in the gator driver source code.

Incorporating the simulated examples from `gator_events_mmapped.c` into gator is a good way to familiarize yourself with the process of adding your own counters.

The gator driver source code is available from either of the following locations:

- *`DS-5_install_directory`*`/sw/streamline/gator/driver/`
- *https://github.com/ARM-software/gator*

To add a custom counter using `gator_events_mmapped.c`, follow these steps:

Procedure

1. Open the `gator_events_mmapped.c` example file in the editor of your choice.
2. Copy the XML from the comments section of `gator_events_mmapped.c`.
3. Create an XML file in the same directory as the `gatord` source code, and call it `events-mmap.xml`.
4. Add the copied XML from the comments section of `gator_events_mmapped.c` to `events-mmap.xml`.
5. Remove any * comment markers from the copied XML.
6. Save `events-mmap.xml`.
7. Rebuild `gatord` and copy it to the target.
8. Kill the old `gatord` process if it is already running, then enter `./gatord &` on the command line of your target to launch the newly built `gatord`.
9. Open the **Counter Configuration** dialog using the button in the **Streamline Data** view.
   A new category, **mmapped**, appears in the **Counter Configuration** dialog box with the **Sine**, **Triangle**, and **PWM** simulated counters.
10. Add **Sine** to list of counters.
11. Run a capture session.
    If successful, the waveform generated by the simulated **Sine** counter appears in the charts section of the **Live** and **Timeline** views.

Figure 12-1 Sine counter chart

Related references

12.6 gator_events functions on page 12-171.
12.8 Attributes for custom, filesystem, and ftrace counters in the events XML file on page 12-173.

12.5 Creating custom performance counters with kernel space gator

In addition to the hardware-specific and Linux performance counters that you can configure using the **Counter Configuration** dialog, the gator daemon and driver provide hooks that enable you to create custom counters.

Note

This topic only applies if you are using kernel space gator. If you are using user space gator, create custom counters using the macros `ANNOTATE_ABSOLUTE_COUNTER`, `ANNOTATE_DELTA_COUNTER`, and `ANNOTATE_COUNTER_VALUE` instead.

Streamline derives its default set of counters from the performance monitoring unit, Linux hooks, and memory-mapped peripherals. You can add your own counters to this list if there is a hardware metric that you want to track which Streamline does not provide by default.

To create your own counters, mimic the methods used in the `gator_events_mmapped.c` file or any of the other `gator_events_`*`x`* files included with the gator driver source.

The gator driver source code is available from either of the following locations:

- *`DS-5_install_directory`*`/sw/streamline/gator/driver/`
- *https://github.com/ARM-software/gator*

Follow these steps to ensure that `gatord` interacts with your custom source:

Procedure

1. Create an empty `gator_events_`*`your_custom`*`.c` file or duplicate `gator_events_mmapped.c`.
2. Update the makefile to build the new `gator_events_`*`your_custom`*`.c` file.
3. Add the preprocessor directive `#include "gator.h"` if you do not use `gator_events_mmapped.c` as a template.
4. Implement the following functions in your new source file: `gator_events_`*`your_custom`*`_init`, `gator_events_`*`your_custom`*`_interface`, `gator_events_`*`your_custom`*`_create_files`, `gator_events_`*`your_custom`*`_start`, `gator_events_`*`your_custom`*`_read`, and `gator_events_`*`your_custom`*`_stop`.
5. Add `gator_events_`*`your_custom`*`_init` to `GATOR_EVENTS_LIST` in `gator_main.c`.
6. All of your new counters must be added to the events list. To do this:
   a. Create a new XML file in the same directory as the makefile before building `gatord`. The makefile pulls in any file that begins with `events-`, so create an XML file called `events-`*`YourCustom`*`.xml`.
   b. Define how many counters exist in your custom set using the `<counter_set name="counter_name" count="x">` tag. Give your counter a unique, descriptive name and enter how many counters are available for the count attribute. For example: `<counter_set name="ARM_Cortex-A9_cnt" count="6"/>`
   c. For each counter set, you must list each of the possible events. Define the event category using the `<category>` tag.
   d. Define the individual events for each event category, including all of the necessary attributes.

      A basic event example:

      ```
      <event event="0x01" title="Cache" name="Instruction refill"
      description="Instruction fetch that causes a refill of at least
      the level of instruction or unified cache closest to the processor"/>
      ```

      A fixed event example:

      ```
      <event counter="ARM_Cortex-A9_ccnt" title="Clock" name="Cycles"
      display="hertz" units="Hz" average_selection="yes"
      description="The number of core clock cycles"/>
      ```

7. Re-build `gatord` after you create your new XML file.

Related concepts

9.1 Annotate overview on page 9-135.

Related references

12.6 gator_events functions on page 12-171.

12.8 Attributes for custom, filesystem, and ftrace counters in the events XML file on page 12-173.

12.6 gator_events functions

To create your own custom counters, you must add gator functions to your source code.

This table gives a brief description of each of the gator events functions:

Gator events function	Description
`gator_events_`*`your_custom`*`_init`	gator calls this function at startup.
`gator_events_`*`your_custom`*`_interface`	Tells gator what triggers calls to your custom events file.
`gator_events_`*`your_custom`*`_create_files`	Adds custom directories and enabled, event, and key files to `/dev/gator/events`.
`gator_events_`*`your_custom`*`_start`	gator calls this at the start of execution.
`gator_events_`*`your_custom`*`_read`	gator calls this at every sample.
`gator_events_`*`your_custom`*`_stop`	gator calls this at the termination of a capture session.

Related tasks

12.4 Using the gator_events_mmapped.c custom counters example on page 12-168.
12.5 Creating custom performance counters with kernel space gator on page 12-169.

12.7 Creating filesystem and ftrace counters

gator supports reading generic files based on entries in events XML files. This feature allows you to add counters to extract data held in files, for example, `/dev`, `/sys` or `/proc` file entries. You can use a similar technique to add counters for `ftrace` data.

To add filesystem counters, you first need to create an `events-xxx.xml` file. Ensure that for each entry in the file, the `counter` attribute values begin with `filesystem_` and are unique. Use `path` attributes to specify the names and locations of the files to read. An example file called `events-Filesystem.xml` containing commented-out entries is provided with the `gatord` source code.

gator reads the files specified in your `events-xxx.xml` ten times per second. By default, it interprets each file as an integer, but if you provide a regular expression using the `regex` attribute, gator applies it and converts the matched entry to an integer.

gator also supports reading `ftrace` data, using a similar technique. In the `events-xxx.xml` file, ensure the `counter` attribute values begin with `ftrace_` and are unique. You can specify regular expressions to extract counter values. For an example, see `events-ftrace.xml` located in the gator daemon source code.

After creating the events XML file, you can either:

- Rebuild `gatord`. In this case, your `events-xxx.xml` file must be in the same location as the `gatord` makefile.
- Restart `gatord` using the `-E` command-line option, specifying the location of your `events-xxx.xml` file. This appends your filesystem and `ftrace` counters to the existing events that gator supports and avoids the need to rebuild `gatord`. If you choose this method, you need to specify the XML header and include an `events` node in your `events-xxx.xml` file. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<events>
  <category name="Filesystem">
    <event counter="filesystem_loginuid" path="/proc/self/loginuid" title="loginuid"
name="loginuid" class="absolute" description="loginuid"/>
    …
  </category>
</events>
```

Note

Some `ftrace` counters are available by default in the **Counter Configuration** dialog, if the target supports them.

Related tasks

2.7 Building the gator daemon on page 2-30.

Related references

12.8 Attributes for custom, filesystem, and ftrace counters in the events XML file on page 12-173.
2.13 gatord command-line options on page 2-39.

12.8 Attributes for custom, filesystem, and ftrace counters in the events XML file

Events XML files define the counters that you can select in the **Counter Configuration** dialog box. In order to add custom, filesystem, or ftrace counters to gator, you must define them in your events XML file.

For descriptions of the valid XML nodes and attributes you can use in the events XML file, see the gator protocol documentation, located in *DS-5_install_directory*`/sw/streamline/protocol/gator/`. In addition, the following attributes are specific to `Filesystem` and `Ftrace` counters:

`regex`
: A POSIX-extended regular expression used to extract a value from a file.

`path`
: The absolute path and name of a file, for example `/proc/stat`.

`enable`
: For `ftrace` counters only, specifies an `ftrace` event to enable. This attribute is optional.

`tracepoint`
: This has the same meaning as `enable`, but uses `perf` instead of `ftrace` when using user space gator. This attribute is optional.

`arg`
: Used in conjunction with `tracepoint` to specify the value to show. This attribute is optional. If not specified, the number of tracepoint events is counted.

Related tasks

12.4 Using the gator_events_mmapped.c custom counters example on page 12-168.
12.5 Creating custom performance counters with kernel space gator on page 12-169.

12.9 Enabling atrace and ttrace annotations

Streamline supports atrace annotations on Android targets that are running Linux kernel versions 3.10 and later, and ttrace annotations on targets that are running Tizen version 2.4.

Streamline converts application-generated atrace macros into either string annotations or counter charts. It also lists any Android `ATRACE_TAG_*` macros that you enable as available events in an **Atrace** section in the **Counter Configuration** dialog. If you expect to see atrace events in this dialog but none are displayed, click on the Warnings tag in the **Counter Configuration** dialog to see why atrace support is not enabled.

To notify running applications that atrace annotation tags have been enabled, the file `notify.dex` must be installed on the target in the same directory as `gatord`. You can install a pre-built version of `notify.dex` as part of target setup, by clicking the **Setup target...** button in the **Connection Browser** dialog. The Java source code for `notify.dex` is available in the following locations:

- *DS-5_install_directory*`/sw/streamline/gator/notify/`
- *https://github.com/ARM-software/gator/tree/master/notify*

To enable Tizen ttrace annotations:

- Specify the path to SDB in the **ADB Path** field in the **Capture & Analysis Options** dialog.
- Select a target Tizen device, with gator running on it, using the **Connection Browser** dialog.
- Select the ttrace events to capture from the **Ttrace** section in the **Counter Configuration** dialog.

Streamline displays the ttrace tags as annotations in the **Timeline** view and **Log** view.

Related concepts

5.2 Counter Configuration dialog box structure on page 5-69.

Related references

5.5 Counter Configuration dialog box settings on page 5-73.

4.1 Capture & Analysis Options dialog box settings on page 4-61.

3.6 Connection Browser dialog box on page 3-53.

12.10 Getting L2C-310 memory-mapped peripherals working with Streamline

The gator driver source provides a file, `gator_events_l2c-310.c`, which contains hard coded offsets for the locations of the L2 cache counter registers. You can configure the offset for your board by specifying a module parameter when loading `gator.ko`.

For example:

```
insmod gator.ko l2c310_addr=<offset>
```

You can disable the l2c-310 counter by providing an offset of zero, for example:

```
insmod gator.ko l2c310_addr=0
```

Related tasks

2.8 Building the gator module on page 2-31.

12.11 Profiling the Linux kernel

If you do not include the kernel in the images in the **Capture & Analysis Options** dialog box, the statistics generated by the kernel are not aligned with source code in the Analysis Reports. Before you can include the Linux kernel in the **Program Images** section of the **Capture & Analysis Options** dialog, you must build a version of vmlinux with kernel debug information enabled.

To profile the Linux kernel, follow these steps:

Procedure

1. Navigate to the kernel build directory.
2. Enter the following command to enable you to change `menuconfig` options:

   ```
   make ARCH=arm CROSS_COMPILE=<cross_compiler_directory>/bin/arm-linux-gnueabihf-
   menuconfig
   ```

3. In the **Kernel Hacking** menu, select the **Compile the kernel with debug info** option. This enables the `CONFIG_DEBUG_INFO` kernel option.

   The options described in *2.6 Required kernel configuration menu options* on page 2-29 must also be enabled.
4. Enter the following command to build the image:

   ```
   make -j5 ARCH=arm CROSS_COMPILE=<cross_compiler_directory>/bin/arm-linux-
   gnueabihf- uImage
   ```

   This creates a new vmlinux image.

   Note

   You can profile a driver by either statically linking it into the kernel image or by adding the module as an image in the **Capture & Analysis Options** dialog box.
5. Optional: Enable kernel stack unwinding using either of the following methods:
   - Remove the comment tags that surround `GATOR_KERNEL_STACK_UNWINDING` in the Makefile and rebuild `gator.ko`.
   - Run the following command as root on the target after `gatord` has started:

     ```
     echo 1 > /sys/module/gator/parameters/kernel_stack_unwinding
     ```

   Note

   - This step is only required if you previously built `gator.ko` with kernel stack unwinding turned off.
   - Enabling kernel stack unwinding might trigger errors that appear at millisecond intervals during the capture session. If you experience this behavior, disable kernel stack unwinding.
6. Open the **Capture & Analysis Options** dialog box.
7. Click the **Add ELF image...** button in the **Program Images** section.
8. Navigate to your vmlinux file and select it.
9. Click OK
10. Start a new capture session.

Related tasks

2.8 Building the gator module on page 2-31.

Related references

2.6 Required kernel configuration menu options on page 2-29.

12.12 Adding support to gatord for a new CPU or perf PMU

Use either of the following ways to add support for a new CPU or perf PMU to `gatord`. The information in this topic applies to gator version 23 and later.

Note

- Perf support in Linux for the PMU is required because `gatord` uses perf to read the hardware counters.
- Only XML changes are required, no code changes are necessary.
- Rebuilding `gatord` after the XML changes is recommended but not required because you can pass PMUs and events to `gatord` on the command line.

Make the following changes, then rebuild `gatord`:

- Add a line to *`DS-5_install_directory`*`/sw/streamline/gator/daemon/pmus.xml` describing the new PMU. For CPUs, the following information is required:
  - The CPU Implementer and Primary part number from the Main ID Register.
  - The number of generic hardware counters that can be selected simultaneously.
  - Optionally, set the perf PMU name of the CPU to ensure correct operation in multi-PMU ARM big.LITTLE™ configurations.
- Create an events XML file, named `events-`*`xxx`*`.xml` in the gator daemon source directory that defines the events that the new PMU generates. This file should exclude the XML header and `<events>` element. See the Cortex-A15 events XML file, *`DS-5_install_directory`*`/sw/streamline/gator/daemon/events-Cortex-A15.xml` for an example.

Alternatively, to add support without having to rebuild `gatord`, do the following:

- Create an events XML file that defines the events that the new PMU generates. This file must include the XML header and `<events>` element, for example:

```
<?xml version="1.0" encoding="UTF-8"?>
<events>
  <counter_set name="ARMv7_Cortex_A9_cnt" count="6"/>
  <category name="Cortex-A9" counter_set="ARMv7_Cortex_A9_cnt" per_cpu="yes"
supports_event_based_sampling="yes">
    <event counter="ARMv7_Cortex_A9_ccnt" event="0xff" title="Clock" name="Cycles" display="hertz" units="Hz"
average_selection="yes" average_cores="yes" description="The number of core clock cycles"/>
    <event event="0x00" title="Software" name="Increment" description="Incremented only on writes to the
Software Increment Register"/>
    <event event="0x01" title="Cache" name="Instruction refill" description="Instruction fetch that causes a
refill of at least the level of instruction or unified cache closest to the processor"/>
    <!-- ... -->
  </category>
</events>
```

- Create an XML file that defines information about the new PMU. For the required format, see the gator `pmus.xml`. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<pmus>
  <pmu pmnc_name="ARMv7_Cortex_A9" cpuid="0x41c09" core_name="Cortex-A9"
pmnc_counters="6"/>
</pmus>
```

- Copy these files to the target and restart `gatord` using the following flags:
  - `-E` to specify the location of the events XML file. This flag causes the events to be appended to the list of events that `gatord` supports.
  - `-P` to specify the location of the PMU XML file. This option causes the new PMU to be added to the list of PMUs defined in `pmus.xml` that `gatord` has built-in support for.

Related tasks

2.7 Building the gator daemon on page 2-30.

Related references

2.13 gatord command-line options on page 2-39.

12.13 Increasing the memory that is available for Streamline

To increase the amount of memory that is available for Streamline, you must modify the Streamline ini file.

Streamline supports multiple operating systems, so you must select the ini file that is specific to your operating system. To find the correct file, see the following table:

Operating system	Ini file location
Windows 64-bit, GUI	*DS-5_install_directory*/sw/streamline/Streamline-gui.ini
Windows 64-bit, command line	*DS-5_install_directory*/sw/streamline/Streamline.ini
Windows 32-bit, GUI	*DS-5_install_directory*/sw/streamline/Streamline-32-gui.ini
Windows 32-bit, command line	*DS-5_install_directory*/sw/streamline/Streamline-32.ini
Linux 64-bit	*DS-5_install_directory*/sw/streamline/.Streamline.ini
Linux 32-bit	*DS-5_install_directory*/sw/streamline/.Streamline-32.ini
Mac OS	*DS-5_install_directory*/sw/streamline/Streamline.app/Contents/MacOS/Streamline.ini

Procedure

1. Open the correct ini file and edit the `-Xmx` setting. For example, to set aside 4GB of memory for Streamline, change it to:

   ```
   -Xmx4g
   ```

   Note

   You might have to change the access permissions for the file before you can edit it.

2. Save and close the file.
3. Restart Streamline.

Related information

"JVM terminated" error when launching DS-5 / OutOfMemoryError reported when using DS-5.

Chapter 13
Bare-Metal Support

Describes the bare-metal support available within Streamline.

It contains the following sections:

13.1 Bare-metal support overview

Bare-metal support allows Streamline to visualize elements of the system state of a target device that is not running a Linux based operating system.

Specifically, support is provided for systems without the following:

- A connected trace device.
- A Linux based operating system for which the `gator` daemon and kernel module are provided.

Bare-metal support is provided by an agent that is referred to as *barman*. It consists of two C source files, `barman.c` and `barman.h` that you build into the executable that runs on the target device. These files are generated by a configuration and generation utility.

In order to use barman, you must modify your existing executable to do the following:

- Initialize barman at runtime.
- Periodically call the data collection routines that barman provides.
- Optionally, stop the capture.
- Optionally, extract the raw data that barman collects and provide it to Streamline for analysis.

Barman has the following features:

- It captures PMU counter values from Cortex-A and Cortex-R class processors.
- It captures sampled PC values.
- It captures custom counters.
- It allows you to control the sample rate.
- It writes the data that it collects to memory.
- It has low data collection overhead.

Barman supports the following ARM architectures:

- ARMv7-R
- ARMv7-A
- ARMv8-A, both AArch32 and AArch64.

See *`DS-5_install_directory`*`/sw/streamline/examples/barman` for Bare-Metal examples.

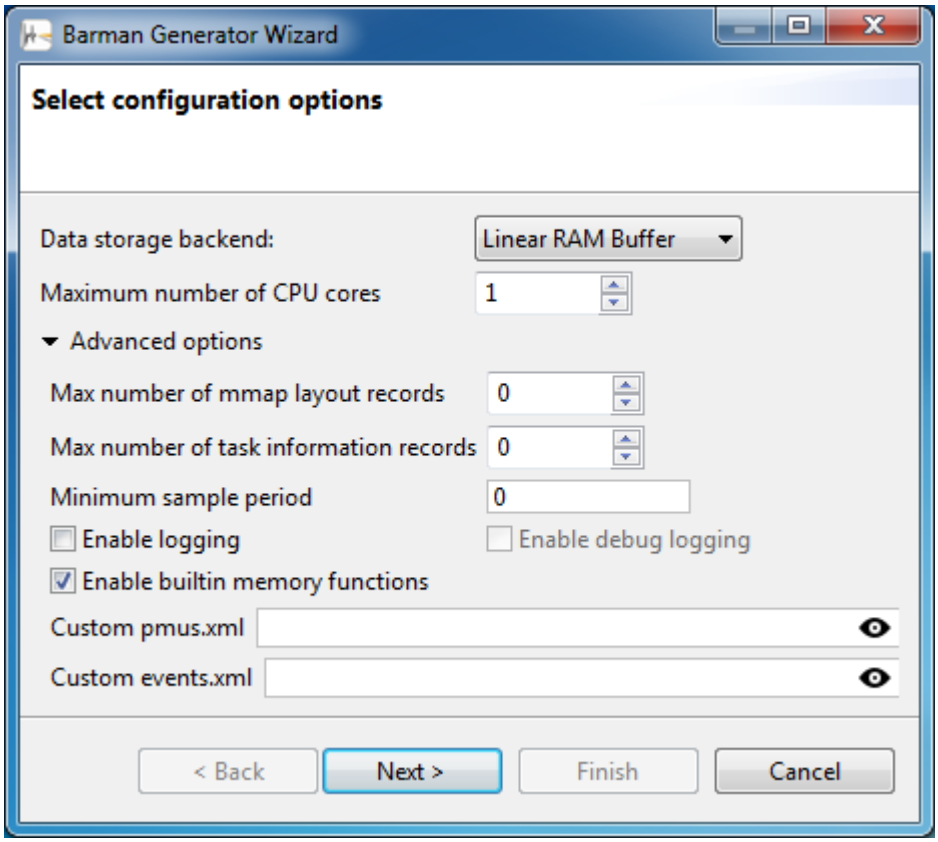
13.2 Configuring barman

You must configure barman with the configuration and generation utility before you compile the binary executable to be analyzed. Barman must then be built into the executable.

The configuration and generation utility is a wizard dialog available from the **Streamline** menu. The generated header and source files, and the configuration XML file, are then saved into a folder of your choice. The generation mechanism is also accessible from the command line.

Procedure

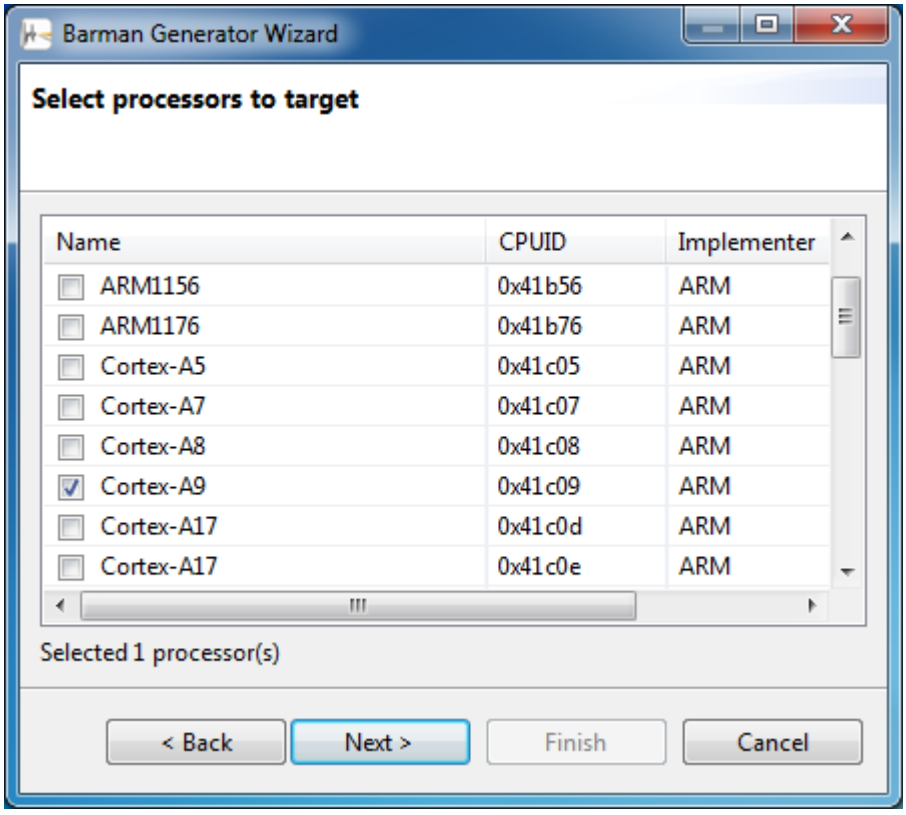
1. Access this utility from **Streamline** > **Generate Barman Sources**.
2. Configure the default configuration options, such as the number of processor elements, whether you intend to supply executable image memory map information, whether you intend to provide process or task level information (for example if you are running an RTOS), and configure data storage mode.



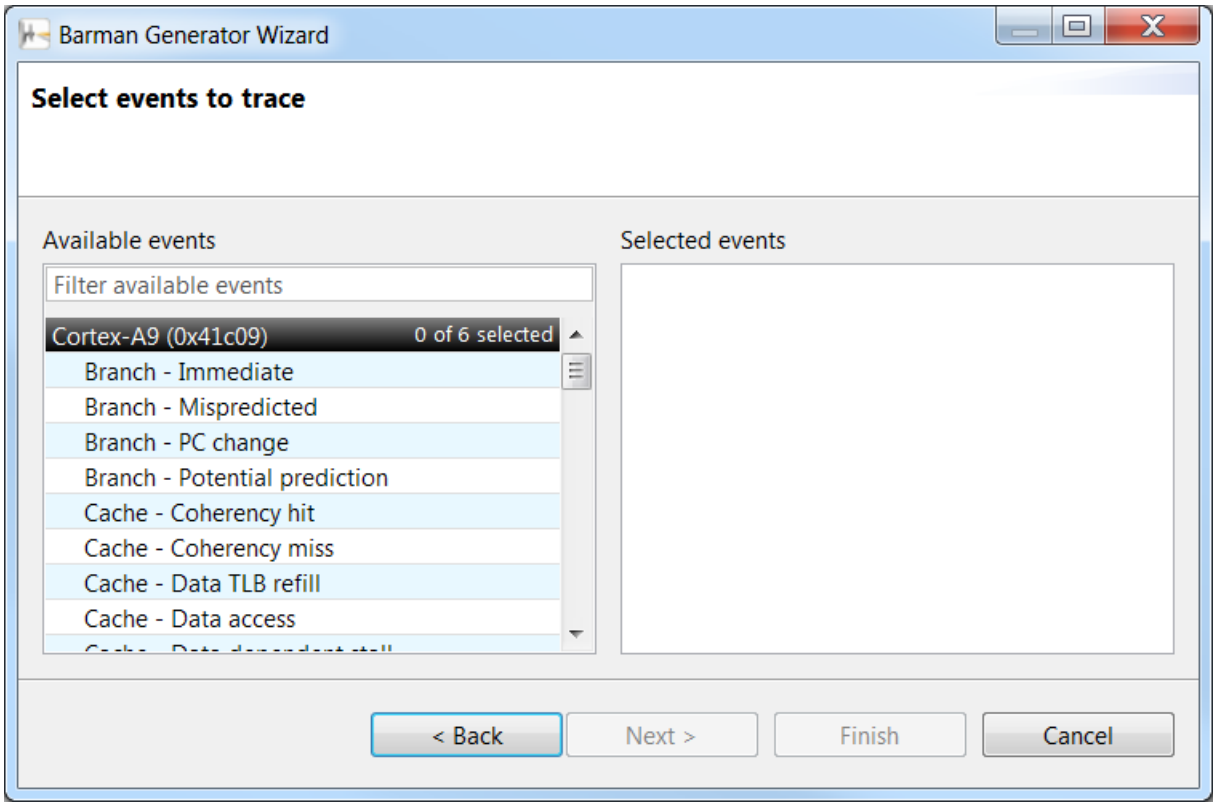
Note

See *Chapter 14 Profiling with System Trace Macrocell* on page 14-201 for information about using the **STM Interface** data storage backend.

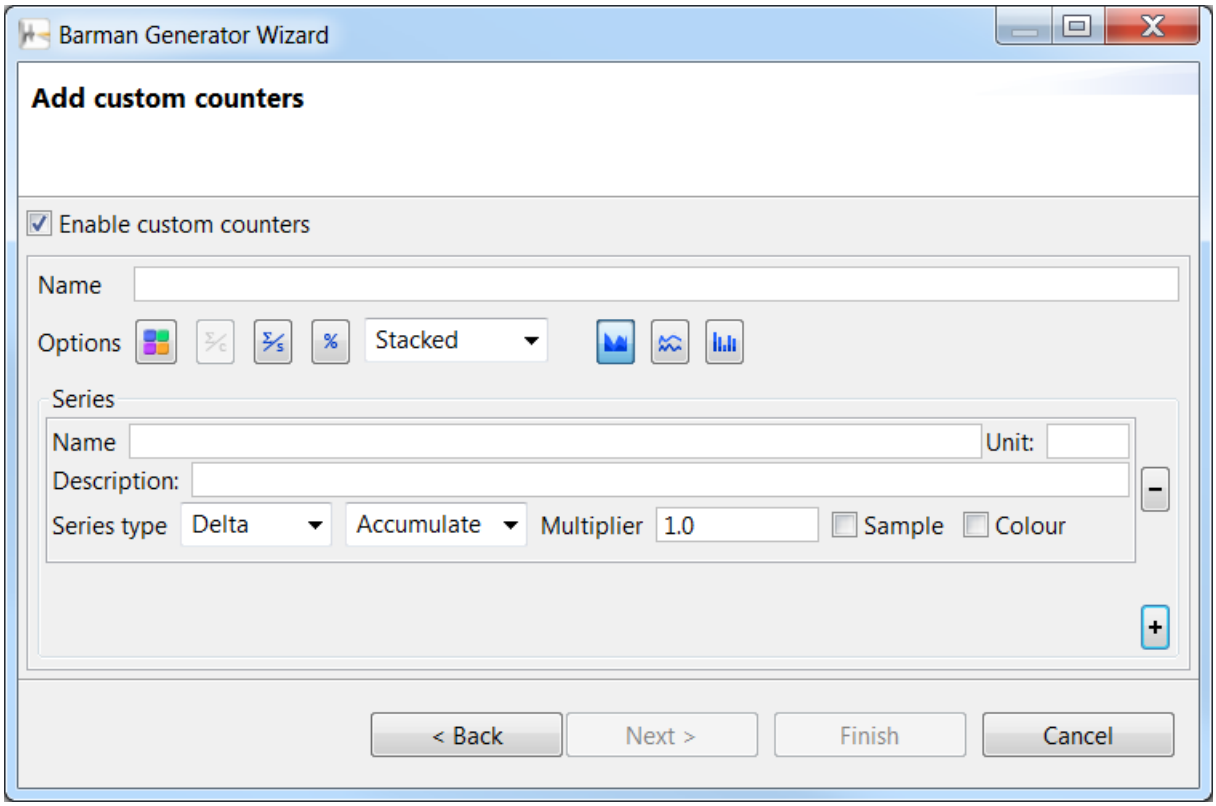
3. Select the target processor from the pre-defined list.



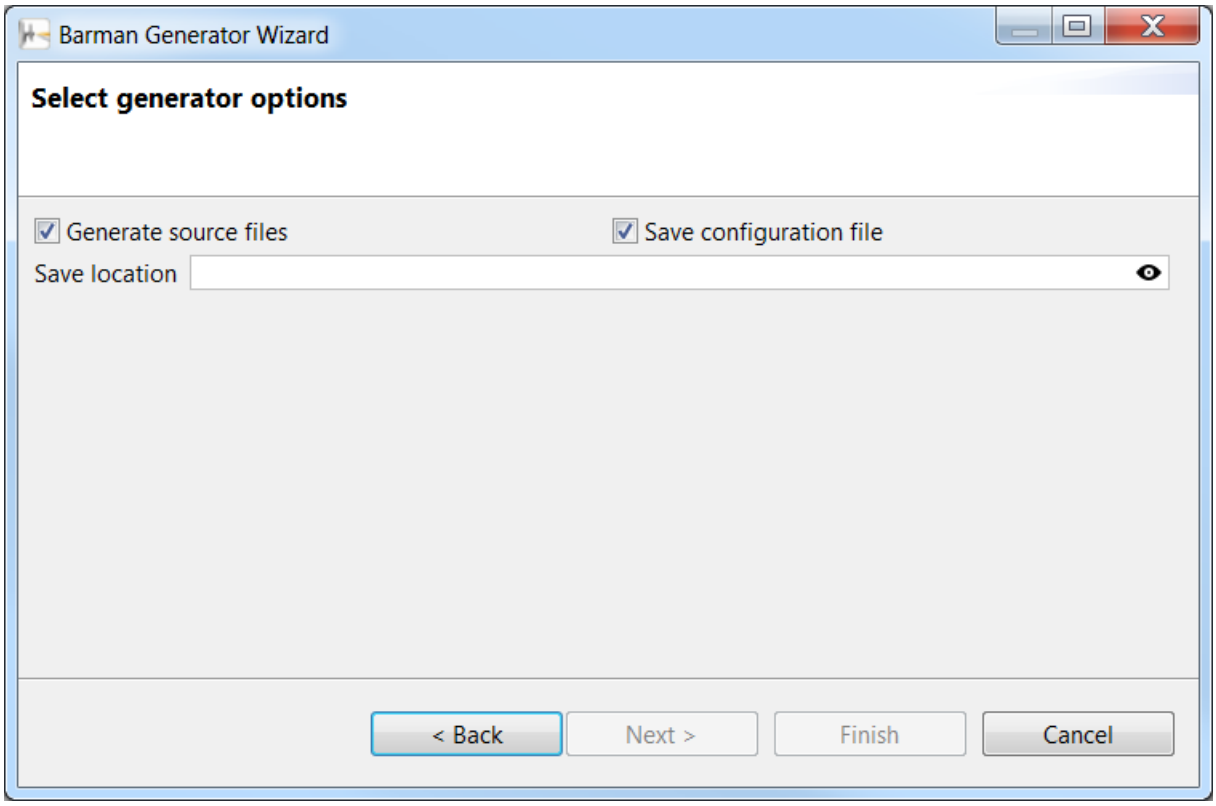
4. Select the PMU counters to collect during the capture session by double clicking on them in the **Available events** list. Alternatively you can drag and drop the events into the **Selected events** list. To deselect events, drag and drop them back into the **Available events** list.



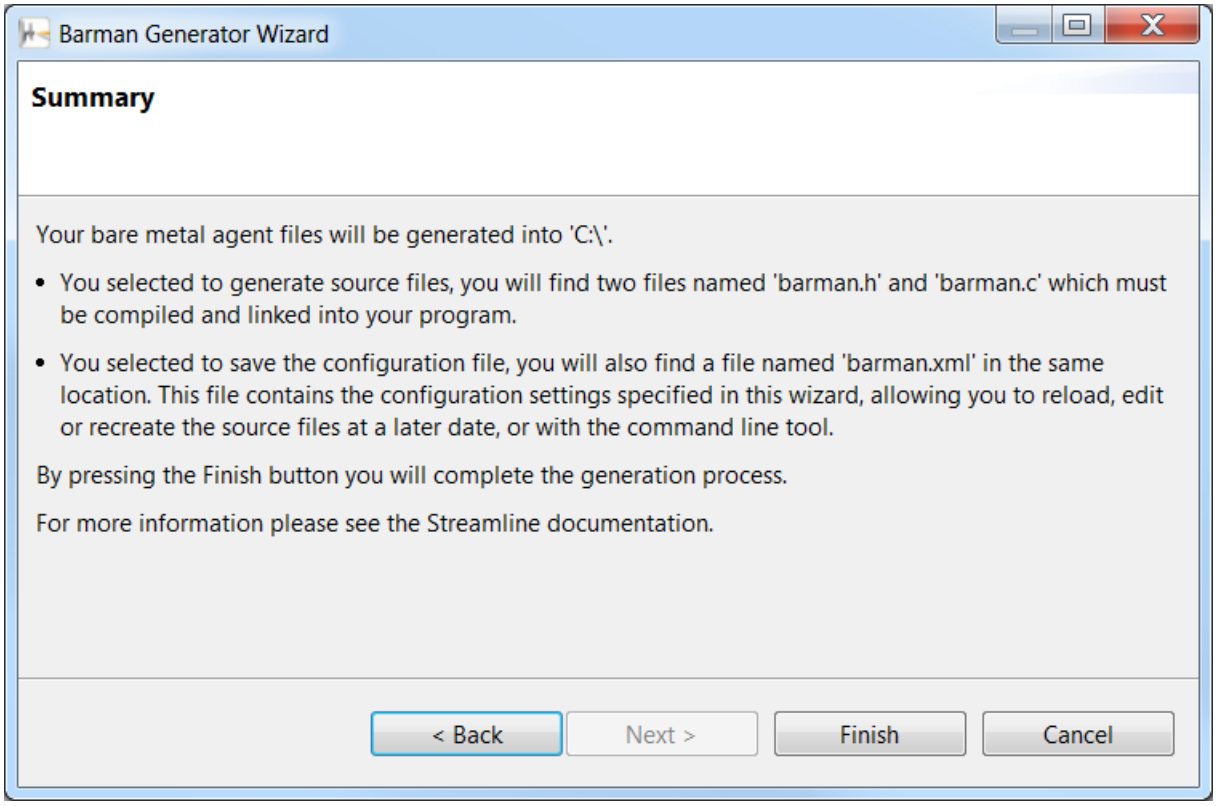
5. Add custom counters.



6. Select generator options.



7. Finish.



The setup process produces the following output:

- A configuration file, `barman.xml`, which contains the settings that were entered into the configuration wizard, and which can be used to reproduce the same configuration later.
- `barman.c`. You must compile and link this file into the bare-metal executable.
- `barman.h`. You must include this header when calling any of the functions within the agent. It also declares function prototypes for the functions you must implement.

You need the compiler flag `--gnu` for ARMCC (ARM Compiler 5) to compile `barman.c`.

Related tasks

16.5 Accessing the Bare-Metal generation mechanism from the command line on page 16-222.

13.3 Custom counters

You can configure one custom chart, with one or more series, in the configuration wizard.

This section contains the following subsections:

13.3.1 Configuring custom counters

The following chart properties can be configured:

Name
Human readable name for the chart.

Series Composition
Defines how to arrange series on the chart (stacked, overlay, or logarithmic).

Rendering Type
Defines how to render series on the chart (filled, line, or bar).

Per Processor
Indicates whether the data in the chart is per processor.

Average Selection
Sets whether the Cross Section Marker in Streamline displays average values.

Average Cores
Sets whether Streamline averages the values of multiple cores when viewing the aggregate data of a per processor chart.

Percentage
Sets whether to display data as a percentage of the maximum value in the chart.

The following series properties can be configured:

Name
Human readable name for the series.

Units
Defines the unit type to display in Streamline.

Sampled
When set to true, the value for this counter is sampled along with the PMU counters. When false, you must call a function to update the counter value.

Multiplier
Number to multiply by for fixed-point math. As the data sent from the agent is int64, it must be scaled. For example, the value 1.23 can be represented by the value 123 with a multiplier of 0.01.

Class
Specifies the nature of the data that is fed into the chart as follows:

delta
Used for values that increment or are accumulated over time, such as hardware performance counters. The exact time when the data occurs is unknown and therefore the data is interpolated between timestamps.

incident
The same as delta, except the exact time is known so no interpolation is calculated. Used for counters such as software trace.

absolute
Used for singular or impulse values, such as system memory used.

Display
The display value determines how to calculate the data when zooming out for each time bin as follows:

accumulate
Sum up the data (valid only for delta and incident class counters).

hertz
Does the same as accumulate then normalizes the value to one second (valid only for delta and incident class counters).

minimum
Display the smallest value encountered (valid only for absolute class counters).

maximum
Display the largest value encountered (valid only for absolute class counters).

average
Display the average (valid only for absolute class counters).

Colour
The color to display the series in. If not set, Streamline selects a color.

Description
Human readable description for the series. This description becomes the tooltip when hovering over the series in Streamline.

13.3.2 Sampled and nonsampled counters

Sampled counters are polled when the PMU counter values are read. For each sampled counter, a function prototype is generated of the following form:

```
extern bm_bool barman_cc_<chart_name>_<series_name>_sample_now(bm_uint64 * value_out);
```

For example:

```
extern bm_bool barman_cc_interrupts_fiq_sample_now(bm_uint64 * value_out);
```

You must implement this function to set the value of the `uint64` at `*value_out` to the value of the counter, then return `BM_TRUE`. If the counter value cannot be sampled, the function can return `BM_FALSE` and be skipped.

You are responsible for writing nonsampled counters to the capture. For each nonsampled series, the following two functions are declared:

```
            bm_bool barman_cc_<chart_name>_<series_name>_update_value(bm_uint64
timestamp, bm_uint32 core, bm_uint64 value);
            bm_bool barman_cc_<chart_name>_<series_name>_update_value_now(bm_uint64
value);
```

For example:

```
            bm_bool barman_cc_interrupts_fiq_update_value(bm_uint64 timestamp, bm_uint32
core, bm_uint64 value);
            bm_bool barman_cc_interrupts_fiq_update_value_now(bm_uint64 value);
```

The second function is a shorthand for the first that passes the current timestamp and core number to the appropriate arguments.

When you call these functions, the value for the counter is stored to the capture.

13.4 Data storage

Barman uses a simple abstraction layer for handling the storage of collected data. Typically, the data that barman collects is stored in a RAM buffer on the target.

You can choose from the following data storage modes provided:

Linear RAM buffer mode
: Data collection stops when the buffer is full. This mode ensures that no collected data is lost, but no further data can be recorded.

Circular RAM buffer mode
: Data collection continues after the buffer is full and the oldest data is lost as it is overwritten by the newest data. This mode gives you control over when the data collection ends.

STM Interface
: System Trace Macrocell (STM) data is collected on a DSTREAM device that is connected to the target. You then dump the STM trace into a file, which you can import into Streamline for analysis.

For barman to be able to use either of the RAM buffer modes, you must first provide the RAM buffer on the target device. The RAM buffer is a dedicated, contiguous area of RAM that barman can write data to. On multiprocessor systems, the RAM buffer must be at the same address for all processors. It is your responsibility to allocate memory for the RAM buffer, either statically or dynamically.

13.5 Interfacing with barman

When barman is linked into your executable code, the code must call the following functions:

1. `barman_initialize` to initialize barman.
2. `barman_enable_sampling` to enable sampling.
3. The appropriate sample function, `barman_sample_counters` or `barman_sample_counters_with_program_counter`, to periodically collect data.

In a multiprocessor system, a call to one of the sampling functions only reads the counters for the processor element the code is currently executing on.

If you are running a preemptive kernel, RTOS, or similar, you must ensure that the thread running a call to a sampling function is not migrated from one processor element to another during the execution of the call.

In a multiprocessor system, if you are using periodic sampling (for example with a timer interrupt), you must provide a mechanism to call the sampling function for each processor element. In other words, to capture the counters of each processor element, there must be a timer interrupt or thread that is run separately on each processor element.

13.5.1 Configuration #defines

The following defines are configured by the configuration UI and stored in `barman.h`. They can be overridden at compile time as compiler parameters.

Define	Description
`BM_CONFIG_ENABLE_LOGGING`	Enables logging of messages when set to true.
`BM_CONFIG_ENABLE_DEBUG_LOGGING`	If `BM_CONFIG_ENABLE_LOGGING` is true, enables debug messages when set to true.
`BM_CONFIG_ENABLE_BUILTIN_MEMFUNCS`	Enables the use of built-in memory functions such as `__builtin_memset` and `__builtin_memcpy` when set to true.
`BM_CONFIG_MAX_CORES`	The maximum number of processor elements supported.
`BM_CONFIG_MAX_MMAP_LAYOUTS`	The maximum number of `mmap` layout entries to be stored in the data header. Configure to reflect the number of sections to be mapped for any process images.
`BM_CONFIG_MAX_TASK_INFOS`	The maximum number of distinct task entries that will be stored in the data. For single-threaded applications, this can be defined as zero to indicate that no information is provided. For multi-threaded applications or RTOS, this value indicates the maximum number of entries to store in the data header for describing processes, threads, and tasks.
`BM_CONFIG_MIN_SAMPLE_PERIOD`	The minimum period between samples in nanoseconds. If this value is greater than zero, calls to sampling functions are rate limited to ensure that there is a minimum interval of nanoseconds between samples.
`BM_CONFIG_USE_DATASTORE`	Specifies the data store to use. Valid values are `BM_CONFIG_USE_DATASTORE_LINEAR_RAM_BUFFER` and `BM_CONFIG_USE_DATASTORE_CIRCULAR_RAM_BUFFER`.
`BARMAN_DISABLED`	Disables the barman entry points at compile time when defined to a nonzero value. Use to conditionally disable calls to barman, for example in production code.

13.5.2 Annotation #defines

Color macros to use for annotations.

Define	Description
BM_ANNOTATE_COLOR_*color name*	Named annotation color, where *color name* is one of the following colors: RED 0x1bff0000 BLUE 0x1b0000ff GREEN 0x1b00ff00 PURPLE 0x1bff00ff YELLOW 0x1bffff00 CYAN 0x1b00ffff WHITE 0x1bffffff LTGRAY 0x1bbbbbbb DKGRAY 0x1b555555 BLACK 0x1b000000
BM_ANNOTATE_COLOR_CYCLIC	Annotation color that cycles through a predefined set.
BM_ANNOTATE_COLOR_RGB(*R*, *G*, *B*)	Create an annotation color from its components, where *R*, *G*, and *B* are defined as follows: *R* The red component, where $0 \leq R \leq 255$. *B* The blue component, where $0 \leq B \leq 255$. *G* The green component, where $0 \leq G \leq 255$.

13.5.3 Barman public API

Call the following public API functions to use the bare-metal agent.

barman_initialize

```
bm_bool barman_initialize(bm_uint8 * buffer,
    bm_uintptr buffer_length,
    const char * target_name,
    const struct bm_protocol_clock_info * clock_info,
#if BM_CONFIG_MAX_TASK_INFOS > 0
    bm_uint32 num_task_entries,
    const struct bm_protocol_task_info * task_entries,
#endif
#if BM_CONFIG_MAX_MMAP_LAYOUTS > 0
    bm_uint32 num_mmap_entries,
    const struct bm_protocol_mmap_layout * mmap_entries,
#endif
    bm_uint32 timer_sample_rate);
```

Description	Initialize barman.

Parameters	`buffer`
	Pointer to in memory buffer.
	`buffer_length`
	Length of the in memory buffer.
	`target_name`
	Name of the target device.
	`clock_info`
	Information about the monotonic clock used for timestamps.
	`num_task_entries`
	Length of the array of task entries in `task_entries`. If this value is greater than `BM_CONFIG_MAX_MMAP_LAYOUT`, it is truncated.
	`task_entries`
	The task information descriptors. Can be NULL.
	`num_mmap_entries`
	The length of the array of mmap entries in `mmap_entries`. If this value is greater than `BM_CONFIG_MAX_MMAP_LAYOUT`, it is truncated.
	`mmap_entries`
	The mmap image layout descriptors. Can be NULL.
	`timer_sample_rate`
	Timer based sampling rate in Hz. Zero indicates no timer based sampling (assumes max. 4GHz sample rate). This value is informative only, and is used for reporting the timer frequency in the Streamline UI.

Return value	`BM_TRUE`
	On success.
	`BM_FALSE`
	On failure.

Note

If `BM_CONFIG_MAX_TASK_INFOS` ≤ 0, `num_task_entries` and `task_entries` are not present.

If `BM_CONFIG_MAX_MMAP_LAYOUTS` ≤ 0, `num_mmap_entries` and `mmap_entries` are not present.

`barman_enable_sampling`

```
void barman_enable_sampling(void);
```

Description	Enables sampling. Call once all PMUs are enabled and the data store is configured.

`barman_disable_sampling`

```
void barman_disable_sampling(void);
```

Description	Disables sampling without reconfiguring the PMU. Sampling can be resumed by a call to `barman_enable_sampling`.

`barman_sample_counters`

```
void barman_sample_counters(bm_bool sample_return_address);
```

Description	Reads the configured PMU counters for the current processing element and inserts them into the data store. Can also insert a program counter record using the return address as the PC sample.
Parameter	`sample_return_address` `BM_TRUE` to sample the return address as PC, `BM_FALSE` to ignore.

Note

This function must be run on the processing element for the PMU that it intends to sample from, and it must not be migrated to another processing element for the duration of the call. This is necessary as it needs to program the per processing element PMU registers.

`barman_sample_counters_with_program_counter`

```
void barman_sample_counters_with_program_counter(const void * pc);
```

Description	Reads the configured PMU counters for the current processing element and inserts them into the data store.
Parameter	`pc` The PC value to record. The PC entry is not inserted if `pc == BM_NULL`.

Note

This function must be run on the processing element for the PMU that it intends to sample from, and it must not be migrated to another processing element for the duration of the call. This is necessary as it will need to program the per processing element PMU registers.

The following functions are available if `BM_CONFIG_MAX_TASK_INFOS > 0`:

`barman_add_task_record`

```
bm_bool barman_add_task_record(bm_uint64 timestamp, const struct
bm_protocol_task_info * task_entry);
```

Description	Adds a new task information record.

Parameters	`timestamp`
	The timestamp at which the record is inserted.
	`task_entry`
	The new task entry.

Return value	`BM_TRUE`
	On success.
	`BM_FALSE`
	On failure.

`barman_record_task_switch`

```
void barman_record_task_switch(enum bm_task_switch_reason reason);
```

Description	Records that a task switch has occurred. Call this function after the new task is made the current task, such that a call to `barman_ext_get_current_task_id` would return the new task ID. For example, insert it into the scheduler of an RTOS just after the new task is selected to record the task switch.
Parameter	`reason` Reason for the task switch.

Note

Call after the task switch has occurred so that `bm_ext_get_current_task` returns the `task_id` of the switched to task.

The following function is available if `BM_CONFIG_MAX_MMAP_LAYOUTS > 0`:

`barman_add_mmap_record`

```
bm_bool barman_add_mmap_record(bm_uint64 timestamp, const struct
bm_protocol_mmap_layout * mmap_entry);
```

Description	Adds a new `mmap` information record.

Parameters	`timestamp`
	The timestamp at which the record is inserted.
	`mmap_entry`
	The new `mmap` entry.

Return value	`BM_TRUE`
	On success.
	`BM_FALSE`
	On failure.

Data types associated with the public API functions:

`bm_protocol_clock_info`

```
struct bm_protocol_clock_info
{
    bm_uint64 timestamp_base;
    bm_uint64 timestamp_multiplier;
    bm_uint64 timestamp_divisor;
    bm_uint64 unix_base_ns;
};
```

Description	Defines information about the monotonic clock used in the trace. Timestamp information is stored in arbitrary units within samples. This reduces the overhead of making the trace by removing the need to transform the timestamp into nanoseconds at the point the sample is recorded. The host expects timestamps to be in nanoseconds. The arbitrary timestamp information is transformed to nanoseconds according to the following formula: `bm_uint64 nanoseconds = (((timestamp - timestamp_base) * timestamp_multiplier) / timestamp_divisor;` Therefore for a clock that already returns time in nanoseconds, `timestamp_multiplier` and `timestamp_divisor` should be configured as `1` and `1`. If the clock counts in microseconds then the multiplier and divisor should be set to `1000` and `1`. If the clock counts at a rate of *n* Hz, then the multiplier should be set to `1000000000` and the divisor to *n*.

Members	`timestamp_base`
	The base value of the timestamp such that this value is zero in the trace.
	`timestamp_multiplier`
	The clock rate ratio multiplier.
	`timestamp_divisor`
	The clock rate ratio divisor
	`unix_base_ns`
	The unix timestamp base value, in nanoseconds, such that a `timestamp_base` maps to a `unix_base` unix time value.

`bm_protocol_task_info`

```
struct bm_protocol_task_info
{
    bm_task_id_t task_id;
    const char * task_name;
};
```

Description	A task information record. Describes information about a unique task within the system.

Members	`task_id`
	The task ID.
	`task_name`
	The name of the task.

`bm_protocol_mmap_layout`

```
struct bm_protocol_mmap_layout
{
#if BM_CONFIG_MAX_TASK_INFOS > 0
    bm_task_id_t task_id;
#endif
    bm_uintptr base_address;
    bm_uintptr length;
    bm_uintptr image_offset;
    const char * image_name;
};
```

Description	An MMAP layout record. Describes the position of an executable image (or section thereof) in memory, allowing the host to map PC values to the appropriate executable image.

Members	`task_id`
	The task ID to associate with the map.
	`base_address`
	The base address of the image, or image section.
	`length`
	The length of the image, or image section.
	`image_offset`
	The image section offset.
	`image_name`
	The name of the image.

`bm_task_switch_reason`

```
enum bm_task_switch_reason
{
    BM_TASK_SWITCH_REASON_PREEMPTED = 0,
    BM_TASK_SWITCH_REASON_WAIT = 1
};
```

Description	Reason for a task switch.

Members	`BM_TASK_SWITCH_REASON_PREEMPTED`
	Thread is preempted.
	`BM_TASK_SWITCH_REASON_WAIT`
	Thread is blocked waiting, for example on IO.

WFI/WFE event handling functions:

`barman_wfi`

```
void barman_wfi(void);
```

Description	Wraps WFI instruction and sends events before and after the WFI to log the time in WFI. This function is safe to use in place of the usual WFI `asm` instruction, as it degenerates to just a WFI instruction when barman is disabled.

`barman_wfe`

```
void barman_wfe(void);
```

Description	Wraps WFE instruction and sends events before and after the WFE to log the time in WFE. This function is safe to use in place of the usual WFE `asm` instruction as it degenerates to just a WFE instruction when barman is disabled.

`barman_before_idle`

```
void barman_before_idle(void);
```

Description	Call before a WFI/WFE, or other similar halting event, to log entry into the paused state. Can be used in situations where `barman_wfi()`/`barman_wfe()` is not suitable.

Note

- You must use `barman_before_idle` in a pair with `barman_after_idle()`.
- Using `barman_wfi()`/`barman_wfe()` is preferred in most cases, as it takes care of calling the before and after functions.

`barman_after_idle`

```
void barman_after_idle(void);
```

Description	Call after a WFI/WFE, or other similar halting event, to log exit from the paused state. Can be used in situations where `barman_wfi()`/`barman_wfe()` is not suitable.

Note

- You must use `barman_after_idle` in a pair with `barman_before_idle()`.
- Using `barman_wfi()`/`barman_wfe()` is preferred in most cases, as it takes care of calling the before and after functions.

Functions for recording textual annotations:

`barman_annotate_channel`

```
void barman_annotate_channel(bm_uint32 channel, bm_uint32 color, const char * string)
```

Description	Adds a string annotation with a display color, and assigns it to a channel.

Parameters	`channel`
	The channel number.
	`color`
	The annotation color from `bm_annotation_colors`.
	`text`
	The annotation text, or null to end the previous annotation.

Note

Annotation channels and groups are used to organize annotations within the threads and processes section of the **Timeline** view. Each annotation channel appears in its own row under the thread. Channels can also be grouped and displayed under a group name, using the `barman_annotate_name_group` function.

`barman_annotate_name_channel`

```
void barman_annotate_name_channel(bm_uint32 channel, bm_uint32 group, const char *
name)
```

Description	Defines a channel and attaches it to an existing group.

Parameters	`channel`
	The channel number.
	`group`
	The group number.
	`name`
	The name of the channel.

Note

The channel number must be unique within the task.

`barman_annotate_name_group`

```
void barman_annotate_name_group(bm_uint32 group, const char * name)
```

Description	Defines an annotation group.

Parameters	`group`
	The group number.
	`name`
	The name of the group.

Note

The group identifier, `group`, must be unique within the task.

`barman_annotate_marker`

```
void barman_annotate_marker(bm_uint32 color, const char * text)
```

Description	Adds a bookmark with a string and a color to the **Timeline** and **Log** views. The string is displayed in the **Timeline** view when you hover over the bookmark, and in the **Message** column in the **Log** view.

Parameters	`color`
	The marker color from `bm_annotation_colors`.
	`text`
	The marker text, or null for no text.

`bm_annotation_colors`

Description	Color macros for annotations. See *13.5.2 Annotation #defines* on page 13-189.

13.5.4 External functions to implement

You must provide the following external functions.

`barman_ext_get_timestamp`

```
extern bm_uint64 barman_ext_get_timestamp(void);
```

Description	Reads the current sample timestamp value, which must be provided for the time at the point of the call. The timer must provide monotonically incrementing values from an implementation defined start point. The counter must not overflow during the period that it is used. The counter is in arbitrary units. The mechanism for converting those units to nanoseconds is described as part of the protocol data header.
Return value	The timestamp value in arbitrary units.

The following functions have weak linkage implementations that can be overridden if necessary:

`barman_ext_disable_interrupts_local`

```
extern bm_uintptr barman_ext_disable_interrupts_local(void);
```

Description	Disables interrupts on the local processor only. Used to allow atomic accesses to certain resources, for example PMU counters.
Return value	The current interrupt enablement status value. This value must be preserved and passed to `barman_ext_enable_interrupts_local` to restore the previous state.

Note

A weak implementation of this function is provided that modifies DAIF on AArch64, or CPSR on AArch32.

`barman_ext_enable_interrupts_local`

```
extern void barman_ext_enable_interrupts_local(bm_uintptr previous_state);
```

Description	Enables interrupts on the local processor only.
Parameter	`previous_state` The value that was previously returned from `barman_ext_disable_interrupts_local`

Note

A weak implementation of this function is provided that modifies DAIF on AArch64, or CPSR on AArch32.

The following functions must be defined if `BM_CONFIG_MAX_CORES > 1`:

`barman_ext_map_multiprocessor_affinity_to_core_no`

```
extern bm_uint32 barman_ext_map_multiprocessor_affinity_to_core_no(bm_uintptr mpidr);
```

Description	Given the MPIDR register, returns a unique processor number. The implementation must return a value between 0 and *N*, where *N* is the maximum number of processors in the system. For any valid permutation of the arguments, a unique value must be returned. This value must not change between successive calls to this function for the same argument values. `// // Example implementation where processors are arranged as follows: // // aff2 | aff1 | aff0 | cpuno // -----+------+------+------ // 0 | 0 | 0 | 0 // 0 | 0 | 1 | 1 // 0 | 0 | 2 | 2 // 0 | 0 | 3 | 3 // 0 | 1 | 0 | 4 // 0 | 1 | 1 | 5 // bm_uint32 barman_ext_map_multiprocessor_affinity_to_core_no(bm_uintptr mpidr) { return (mpidr & 0x03) + ((mpidr >> 6) & 0x4); }`
Parameter	`mpidr` The value of the MPIDR register.
Return value	The processor number.

Note

This function need only be defined when `BM_CONFIG_MAX_CORES > 1`

`barman_ext_map_multiprocessor_affinity_to_cluster_no`

```
extern bm_uint32 barman_ext_map_multiprocessor_affinity_to_cluster_no(bm_uintptr
mpidr);
```

Description	Given the MPIDR register, return the appropriate cluster number. Cluster IDs should be numbered from 0 to *N*, where *N* is the number of clusters in the system.
	`//`
	`// Example implementation which is compatible with the example implementation of`
	`// barman_ext_map_multiprocessor_affinity_to_core_no given above.`
	`//`
	`bm_uint32 barman_ext_map_multiprocessor_affinity_to_cluster_no(bm_uintptr mpidr)`
	`{`
	`    return ((mpidr >> 8) & 0x1);`
	`}`

Parameter	`mpidr` The value of the MPIDR register.
Return value	The cluster number.

Note

This function need only be defined when `BM_CONFIG_MAX_CORES > 1`

The following function must be defined if `BM_CONFIG_MAX_TASK_INFOS > 0`:

`barman_ext_get_current_task_id`

```
extern bm_task_id_t barman_ext_get_current_task_id(void);
```

Description	Returns the current task ID.

The following functions must be defined if `BM_CONFIG_ENABLE_LOGGING != 0`:

`barman_ext_log_info`

```
void barman_ext_log_info(const char * message, ...);
```

Description	Prints an info message.
Parameter	`message`

`barman_ext_log_warning`

```
void barman_ext_log_warning(const char * message, ...);
```

Description	Prints a warning message.
Parameter	`message`

`barman_ext_log_error`

```
void barman_ext_log_error(const char * message, ...);
```

Description	Prints an error message.
Parameter	`message`

The following function must be defined if `BM_CONFIG_ENABLE_DEBUG_LOGGING != 0`:

`barman_ext_log_debug`

```
void barman_ext_log_debug(const char * message, ...);
```

Description	Prints a debug message.
Parameter	`message`

13.6 Extracting and importing data

You must extract the data from the RAM buffer when the capture is complete.

For example, you could choose to do one of the following:

- Save the data to the file system of the target device, if one exists.
- Retrieve the data from RAM using JTAG during a debug session.
- Transfer the data over one of the available communication interfaces, for example Ethernet or USB.

After extracting the raw data, give the data file a `.raw` extension. You can import this file into Streamline by clicking **Import Capture File(s)…**. The imported data is then available for Streamline to analyze.

If you added a custom pmus.xml or events.xml file during the configuration and generation stage, you must provide a copy of the same file into the .apc directory that is created for the imported capture. The files must be named pmus.xml and events.xml and must be placed in the directory alongside the barman.raw file for them to be detected and used.

Chapter 14
Profiling with System Trace Macrocell

Describes the collection of profiling data using System Trace Macrocell (STM).

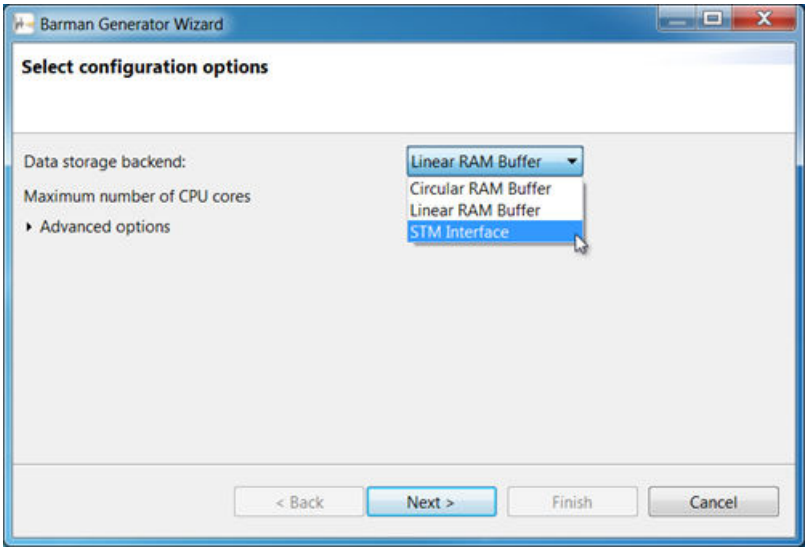
Further information about STM, including the Technical Reference Manual, can be found on *ARM Developer*.

It contains the following sections:

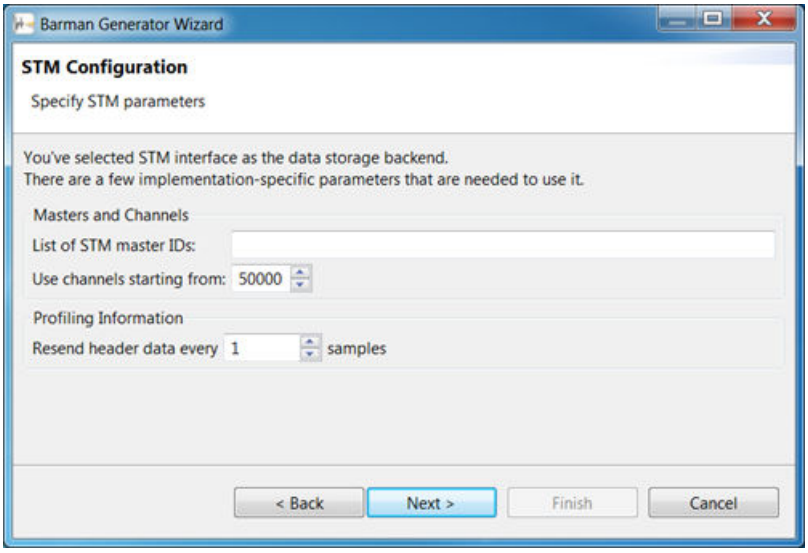
14.1 STM workflow

The workflow for STM involves a complex series of interactions between the applications involved.

1. Generate barman agent code for STM using the **Barman Generator Wizard** dialog in Streamline.
   a. Select **STM Interface** as the data storage backend.



   b. Specify the STM parameters for your project.



   c. Complete the remainder of the wizard as for a standard bare-metal project.
2. Add the barman agent files that the wizard generates to your project.
3. Instrument your bare-metal application code with barman agent calls (initialization, periodic sampling).
4. Compile and link your project.
5. Connect your target to a DSTREAM device.
6. Configure your target for collecting STM data into its RAM buffer.
7. Run the application on a target.
8. When you want to end the profiling, stop the application.
9. Dump the STM trace from the DSTREAM device to a file.
10. Let Streamline import the trace file dump. Streamline reformats it and prepares it for analysis.

Note

- If you are using DS-5, you can dump the STM trace into a file using the following command:

```
trace dump <filename> STM
```

- If you do not launch your bare-metal application from within DS-5, you must handle connecting to DSTREAM, obtaining the trace file, and importing it into Streamline.

Related tasks

13.2 Configuring barman on page 13-182.

14.2 STM channels

Data reaches the STM stimulus ports through channels.

Barman reserves the channels following the channel number that you specify in the **Barman Generator Wizard**. The number of channels is equal to the number of processing units on the target system. Each processing unit then has access to its own independent stimulus port. Having separate channels removes the need for a synchronization mechanism and keeps the probe effect of the barman agent small.

In the simplest scenario, a single processing unit uses a single channel. The data that STM outputs matches the data that was written to the STM stimulus port. It is a stream of bytes directly representing a sequence of sample records.

When multiple processing units write to STM stimulus ports simultaneously, STM performs synchronized writes to the output stream. The input writes performed by the separate processing units are interleaved and associated with channel indicator packets.

Before Streamline can analyze the STM trace, the raw data must be processed to reconstruct the original samples from the interleaved fragments. Any incomplete samples are discarded, and the complete samples are then joined together, with flags marking the beginning and end of each sample.

14.3 Importing an STM trace

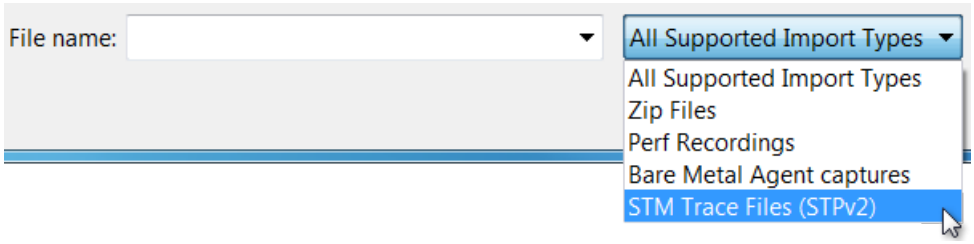
Import STM trace files into Streamline for analysis.

Note

You can also import the STM trace from the command-line.

Procedure

1. Click **Import Capture File(s)…** in the **Streamline Data** view.
2. Select the import file type **STM Trace Files (STPv2)**.



3. Select the trace file to import.
4. Click **Open** and a new dialog box opens.
5. Enter the location of the `barman.xml` file that the **Barman Generator Wizard** produced.

   This file contains information about how to find relevant data in the trace file. For example, the channel numbers used.
6. Click **OK**.

Streamline then reformats the data, and converts the STM trace file into a barman agent raw file.

Related references

16.6 Importing an STM trace from the command-line on page 16-223.

Chapter 15
Troubleshooting Common Streamline Issues

Describes how to troubleshoot some common Streamline issues.

It contains the following sections:

15.1 Troubleshooting target connection issues

You might have problems when trying to start a capture session, for instance by pressing the **Start capture** button. Use these solutions to solve common target connection issues.

Problem	Solution

Error message generated:
`Unable to connect to the gator daemon at` *`target_address`*`.`
`Please verify that the target is reachable and that you are running gator daemon v17 or later. Installation instructions can be found in: streamline/gator/README.md.`
`If connecting over WiFi, please try again or use a wired connection.`
	Make sure `gatord` is running on your target. Enter the following command in the shell of your target:
	`ps ax | 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 &`
	Try connecting to the target again.
	If `gatord` is active and you still receive this error message, try disabling any firewalls on your host machine that might be interfering with communication between it and the target.
	In addition, if you are running Android on your target, make sure the ports are accessible by using the `adb forward` command, for example:
	`adb forward tcp:8080 tcp:8080`

Error message generated: `Unknown host`	Make sure that you have correctly entered the name or IP address of the target in **Address** field. If you have entered a name, try an IP address instead.
When using event-based sampling, Streamline fails to find the PMU.	The PMU on your hardware might not be correctly configured to allow the processor interrupts necessary for Streamline to use event-based sampling. Test on alternate hardware or disable event-based sampling in the **Counter Configuration** dialog box.

(continued)

Problem	Solution

The target is running a firewall, which prevents Streamline from connecting to `gatord`.	There are several possible ways to resolve this issue:
	• Update the firewall to allow connections to `gatord`, which defaults to using port 8080.
	• Use local captures.
	• If the target accepts SSH connections, you can establish an SSH tunnel by using the `ssh` command on the host. For example:
	`ssh user@target -L 8080:localhost:8080 -N`
	In this example, replace *`user`* with the username to log in as and *`target`* with the hostname of the target. On the target, use `localhost` as the hostname.
	Note
	An SSH tunnel requires additional processing on the target.
	• Reverse SSH tunnels are also possible by running `ssh` from the target to the host. For example:
	`ssh user@host -R 8080:localhost:8080 -N`

Delay between clicking the **Capture** button and the **Live** view displaying data.	Use the `gator.ko` kernel module.
	Setting `ftrace` to use a different clock can also help. Use one of the following commands:
	Linux kernel 4.2 or later
	`echo mono_raw > /sys/kernel/debug/tracing/trace_clock`
	Older Linux kernel versions
	`echo perf > /sys/kernel/debug/tracing/trace_clock`

Related tasks

12.1 Capturing data on your target without an Ethernet connection on page 12-164.

Related references

15.5 Troubleshooting report issues on page 15-214.

15.2 Troubleshooting perf import issues

Consult the following tables for potential error messages and warnings related to importing perf data.

Problem	Solution
Error message generated: `Big-endian traces are currently not supported.`	Only import trace files that are made on little-endian targets.

Error message generated:
`Unable to decode individual event ids from the trace.`
`An unexpected error occured.`
`Duplicate event ids found.`
`Invalid cluster map data.`
`Sample found with id that does not map to any known event.`
`Unexpected read_format value.`
	Raise a support issue with ARM and provide an example of the perf recording that triggered the error message.

Error message generated: `File not recognized.` The input file was not a recognized perf recording, or was created with an unsupported version of the tool.	Ensure that the file is in the correct perf format. Update your version of the perf tool if necessary.
Error message generated: `Missing timestamp information. Make sure perf record has -T flag set.` The file does not contain timestamps in events.	Add the `-T` argument to `perf record`.

Error message generated:
`Missing HEADER_EVENT_DESC section.`
`Missing HEADER_PMU_MAPPINGS section.`
`Missing HEADER_TRACING_DATA section.`
`"sample_id_all" flag is not set on at least one attribute.`
The file does not contain sufficient information for it to be successfully converted.
	Upgrade your version of the perf tool.

Error message generated: `A recorded event is missing PERF_SAMPLE_READ sample_type, or PERF_SAMPLE_PERIOD is missing.` An event that is not counted was configured.	Add the `S` modifier, specify a period greater than zero, or use frequency sampling alongside recording the sample period. Use the `-F` argument for frequency sampling and the `-P` argument for the sample period.
Error message generated: `The input file does not contain any sample records for any recognized counter.`	Ensure the recording contains at least one sample for a counter that Streamline recognizes.

Warning	Explanation
Warning in **Timeline** view: `Some counters did not have samples associated with them and are excluded.`	Some of the recorded counters do not have recorded samples, and therefore do not appear in Streamline. The excluded counters are listed.
Warning in **Timeline** view: `Missing cluster map.`	Streamline attempts to work out the cluster configuration based on which processor, or subset of processors, each recorded event is associated with. Streamline also attempts to assign appropriate labels, such as A53 and A57, to these clusters. This warning means that it is not possible to know from the recorded data what the clusters on the device were. As `perf record` does not explicitly store the names of clusters, heuristics are used to work them out. However, this method can often fail, which means that processor PMU counters are not separated into different clusters. Therefore counter values appear as though there is no clustering.

Related concepts

3.3 Importing perf data on page 3-49.

15.3 Troubleshooting MGD Mode issues

If you encounter a problem running MGD, consult the following list of possible errors. These errors are listed in the order in which Streamline detects them.

Problem	Solution
Error message generated: `No MGD Daemon running on target device.`	Launch `mgddaemon` on the target device.
Error message generated: `Timed out connecting to MGD Daemon. Perhaps MGD is already connected.` Streamline detected that `mgddaemon` is already busy tracing and cannot perform a new tracing request.	Stop any ongoing tracing operations.
Error message generated: `Unsupported version of MGD Daemon is running.`	Install a newer version of `mgddaemon` which supports the MGD mode feature.
Error message generated: `No MGD executable found. Please check your Capture and Analysis Options dialog.`	Open the **Capture & Analysis Options** dialog and set the MGD installation directory location in the MGD Mode section.
Error message generated: `Unsupported version of installed MGD application found. Version x.x.x found, but at least version 4.0.0 required.`	Upgrade to MGD version 4.0.0.
Error message generated: `Mali Graphics Debugger x.x.x reported an invalid protocol version number. Please make sure the version of MGD you are using is compatible with this version of Streamline.`	Upgrade to MGD version 4.0.0.
Error message generated: `Unable to detect MGD application version. Please ensure the executable path is valid.`	Ensure the path in the **Capture & Analysis Options** dialog points to a valid installation of MGD.

(continued)

Problem	Solution
Error message generated: `An error occurred when attempting to detect MGD Daemon version.`	Ensure that the connection with the target device is working correctly, and that the correct version of `mgddaemon` is installed and running on the device. Also ensure that `mgddaemon` is contactable through port 5002 and is not, for example, blocked by a firewall.
Error message generated: `Could not determine hostname for target device, or 'adb forward' failed.` If Streamline cannot determine the hostname for the target device, it is not possible for it to work out the IP address where `mgddaemon` is running. Alternatively for adb targets, the command `adb forward` failed and Streamline was not able to forward the `mgddaemon` port so that it could communicate with `mgddaemon`.	For adb connections, ensure that the adb executable is configured in the **Capture & Analysis Options** dialog. Also ensure that there are no conflicting port forwarding rules which are already configured with adb. For non-adb connections, treat this error as a bug.

Related concepts

*6.7 Mali Graphics Debugger (MGD) Mode in **Live** view* on page 6-106.

Related references

4.1 Capture & Analysis Options dialog box settings on page 4-61.

15.4 Troubleshooting Energy Probe issues

If you encounter a problem using the Energy Probe, consult the following list of potential problems and error messages.

Problem	Solution
Difficulty running `caiman` from Streamline.	Run `caiman` on the command line in local mode. For example: `/usr/local/DS-5/bin/caiman -l -r 0:20` Information to assist with debugging is displayed. If no messages are printed after a few seconds, you can kill `caiman`. If there are no problems, the `0000000000` file is non-empty.
Error message generated: `Unable to detect the energy probe`	Disconnect and reconnect the Energy Probe, and ensure the Energy Probe properly enumerates with the OS. This problem can occur on some operating systems that do not properly re-enumerate the Energy Probe device after rebooting, or going into sleep or hibernate.
Error message generated: `Unable to set `/dev/ttyACM0` to raw mode, please verify the device exists`	Check the permissions of `/dev/ttyACM0`. You may need to add your username to a group or modify the permissions of `/dev/ttyACM0`. If the problem persists, a different program may be using the device. Use `lsof` to debug further. On Ubuntu, after plugging in the Energy Probe, the modem-manager opens the device for a while.
`/dev/ttyACM0` does not exist after plugging in the energy probe.	Upgrade to Linux version 2.6.36 or later.
Error message generated: `bash: ./caiman: No such file or directory`	Ensure `/lib/ld-linux.so.2` is installed. It is in the `libc6-i386` package on Ubuntu. This problem can occur when running the 32-bit version of `caiman` on `x86_64`.
The power values in Streamline look incorrect.	Ensure that the shunt resistor values are correct in the **Energy Capture** section of the **Capture & Analysis Options** dialog box.
Missing channels.	Ensure either power, voltage, or current is checked in the **Energy Capture** section of the **Capture & Analysis Options** dialog box.
Data is reporting zero or close to zero values.	Check the connections. If you are using the Energy Probe, ensure that the green LED is on.
Error message generated: `caiman-src/EnergyProbe.cpp:27:21: error: libudev.h: No such file or directory`	Your platform does not support `udev` or is missing the `udev` headers. Either install `libudev-dev`, or equivalent, or disable `udev` support in `CMakeLists.txt` by setting `SUPPORT_UDEV` to `0`.

15.5 Troubleshooting report issues

If you successfully complete a capture session but have a problem with the resulting report data, consult this list of common issues.

Problem	Solution
Streamline does not show any source code in the **Code** view.	Make sure that you use the `-g` option during compilation. Streamline must have debug symbols turned on in order to match instructions to source code. If necessary, ensure that the path prefix substitutions are set correctly. See the **Path Prefix Substitutions** dialog box in the **Code** view.
Streamline does not show source code for shared libraries.	Add the libraries using the **Capture & Analysis Options** dialog box. Click **Add ELF Image...** in the Program Images section, navigate to your shared library, and then add it.
The data in the **Call Paths** view is flat. The presented table is a list rather than a hierarchy.	Use the GCC options `-fno-omit-frame-pointer` and `-marm` during compilation. `-marm` is only required for ARMv7 and earlier architectures. Also, check the **Call Stack Unwinding** option in the **Capture & Analysis Options** dialog box. If frame pointers are being set correctly, the disassembly of the code should contain instructions in the form `add fp,sp,#<n>` at the start of functions. To generate a disassembly, use the following command: `arm-linux-gnueabihf-objdump -d foo.so` **Note** • By default, Streamline does not walk the stack for kernels or loadable kernel modules. These generate flat data in the **Call Paths** view unless you built `gator.ko` to perform kernel stack unwinding. • Streamline does not walk the stack for statically-linked drivers. • User space gator does not support call stack unwinding.
Functions that you know are highly used are missing from the reports. Other functions might seem artificially large.	This can be because of code inlining done by the compiler. To turn inlining off, add `-fno-inline` as an option during compilation.
A newly-generated capture has no data.	If you experience this and the profiling session had event-based sampling enabled, the PMU on your target might not have triggered the interrupts correctly. Test on alternate hardware or disable event-based sampling in the **Counter Configuration** dialog box.

Related tasks

12.11 Profiling the Linux kernel on page 12-176.
*8.3 Path prefix substitution in the **Code** view* on page 8-129.

Related references

15.1 Troubleshooting target connection issues on page 15-207.

Chapter 16
Using Streamline on the Command Line

Describes how to use the `streamline` command to access much of the functionality of Streamline from the command line.

It contains the following sections:

16.1 Opening a Streamline-enabled command prompt or shell

Using Streamline outside of the user interface enables you to perform automated captures and generate text-based reports that you can export into a spreadsheet application.

The way you open a Streamline-enabled command prompt or shell depends on your operating system:

- On Windows, select **Start** > **All Programs** > **ARM DS-5** > **DS-5 Command Prompt** .
- On Linux, add the *DS-5_install_directory*`/bin` location to your `PATH` environment variable then open a UNIX bash shell.

Related tasks

16.3 Outputting command-line data to a file on page 16-220.

16.4 Exporting the Heat Map from the command-line on page 16-221.

Related references

16.2 Streamline command-line options on page 16-217.

16.2 Streamline command-line options

The `streamline` command has different modes that enable you to use most features of Streamline outside of the graphical user interface.

Streamline command-line modes

Use the `streamline` command in a Streamline-enabled shell with the following syntax:

`streamline <mode> [options] <file...>`

Use either of the following options directly after `streamline` on the command line:

Mode	Description
`-capture`	This mode initiates a capture session. You must specify a valid `session.xml` file with this option. The `session.xml` file defines your target hardware and the parameters of the capture session. For example: `streamline -capture session.xml` To create a `session.xml` file, enter your settings and then use the **Export...** option in the **Capture & Analysis Options** dialog box.
`-report`	This mode reads data from a capture and outputs it to your console or to a file. You can use additional options to filter and format the data. You must enter a valid `.apc` capture after the declaration of report mode. For example: `streamline -report threads_001.apc`

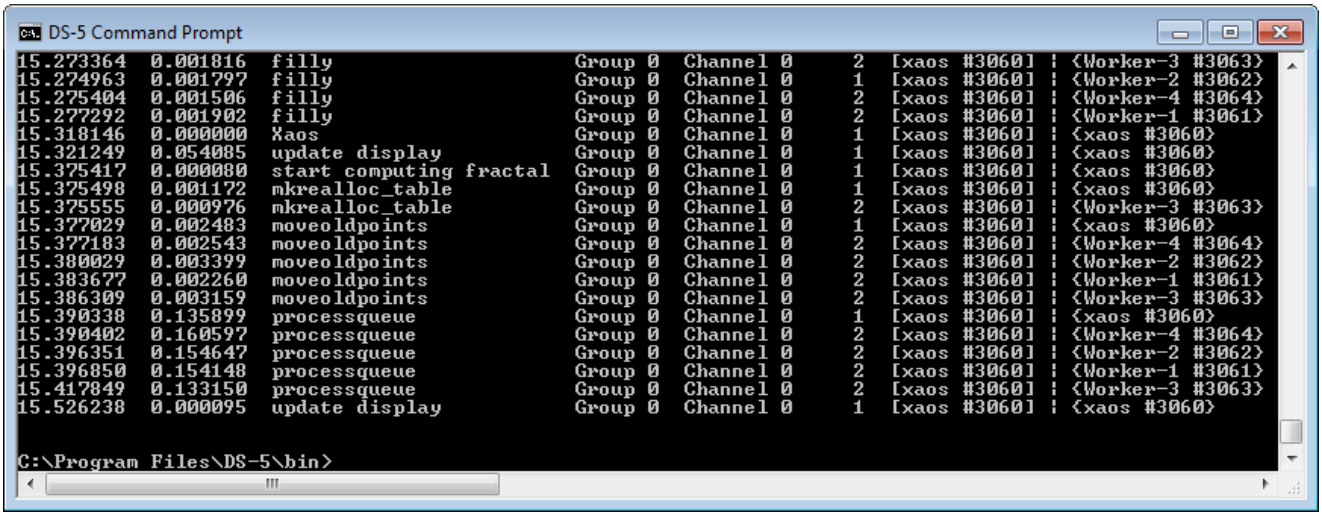


Figure 16-1 Functions report generated using report mode

Options common to both modes

The following options are available in both modes:

Option	Description
`-h, -?, -help`	Outputs help information to the console, listing each mode and option as well as the required syntax.
`-v, -version`	Displays the program version information.
`-o, -output <filename>`	In capture mode, defines the name of the capture file to create. In report mode, defines the name of the file to which to send the output. If you do not specify this option, output is sent to the console.

Capture mode options

The following options are available only in capture mode:

Option	Description
`-duration` <*`seconds`*>	Sets the number of seconds that you want the capture to last. The capture automatically terminates when it reaches that duration. For example, the following command triggers a 60 second capture session: `streamline -capture -duration 60 session.xml`
`-c, -config <configuration.xml>`	Defines the location of the configuration file to send to the target.
`-retrieve-image` <*`regex`*>	Specifies a regex that is used to match the processes to retrieve from the target.
`-username` <*`string`*>	Specifies the username to use when retrieving images from the target.
`-password` <*`string`*>	Specifies the password to use when retrieving images from the target.
`-include-libs`	Fetches the relevant libraries when retrieving images from the target.
`-p, -pmus <pmus.xml>`	Specify the path to your `pmus.xml` file.
`-e, -events <events.xml>`	Specify the path to your `events.xml` file.

Report mode options

The following options are unique to report mode:

Option	Description
`-all`	Outputs the contents of the **Timeline**, **Call Paths**, **Functions**, and **Log** views, including bookmarks and custom activity map tables. This is the default option.
`-callpath`	Outputs the contents of the table data of the **Call Paths** view. Subordinate functions are indented.
`-function`	Outputs the contents of the **Functions** view.
`-log`	Outputs the contents of the **Log** view.
`-timeline`	Outputs the contents of the **Timeline** view.
`-cam`	Outputs the contents of custom activity map tables. **Note** To export custom activity map data, the report must contain custom activity map annotations.
`-heatmap`	Outputs the contents of the **Heat Map**.

(continued)

Option	Description
`-opencl`	Outputs the contents of the OpenCL table. **Note** To export OpenCL data, the report must contain OpenCL annotations.
`-bookmarks`	Outputs all bookmarks stored in a capture, listing the time index location of the bookmark and its text.
`-process <regex>#<PID>`	A regex matching the process for which to filter the timeline data and its PID, separated by a hash
`-template <filename>`	Use this option with a valid chart configuration template to list data for your customized charts on the command line. Use the **Switch and manage templates** button in the **Live** or **Timeline** view to create a chart configuration template, then use that file with the `-template` option.
`-per_core`	Outputs per-core data when used with the `-timeline` option.
`-scale <number>`	Sets the number of bins per second to use in the report, by default 1000.
`-start <seconds>`	Filters output data to start at the specified time within the timeline. For example if you enter 0.005 with this option, all data before the 5 millisecond mark is not included in the output.
`-stop <seconds>`	Filters output data to stop at the specified time within the timeline.
`-bstart <name>`	Filter the data to start at the first bookmark with the provided name. For example, if you enter `-bstart redflag`, all data before the first instance of the bookmark title `redflag` is filtered from the output.
`-bstop <name>`	Filter the data to end at the first bookmark with the provided name.
`-format <space|tab|csv>`	Specifies the format of the output. Format the tables using spaces, tabs or comma-separated values. This can be useful if you want to easily convert output text files to your favorite spreadsheet program. The default format is spaces, making the tables easy to read when printed on the command line.

You can define multiple report types using these options. For example, to output the **Call Paths and Functions** data from the `thread_001.apc` capture, enter:

```
streamline -report -callpath -function thread_001.apc
```

Related tasks

16.1 Opening a Streamline-enabled command prompt or shell on page 16-216.
16.3 Outputting command-line data to a file on page 16-220.
16.4 Exporting the Heat Map from the command-line on page 16-221.

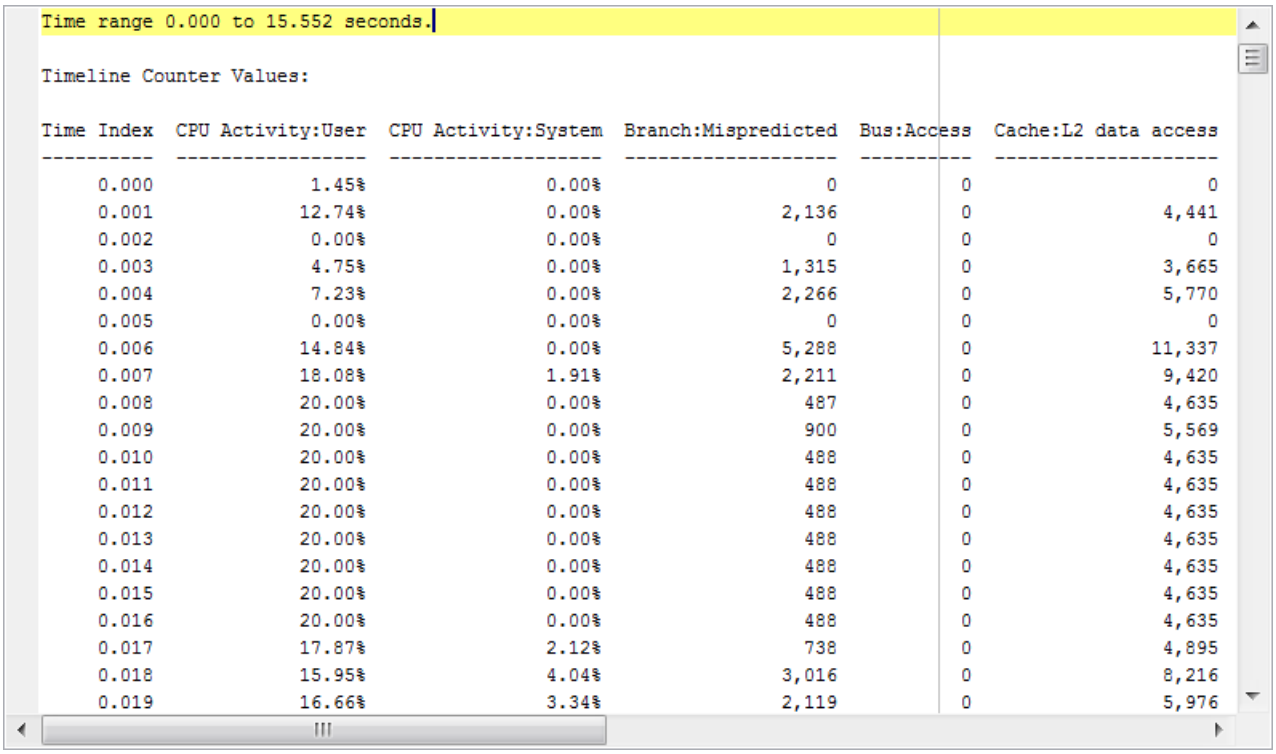
16.3 Outputting command-line data to a file

Because the reports generated by Streamline can be very large, it can be useful to send the output to a file. You can then import data from this file to your favorite spreadsheet application.

Specify the output file to the `streamline` command using the `-o` option, for example:

```
streamline -report -timeline capture_001.apc -o output.txt
```

This creates a file with the given name, if it does not already exist, and outputs the data to the new file instead of to the command window. If the file already exists, it is overwritten.



```
Time range 0.000 to 15.552 seconds.

Timeline Counter Values:

Time Index  CPU Activity:User  CPU Activity:System  Branch:Mispredicted  Bus:Access  Cache:L2 data access
----------  -----------------  -------------------  -------------------  ----------  --------------------
     0.000              1.45%                0.00%                    0           0                     0
     0.001             12.74%                0.00%                2,136           0                 4,441
     0.002              0.00%                0.00%                    0           0                     0
     0.003              4.75%                0.00%                1,315           0                 3,665
     0.004              7.23%                0.00%                2,266           0                 5,770
     0.005              0.00%                0.00%                    0           0                     0
     0.006             14.84%                0.00%                5,288           0                11,337
     0.007             18.08%                1.91%                2,211           0                 9,420
     0.008             20.00%                0.00%                  487           0                 4,635
     0.009             20.00%                0.00%                  900           0                 5,569
     0.010             20.00%                0.00%                  488           0                 4,635
     0.011             20.00%                0.00%                  488           0                 4,635
     0.012             20.00%                0.00%                  488           0                 4,635
     0.013             20.00%                0.00%                  488           0                 4,635
     0.014             20.00%                0.00%                  488           0                 4,635
     0.015             20.00%                0.00%                  488           0                 4,635
     0.016             20.00%                0.00%                  488           0                 4,635
     0.017             17.87%                2.12%                  738           0                 4,895
     0.018             15.95%                4.04%                3,016           0                 8,216
     0.019             16.66%                3.34%                2,119           0                 5,976
```

Figure 16-2 Timeline view data output to a text file

Related tasks

16.1 Opening a Streamline-enabled command prompt or shell on page 16-216.

16.4 Exporting the Heat Map from the command-line on page 16-221.

Related references

16.2 Streamline command-line options on page 16-217.

16.4 Exporting the Heat Map from the command-line

Export the data that is contained in the **Heat Map** mode of the details panel to a text file.

Enter the following at the command-line:

```
./streamline -report <apc-name> -heatmap "<source>"
```

`<source>` is the activity source that is used in the analysis report. This parameter is optional. If you do not provide the activity source, **CPU Activity** is exported by default.

The following options are also available:

- `-o, -output <filename>`
- `-process <regex>#<PID>`
- `-scale <number>`
- `-start <seconds>`
- `-stop <seconds>`
- `-bstart <name>`
- `-bstop <name>`
- `-format <space|tab|csv>`

Note

The **Heat Map** data is not included in `-all`, and therefore not exported when executing the following command:

```
./streamline -report  -all
```

Related concepts

6.6.2 Heat Map mode on page 6-97.
6.6.5 Selecting the activity source on page 6-100.

Related tasks

16.1 Opening a Streamline-enabled command prompt or shell on page 16-216.
16.3 Outputting command-line data to a file on page 16-220.

Related references

16.2 Streamline command-line options on page 16-217.
16.2 Streamline command-line options on page 16-217.

16.5 Accessing the Bare-Metal generation mechanism from the command line

You can pass the configured, and optionally modified, XML file produced in the Bare-Metal configuration process to the command line. The generator then outputs the source and header files.

Enter `streamline -generate-bare-metal-agent <options>`

The following command-line arguments are available:

`-c, -config <config.xml>`
: The configuration file to use to generate the bare-metal agent.

`-p, -pmus <pmus.xml>`
: Specify the path to your `pmus.xml` file.

`-e, -events <events.xml>`
: Specify the path to your `events.xml` file.

`-o, -output <output_path>`
: Specify the output path to where the generated files will be written.

Related tasks

13.2 Configuring barman on page 13-182.

16.6 Importing an STM trace from the command-line

Import an STM trace file and create a capture from it.

Enter the following command:

```
./streamline <arguments>
```

The arguments that are required are:

`--import-stm`

`--trace <path_to_stm_trace_dump_file>`
: Provide the path to the trace file that was output by STM.

`--barman-xml <path_to_generated_barman_xml_file>`
: Provide the path to the `barman.xml` file that was generated by the **Barman Generator Wizard**.

Streamline then reformats the data, and converts the STM trace file into a barman agent raw file.

Related references

Chapter 14 Profiling with System Trace Macrocell on page 14-201.

Glossary

This glossary defines some of the terms that are used in the ARM DS-5 Streamline User Guide.

Backtrace

A backtrace is a snapshot of a thread's call stack at a specific moment in time.

Call stack unwinding

See *Backtrace* on page 224.

Counter

A counter counts how often a particular event occurs. Counters are read on a schedule switch and on a sample.

Event-based sampling (EBS)

See *Sample* on page 224.

Sample

A sample is when the running thread is interrupted and a backtrace is collected. When event-based sampling (EBS) is not enabled, a sample is collected a fixed number of times per second as specified by the sample rate. When EBS is enabled, a sample should be taken when the count of the counter is a multiple of the count specified.

See also

Backtrace on page 224.