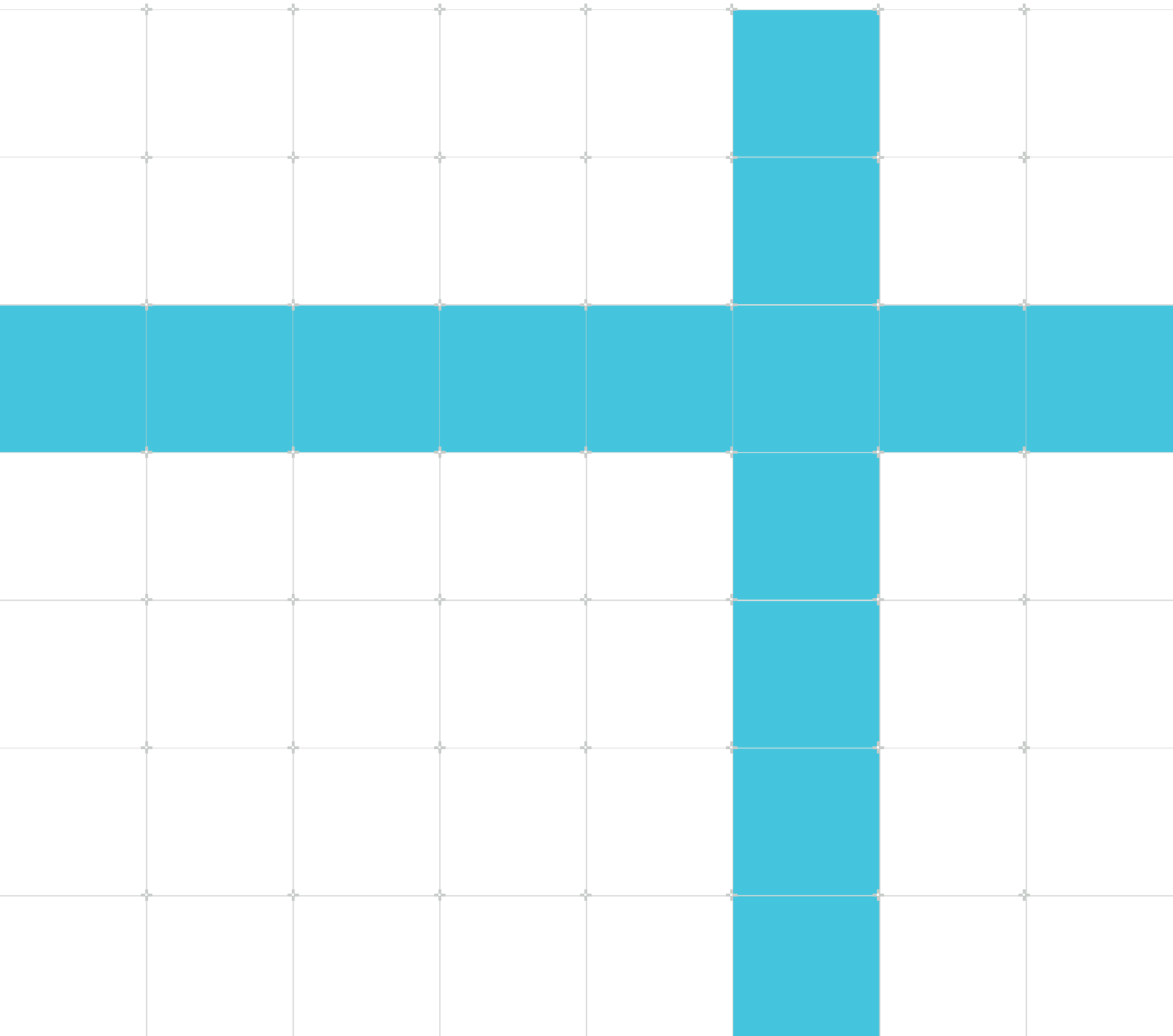
Arm Streamline

Version 9.7

User Guide

Issue 00
101816_9.7_00_en



Arm Streamline User Guide

This document (101816_9.7_00_en) was issued on 2025-07-31. There might be a later issue at https://developer.arm.com/documentation/101816

The product version is 9.7.

See also: Proprietary notice | Product and document information | Useful resources

Start reading

If you prefer, you can skip to the start of the content.

Intended audience

This book describes how to use Arm® Streamline Performance Analyzer to capture, analyze, and display real-time performance measurements from your system. It is aimed at all users of Arm® Streamline.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com.

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Contents

1. Getting started with Streamline....7
1.1 Introduction to Streamline....7
1.2 Install Streamline....9
1.3 Standards compliance....11
1.4 Profile your Android application....11
1.5 Profile your Linux application....12
1.6 Profile your bare-metal application....13

2. Capture a Streamline profile....14
2.1 Starting a capture....14
2.2 Run a command on the target....18
2.3 Live view overview....19
2.3.1 Process list....19
2.3.2 Adding and removing events in the Live view....20
2.3.3 Software-generated events in the Live view....21
2.4 Set capture options....22
2.5 Analysis options....23
2.5.1 Generate debug information....23
2.5.2 Select report resolution....24
2.5.3 Select images and libraries for profiling....25
2.5.4 Program Image Table Reference....26
2.6 Download process images....27
2.7 Separate debug files....29
2.8 Counter Configuration....30
2.8.1 Add counters from a template....30
2.8.2 Create a custom counter configuration....31
2.8.3 Setting up event-based sampling....32
2.8.4 Importing and exporting counter configuration files....33
2.8.5 Configure SPE counters....34
2.8.6 Enable Mali Timeline events using the Streamline GUI....37
2.8.7 Enable Mali Timeline events for headless capture....40
2.9 Set up a local capture on a device using the Streamline GUI....43

2.10 Set up a local capture on a device using the command-line....45
2.11 Overview of CPU profiling....47

3. Annotate your code....49
3.1 View annotations....49
3.2 User space annotations....51
3.2.1 String annotations....51
3.2.2 Visual annotations....52
3.2.3 Marker annotations....54
3.2.4 Custom counter annotations....55
3.2.5 Group and Channel annotations....57
3.2.6 Custom Activity Map annotations....58
3.3 Kernel space annotations....60
3.4 Add annotations to your code....62

4. Analyze your capture....66
4.1 Timeline overview....66
4.2 Arm NN Timeline....68
4.3 Working with charts....69
4.4 Analyze a region of time with the cross-section marker....70
4.5 Analyze a region of time with the calipers....71
4.6 Customizing your charts....72
4.6.1 Chart configuration panel....72
4.6.2 Chart configuration toolbar options....72
4.6.3 Chart configuration series options....76
4.6.4 Create a chart configuration template....78
4.6.5 Apply a chart configuration template....79
4.6.6 Match dynamic counter sources....80
4.7 Viewing application activity....82
4.7.1 Details panel modes....82
4.7.2 Heat Map mode....83
4.7.3 Core Map and Cluster Map modes....85
4.7.4 Filtering by threads or processes....86
4.7.5 Selecting the activity source....86
4.7.6 Samples mode....87
4.7.7 Processes mode....88
4.7.8 Mali Timeline mode....89

4.7.9 Images mode....92
4.7.10 OpenCL mode....92
4.8 Toolbar options in the Live and Timeline views....95
4.9 Contextual menu options in the Live and Timeline views....97
4.10 Warnings tag....99
4.11 Analyze your code....99
4.11.1 Analyzing Call Paths....100
4.11.2 Analyzing Functions....101
4.11.3 Sorting table reports....102
4.11.4 Viewing your code....103
4.12 Streamline Data view....105
4.12.1 Capture color-coding....106
4.12.2 Re-analyze stored capture data....106
4.12.3 Duplicating a capture....107

5. Use Streamline from the command line....109
5.1 Open a Streamline-enabled command prompt or shell....109
5.2 List modes....110
5.3 Generate-config mode....110
5.4 Capture mode....111
5.5 Analyze mode....113
5.6 Report mode....113
5.7 Import modes....117

6. Add custom counters from files and trace events....119
6.1 Creating a configuration.xml file....119
6.2 Filesystem and ftrace counters....121
6.2.1 Filesystem counters....121
6.2.2 ftrace event counters....122
6.2.3 Create filesystem and ftrace counters....124
6.3 Counter classes....124

7. Keyboard shortcuts....127
7.1 Keyboard shortcuts in the Live and Timeline views....127
7.2 Keyboard shortcuts in the table views....128
7.3 Keyboard shortcuts in the code view....129

8. Importing raw perf data....130
8.1 Import raw perf data....130

9. JIT profiling support....132
9.1 JIT profiling using gator....132
9.2 JIT profiling using perf....135

10. Troubleshooting Streamline....138
10.1 Troubleshooting report issues....138
10.2 Troubleshooting missing source files....140
10.3 Troubleshooting perf import issues....141
10.4 Troubleshooting Streamline crashes during a capture....142

Proprietary notice....144

Product and document information....146
Product status....146
Revision history....146
Conventions....147

Useful resources....149

1. Getting started with Streamline

Quickly get started profiling your Android, Linux, or bare-metal application with Streamline.

Streamline is an application profiler that can capture data from multiple sources, including:

- Program Counter (PC) samples from running application threads.
- Samples from the hardware Performance Monitoring Unit (PMU) counters in the Arm CPU, Arm® Mali™ GPUs, and Arm® Immortalis™ GPUs.
- Thread scheduling information from the Linux kernel.
- Software-generated annotations and counters from running applications.

This data capture is extensible, allowing collection from other hardware or software sources that are available.

This chapter provides an introduction to Streamline and directs you to the appropriate getting started instructions for your application.

1.1 Introduction to Streamline

Streamline helps you optimize software for devices that use Arm® processors.

Evaluate where the software in your system spends most of its time by capturing a performance profile of your application running on a target device. Quickly determine whether your performance bottleneck relates to the CPU processing or GPU rendering using interactive charts and comprehensive data visualizations.

For CPU bottlenecks, use the native profiling functionality to locate specific problem areas in your application code. Investigate how processes, threads, and functions behave, from high-level views, right down to line-by-line source code analysis. The basic profile is based on regular sampling of the PC (Program Counter) of the running threads, allowing identification of the hotspots in the running application. Hardware performance counters that are provided by the target processors can supplement this analysis. These counters enable hotspot analysis to include knowledge of hardware events such as cache misses and branch mispredictions.

For GPU bottlenecks, use performance data from the Arm® Mali™ GPU driver and hardware performance counters to explore the rendering workload efficiency. Visualize the workload breakdown, pipeline loading, and execution characteristics to quickly identify where to apply rendering optimizations.

With Streamline, you can:

- Find hot spots in your code to be targeted for software optimization.
- Identify the processor that is the major bottleneck in the performance of your application.

- Use CPU performance counters to provide insights into L1 and L2 cache efficiency, enabling cache-aware profiling.
- Identify the cause of heavy rendering loads which cause poor GPU performance, and use GPU performance counters to identify workload inefficiencies.
- Reduce device power consumption and improve energy efficiency by optimizing workloads using performance counters from the CPU, GPU, and memory system.

Android application profiling

Streamline supports data capture on Android devices. Streamline collects CPU performance data and Arm® Mali™ GPU, or Arm® Immortalis™ GPU, performance data so that you can profile your debuggable game or application without device modification. Streamline also supports non-debuggable application profiling on a rooted device. To configure Streamline to collect the right data, use the templates to select the most appropriate set of counters for your device. Alternatively use the built-in templates as a starting point for a customized data visualization. See Profile your Android application.

To learn how to set up your Android target for profiling with Streamline, see the Arm Streamline Target Setup Guide for Android

Android system profiling

In addition to the single application profiling for non-root devices, Streamline supports system-wide Android profiling when running on development devices with root access. System profiling enables manufacturers to simultaneously monitor all applications and services running on their device, allowing identification of problematic processes or scheduling behaviors.

Linux system profiling

Streamline supports system-wide profiling of applications running on Linux-based embedded devices. Analyze the behavior of the system hardware by selecting the required Arm® CPU or Mali™ GPU hardware performance counters for your scenario. To provide more context to the analysis, you can use software annotations in Arm® software libraries, such as the Mali™ GPU OpenCL device driver.Profile your Linux application

To learn how to set up your Linux target for profiling with Streamline, see the Arm Streamline Target Setup Guide for Linux

Bare-metal application profiling

Streamline can profile bare-metal software running on Arm® processors, emitting data over an Arm®CoreSight® ITM, or STM data channel. It can also profile data captured to an on-device memory buffer. Support is included for PC sampling, performance counter sampling, and application-generated annotations.

The target agent for bare-metal profiling is provided as a small source library that is integrated directly into the bare-metal application that is being profiled. This library - called Barman - is auto-generated based on the data channel configuration that is required. It provides optional hooks that allow for lightweight RTOS integration, such as annotation of thread or task context switches. This integration can supplement the basic performance information in the data visualizations.Profile your bare-metal application

To learn how to set up your bare-metal target for profiling with Streamline, see the Arm Streamline Target Setup Guide for Bare-metal Applications

ASTF trace support

Streamline supports the import and visualization of Arm Structured Trace Format (ASTF) instruction traces. For more information, contact Arm at performancestudio@arm.com.

1.2 Install Streamline

This topic describes how to install Arm® Streamline Performance Analyzer as part of Arm® Performance Studio.

Before you begin

- Download the free Arm® Performance Studio install package for your platform at Arm Performance Studio Downloads.

About this task

If you are using Streamline as part of Arm® Development Studio, see the instructions about installing and configuring Arm® Development Studio in the Arm Development Studio Getting Started Guide

Procedure

1. Install:
   - On 64-bit Windows:

     Arm® Performance Studio is provided with an installer executable. Double-click the `.exe` file and follow the instructions in the **Setup Wizard**.
   - On macOS:

     Arm® Performance Studio is provided as a dmg package. To mount the package, double-click the dmg package and follow the instructions. For easy access, the directory tree copies to the **Applications** folder on your local file system.
   - On Linux:

     Arm® Performance Studio is provided as a gzipped tar archive. Use a recent version (1.13 or later) of GNU tar to extract the tar archive to your preferred location:

     ```
     tar xvzf Arm_Performance_Studio_<version>_linux.tgz
     ```
2. Ensure you have installed Python 3.8, or later. Python is used to run the `streamline_me.py` script, which configures and collects data for headless captures on Android devices.
3. Install Android Debug Bridge (adb). Arm® Performance Studio uses the `adb` utility to connect to the target device.
   The adb utility is in the Android SDK platform tools, which are installed as part of Android Studio.

You can also download the latest version of adb from the Android SDK platform tools site.


Note

Streamline uses the system adb path. To use a different adb path, replace the **ADB Path** in **Window > Preferences > Streamline > External Tools**.

4. Add the path to the Android SDK and Python executables to your `PATH` environment variable.

Next steps

1. To launch Streamline:
   - On 64-bit Windows:

     Open the Windows **Start menu**, search for **Arm Streamline**, and select the version of the tool that you want to open.
   - On macOS:

     Navigate to the `<installation_directory>/streamline` folder, and open the `Streamline.app` file.
   - On Linux:

     Navigate to the location where you extracted the package, go to the Streamline directory, and run the Streamline file:

     ```
     cd <install_location>/streamline
     ./Streamline
     ```


Note

If you are running Streamline on the command line, you can use the `Streamline-cli` script to run without triggering any GUI elements.

2. Set up your target for Streamline:
   - Android devices, see: Arm Streamline Target Setup Guide for Android
   - Linux devices, see: Arm Streamline Target Setup Guide for Linux
   - Bare-metal devices, see: Arm Streamline Target Setup Guide for Bare-metal Applications
3. Profile your application:
   - Profile your Android application
   - Profile your Linux application
   - Profile your bare-metal application

Related information

Arm Streamline Target Setup Guide for Android

Arm Streamline Target Setup Guide for Linux
Arm Streamline Target Setup Guide for Bare-metal Applications
Profile your Android application on page 11
Profile your Linux application on page 12
Profile your bare-metal application on page 13

1.3 Standards compliance

Streamline conforms to various levels of compliance.

Hardware

Streamline supports all Arm®v7 and later architecture-based CPUs running Linux and Perf, and is tested on Arm®Cortex®-A cores. Arm®Cortex®-R and Arm®Cortex®-M cores are also supported for bare-metal profiling.

Streamline supports Arm® Mali™ and Arm® Immortalis™ GPUs implementing the Midgard (excluding Mali™-T600 series), Bifrost, Valhall, and 5th Gen architectures.

Executable and Linking Format (ELF)

Streamline can read ELF-formatted executable images.


Note

Streamline can scan **Android Application PacKage (APK)** file archives for ELF images. It extracts the valid ELF executable images and includes them as program images for analysis.

Debugging With Attributed Record Formats (DWARF)

Streamline can read debug information from ELF images in the DWARF 2, DWARF 3, DWARF 4, and DWARF 5 formats.

1.4 Profile your Android application

This topic describes where to find the information about how set up your Android device and application for profiling.

Before you begin

- Download and install Arm® Performance Studio or Arm® Development Studio, and install adb to Set up your host machine.


Note

The latest version of Streamline is compatible with captures taken with earlier versions of Streamline for up to two years. You might have to recapture data to replace captures that are older than two years.

- Compile your application, connect to the host, and set up your device as described in Arm Streamline Target Setup Guide for Android.

Procedure

Follow the procedures in Profile your application in the Arm Streamline Target Setup Guide for Android.

Next steps

- Analyze your capture.
- See the Android performance triage with Streamline tutorial on the Arm Developer website.

Related information

Arm Streamline Target Setup Guide for Android
Compile your application for Android
Set up your host machine
Set up your device
Analyze your capture on page 66
Android performance triage with Streamline

1.5 Profile your Linux application

This topic describes where to find the information about how set up your Linux device and application for profiling.

Before you begin

- Download and install Arm® Performance Studio.

The latest version of Streamline is compatible with captures taken with earlier versions of Streamline for up to two years. You might have to recapture data to replace captures that are older than two years.

Procedure

Follow the instructions in Introduction to using Streamline with a Linux device in the Arm Streamline Target Setup Guide for Linux.

Next steps

Analyze your capture.

Related information

Arm Streamline Target Setup Guide for Linux
Introduction to using Streamline with a Linux device
Analyze your capture on page 66

1.6 Profile your bare-metal application

This topic describes where to find the information about how set up your bare-metal device and application for profiling.

Before you begin

- Download and install Arm® Performance Studio.


Note

The latest version of Streamline is compatible with captures taken with earlier versions of Streamline for up to two years. You might have to recapture data to replace captures that are older than two years.

Procedure

Follow the instructions in Profiling with the bare-metal agent in the Arm Streamline Target Setup Guide for Bare-metal.

Next steps

Analyze your capture.

Related information

Arm Streamline Target Setup Guide for Bare-metal Applications
Profiling with the bare-metal agent
Analyze your capture on page 66

2. Capture a Streamline profile

See how to capture a profile of your application using Streamline. Learn how to use the capture settings to change what information is captured and reported.

2.1 Starting a capture

Start a capture session to profile data from your application in real time. When the capture session finishes, Streamline automatically opens a report for you to analyze later.

Before you begin

Set up your Linux or Android target as described in the relevant Streamline target setup guide.

About this task

For instructions on how to capture data locally on a target, and for more target-specific information, see the Streamline target setup guides.

Procedure

1. In the **Start** view, select your target device type. Then select your device from the list of detected targets, or enter the address of your target.

   Note

   By default, `gatord` uses port 8080 to communicate with the host machine. To use a different port, append a colon followed by the port number to the target address. For example, to use port 5050 with target address 10.99.28.54, enter `10.99.28.54:5050` in the target address field.

2. Android users only, select the package you want to profile from the list of packages available on the selected device.

   Note

   - To profile non-debuggable packages in the list, you must be running a development device with root access.
   - Applications must have at least one MAIN or launchable activity.

   a. Optionally, enter arguments for activities that require additional configuration, or for activities that cannot be listed in the table:

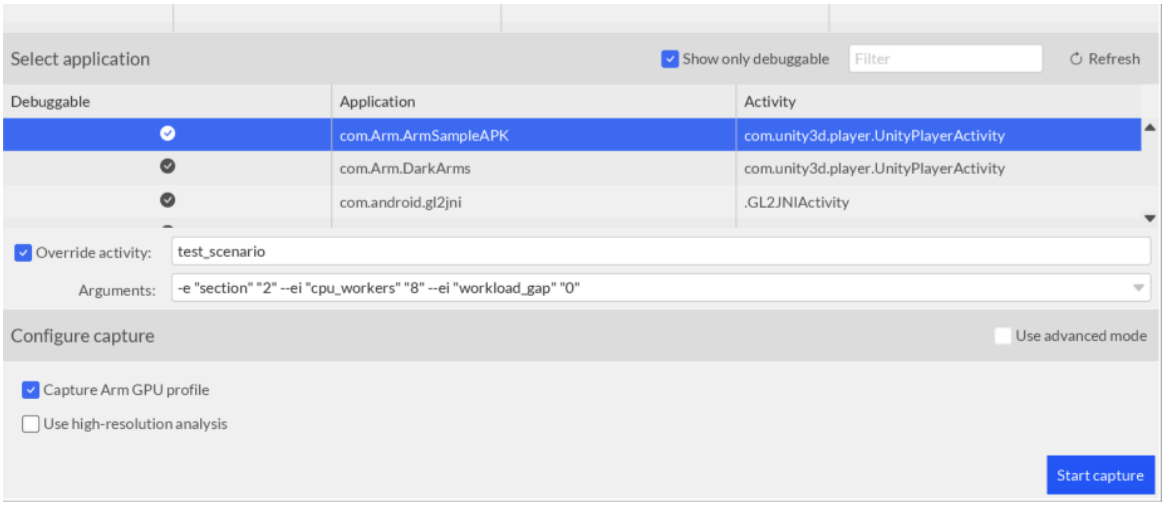
   b. Select the **Override activity** check box, and enter a suitable name for the activity.

   c. In the **Arguments** text box, enter the arguments that you want to run on the selected package.

      You can enter multiple arguments passed on a single line. See valid argument options in https://developer.android.com/tools/adb#IntentSpec.

If the arguments are valid, when you click **Start capture**, the arguments are stored in history so that you can select them again from a drop-down menu. A maximum of 10 arguments are stored before they are overwritten.

Figure 2-1: Example of arguments to run on the selected package.



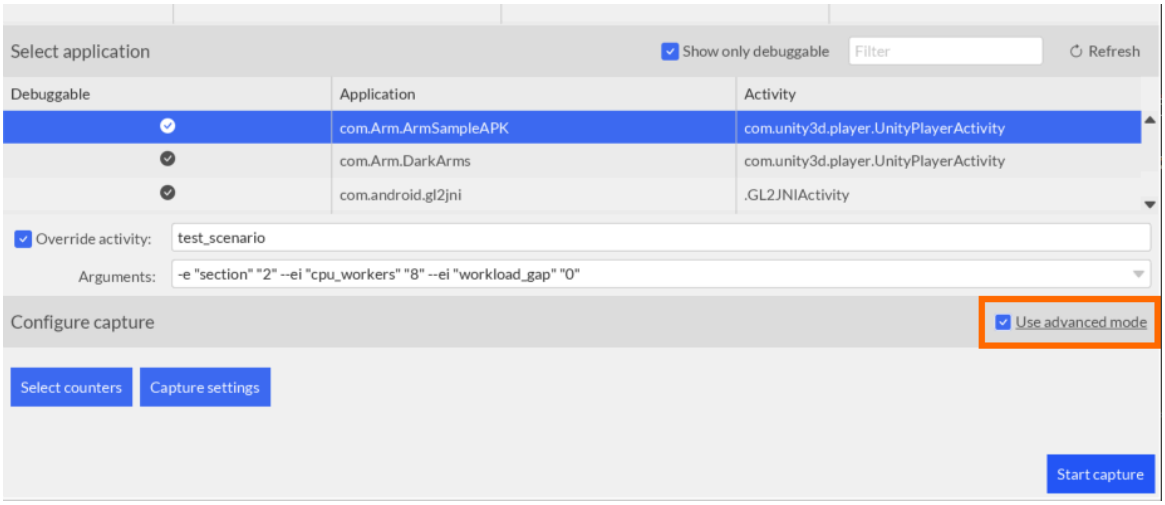
3. TCP users only, optionally enter the details for any command you want to run on the application. See Run a command on the target.
4. Select the counter template that you want to use to review performance of your CPU and GPU:
   - To use the most appropriate counter template for your Arm GPU, select the **Capture Arm GPU profile** checkbox.
   - To use the default counter template, which captures basic information such as CPU and memory usage, but no GPU data, clear the **Capture Arm GPU profile** checkbox.

     Note

     If your device does not contain an Arm GPU, the **Capture Arm GPU profile** checkbox is disabled.

   - To use a different counter template, select the **Use advanced mode** checkbox.

Figure 2-2: Connect using advanced mode.



a. Click **Select counters**.

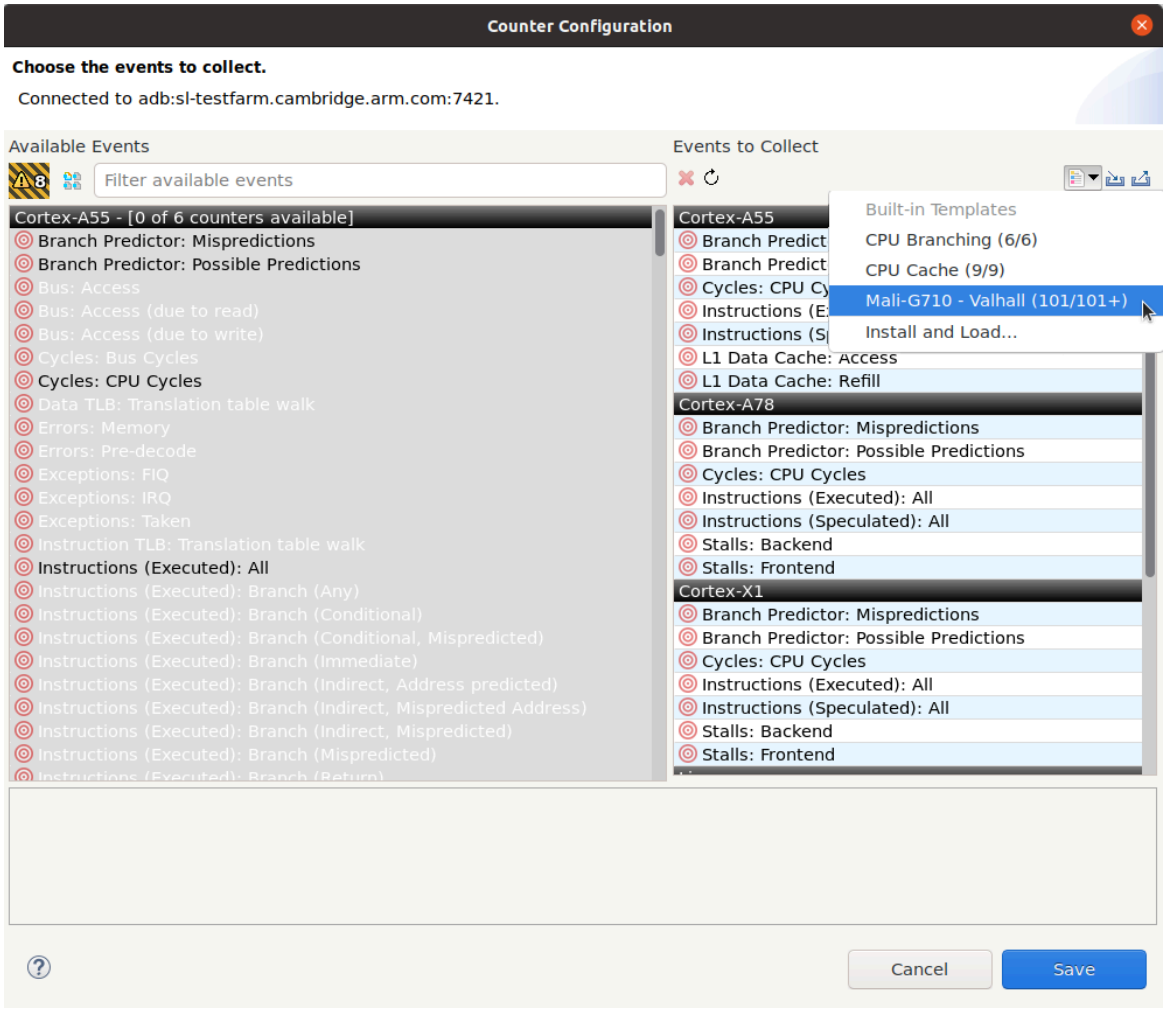
b. Click **Add counters from a template** to see a list of available templates.

c. Select a counter template appropriate for the GPU in your device, then **Save** your changes.

The number of counters in the template that your device supports is shown next to each template. For example, here, 101 of the 101 available counters in the Arm® Mali™ template are supported in the connected device. Streamline notifies you if the target device does not support all the counters that are defined in the selected template.

Alternatively, to import an existing counter template, click **Install and Load**.

Figure 2-3: Templates available from the Select Counters dialog box.



5. Optionally, in advanced mode, you can set options for your capture. For example, change the sample rate, capture duration, or resolution. You can also choose to download images from the target when the capture session ends. See Download process images for more information.
   To open the **Capture Settings** dialog box, click **Capture Settings**.

   Click **Save** to close the **Capture Settings** dialog box.
6. Click **Start capture**.
7. Browse to a location on the host and enter the name of the capture file. Streamline then switches to the **Live** view and starts the application automatically.
   The **Live** view shows charts for each counter that you selected. Below the charts is a list of running processes in your application with their CPU usage. The charts now start updating in real time to show the data that Streamline captures from your running application.
8. In the **Capture Control** view:
   - To stop the capture session and create a performance report, click **Stop capture and analyze**.

- To stop the capture session and discard the collected data, click **Stop capture and discard** .

If **Stop capture and analyze** is grayed out, **Discard data** has been selected in the **Capture Settings** dialog box. To generate a performance report, clear **Discard data** in the **Capture Settings** dialog box, click **Save** to close the dialog box, then click **Start capture**.

Results

When data analysis completes, the application stops on the device. Streamline opens the **Timeline** view and shows the report.

Next steps

- Download process images
- Analyze your capture
- See tutorial Android performance triage with Streamline on the Arm Developer website.

Related information

Run a command on the target on page 18
Set capture options on page 22
Analysis options on page 23
Download process images on page 27
Counter Configuration on page 29
Arm Streamline Target Setup Guide for Android
Arm Streamline Target Setup Guide for Linux
Arm Streamline Target Setup Guide for Bare-metal Applications

2.2 Run a command on the target

Specify a command to run on the target after profiling starts.

Before you begin

- To allow Streamline to communicate with the target, install and run `gatord` on the target.
- When you start `gatord`, you must use the `-a` option.
- In the **Start** view, your device type must be **TCP**.

Procedure

1. In the **Start** view, under **Configure application**, enter a command to run on the target.
2. The command runs in the current working directory of `gatord`. To run the command in a different directory, enter the absolute path to that directory in the **Working directory** field.

3. The command runs as the same user as `gatord`. For example, if `gatord` is run as root, the command runs as root. To run the command as a different user, enter the user account in the **User name** field.
4. To stop the capture session when the command specified in the **Command** field ends, select **Stop on exit**. If you want the capture session to continue after the command ends, leave the checkbox clear.

Results

When you click **Start capture**, `gatord` runs the specified command on the target.

2.3 Live view overview

The **Live** view shows data capture in real time during the capture process. When you start a capture session, the **Live** view opens automatically.

The **Live** view gives feedback during a capture session. The full functionality of Streamline is only available after you have stopped your capture session and Streamline has processed the data in the capture.

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


Note

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

2.3.1 Process list

The process list shows the known processes and their usage statistics. This list continuously updates while the capture session is running. When a process ends, its name and ID remain in the list but are shown in gray. For example, the row for xaos in the following list is gray:

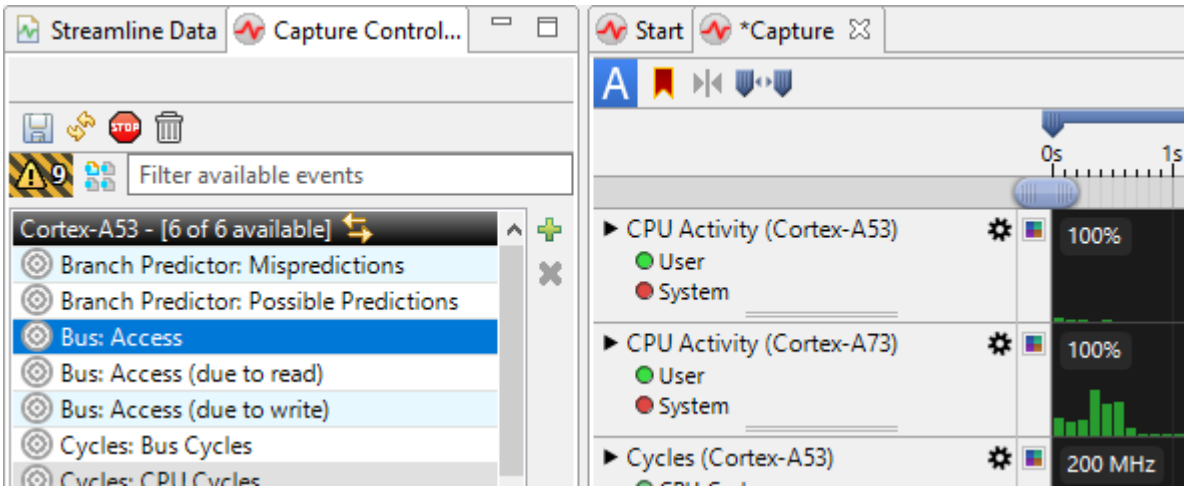
Figure 2-4: Dead process in Live view

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%

2.3.2 Adding and removing events in the Live view

During data capture, add or remove events in the interactive counters panel in the **Capture Control** view.

Figure 2-5: Interactive counters panel and Live view



To select events and add them to the **Live** view, do one of the following actions:

- Select an event, then click **Add selected events** .
- Double-click the event.
- Select an event, then press Enter.
- Drag an event onto the chart panel.

To remove events from the **Live** view, do one of the following actions:

- Select an event, then click **Remove selected events** .
- Select an event, then press Delete.
- Remove the chart or series.

> **Note**
> Removing a chart or series only removes the counter if no other chart or series is using it.

- Drag and drop a chart into the interactive counters panel.

When you have finished configuring the **Live** view charts, you can update the view:

- To save the current counter configuration and restart the capture with the new configuration, click **Save and restart** .


Note

If you have selected the **Discard data** option in the **Capture Settings** dialog box, this button is inactive and appears grayed out. When the capture session terminates, Streamline discards the data.

- To discard the current capture and restart with the new configuration, click **Restart capture** .

Changes to the counter configuration also occur when you:

- Edit a series expression, counters are automatically added or removed.
- **Switch and manage templates** .
- **Add default charts** .

Setting event-based sampling

If an event supports event-based sampling, **Toggle event-based sampling** appears before the event name in the interactive counters panel.

To enable or disable event-based sampling for an event, click **Toggle event-based sampling** . Set the reporting threshold in the **Threshold** field that appears at the bottom of the interactive counters panel when event-based sampling is enabled.

Other user indicators

- A gray warning triangle next to the series title indicates event charts that have been configured but not committed.
- A red triangle next to the series title indicates that there are no available counters for the chart.
- Hovering over an event in the interactive counters panel highlights any series that is using it.
- Hovering over a series or chart title highlights any events in the interactive counters panel that it is using.
- Not all events are linked to the charts. Dynamic counters, like atrace, can only be configured using the counters panel, not using the charts. A double arrow in the category heading indicates linked categories.

2.3.3 Software-generated events in the Live view

During a capture, the application on the target device can generate software-defined events and counters. These software-defined counters automatically appear in the **Live** view when they are first generated, but they are not available to select in the **Select Counters** dialog box.

When you apply a template in the **Live** view, the template is applied to any existing counters. After you have applied a template, any new software counters generated will automatically appear in the **Live** view, even if they are not part of the selected template. To filter out any unwanted series, reapply the template.

2.4 Set capture options

The default capture settings work well for most situations but you can change the capture settings to customize your profiling session. These settings include the sample rate, buffer size, and duration of the capture.

Before you begin

If you are recording call stacks, compile your images and libraries with frame pointers enabled using the `-fno-omit-frame-pointer` compiler option. For Arm®v7 software applications, you must also disable the Thumb® instruction set by using the `-marm` option.

Procedure

1. In the **Start** view, select the **Use advanced mode** checkbox.
2. Click **Capture Settings**.
3. The **Normal** sample frequency is 1kHz for CPU events, and 2kHz for GPU events. To change how regularly Streamline collects samples, change **Sample rate** to:
   - **High** to increase sample frequency to 10kHz in both CPU and GPU events. This enhances visibility, but also increases overhead.
   - **Low** if you have a slow target, or if the target is heavily loaded, for a longer capture at a sample frequency of 100Hz for both CPU and GPU events.
   - **None** to have the lowest level of intrusion on your code, and disable periodic sampling. Any report columns that rely on sampling show zero values.

   
   Note

   For older GPUs, such as Bifrost and earlier, GPU sampling in `Normal` mode is set to 1kHz.
4. If you do not want to display real-time data during a capture session, set **Buffer mode** to the appropriate buffer size. The capture ends when the buffer is full:
   - **Large** to set the buffer size to 256MB
   - **Medium** to set the buffer size to 64MB
   - **Small** to set the buffer size to 16MB

   Set **Buffer mode** to **Streaming** to enable **Live** view and allow streaming of target data directly to your host during capture.
5. To record call stacks and improve the visibility of the behavior of the target, select **Call stack unwinding**.
   - Recording call stacks increases the amount of raw data that the target sends to the host.
   - To support call stack unwinding for binaries that are created using GCC or LLVM, enable frame pointers before you compile them.

6. To let Streamline decide which events to capture, select **Auto** in the **Exclude kernel events** drop-down menu. If you start your capture in the **Android** tab in the **Start** view, Streamline does not capture kernel events. If you start gator manually, you can override this using the `--system-wide` command line option.
   - To capture userspace events only, and ignore kernel events, select **Yes** in the **Exclude kernel events** drop-down menu.
   - To capture kernel events and userspace events, select **No** in the **Exclude kernel events** drop-down menu.
7. To collect Ftrace data more efficiently, select **Use more efficient Ftrace collection**. The **Live** view does not update charts that use Ftrace data, but the data is in the finished report.
8. To set a **Duration** for the capture, enter the time that you want the capture session to last either in seconds or in minutes and seconds.
   - Enter 60 to set a capture session for 60 seconds.
   - Enter 1:20 to set a capture session for 1 minute 20 seconds.

   If you do not enter a duration, the capture session continues until you stop it manually or, if you are not using streaming, until the buffer is full.
9. To download images from the target when the capture session ends, select the **Download images from target** checkbox.


Note

You can also set this preference in **Window > Preferences > Streamline**.

10. Click **Save**.

Related information

Download process images on page 27

2.5 Analysis options

Use the **Capture Settings** dialog box to change settings that control the data that is provided for analysis of your capture sessions. For example, debug information, report data, and image files.

To open the **Capture Settings** dialog box, in the **Start** view, click **Capture Settings**.

2.5.1 Generate debug information

Process DWARF debug information and line numbers for a higher level of detail in your capture session.

Before you begin

To process debug information, you must have built the image using the `-g` compiler option.

Procedure

1. In the **Start** view, select the **Use advanced mode** checkbox.
2. Click **Capture Settings**.
3. In the **Analysis** section, select **Enable source code cross-references** to process source code mappings and call stack information. If you clear **Enable source code cross-references**, the source section shows a `No source available` message.


Note

- Selecting **Enable source code cross-references** results in higher memory usage.
- This option can be changed when you re-analyze the stored capture.

4. Click **Save**.

Results

The report data that is automatically generated after the termination of the capture session contains information to help you debug your code, including:

- Source code mappings.
- Call stack information.

Related information

Standards compliance on page 11

2.5.2 Select report resolution

Change the data resolution to determine the level of detail provided in the report data that is generated from your capture session. A higher level of detail allows you to view data with a finer time granularity, but results in a longer analysis time and higher memory usage.

Procedure

1. In the **Start** view, click **Capture Settings**.
2. In the **Analysis** section, select the **Resolution** you require for data analysis:
   - **Summary** to set the time bin to 10 milliseconds
   - **Normal** to set the time bin to 500 microseconds
   - **High** to set the time bin to 100 microseconds
   - **Ultra-High** to set the time bin to 1 microsecond


Note

You can change the **Resolution** option when you re-analyze the stored capture.


Note

The **Summary** option is limited to two zoom levels; 1 second and 10 milliseconds. Its intended use is for summarizing large, or very long, captures where fine-grained inspection of the **Timeline** view is not required.

3. Click **Save**.

Results

The automatically generated report data contains the specified level of detail processed from your capture session.

2.5.3 Select images and libraries for profiling

Browse your file system, and select the images and libraries you want to profile.

Before you begin

- When compiling images on your host, ensure that you use the `-g` compiler option to enable debug symbols.
- To improve the call path quality, disable inlining with the `-fno-inline` compiler option.


Note

For more information on compiling your application, see Compile your application for Android or Compile your application for Linux.

Procedure

1. In the **Start** view, click **Use advanced mode > Capture settings**.
2. In the **Program Images** section, control symbol loading for the ELF images, executables, or APK listed:


Note

If you compiled your application for multiple architectures, you must unzip the `app-debug.apk` to see the `.so` files in **Program Images**.

   - Select **Use all images for analysis** or **Use none of the images for analysis**.
   - To browse your file system and choose an image executable or APK, click **Add image**, select the required file, then click **Open**.

     If you select an image with the extracted debug information and CFI on this step, Streamline does not show the disassembly of this image.

     Streamline relies on executable code to generate accurate callpath information. Arm recommends that you always provide complete ELF images for analysis.

- To select a separate file containing the extracted debug information and CFI, click **Select separate debug image**.

  This option is available when the **Debug Info** or **CFI** column displays the  icon.
- To remove the selected entries, click **Remove**.

3. In the **Program Images** table, select the checkbox in the **Use** column for each image you want to include in the analysis.
4. Click **Save**.
   You can change these images when you re-analyze the stored capture.

Results

The images you selected in the **Program Images** table are now ready for profiling.

See Download process images for more information about downloading images during your capture session, instead of manually selecting images and libraries on the host in the **Program Images** section.

Related information

Program Image Table Reference on page 26
Download process images on page 27

2.5.4 Program Image Table Reference

Symbols are displayed in the **Program Images** table to show which information is available for each program image.

Here is what the following symbols mean in the **Program Images** table:

Code

Displays the following icons to indicate whether program code is available:

✔ The image contains program code.

● No program code is available. Streamline will not be able to show the disassembled code of this image.

Symbols

Displays the following icons to indicate whether symbol information is available:

✔ The image contains symbol information.

● No symbol information is available.

Debug Info

Displays the following icons to indicate whether debug information is available:

✔ Debug information is available, providing source line numbers and inlining information.

 Debug information is available in a separate file (split debug image).

 File contains no debug information, and no link to external debug information.

 Availability of debug information is unknown.

CFI

Displays the following icons to indicate whether Call Frame Information (CFI) is available:

✔ The `debug_frame` or `eh_frame` section is available. This information is used to determine stack frame sizes in the **Call Paths** view.

 The `debug_frame` or `eh_frame` section is available in a separate file.

 File contains no CFI, and no link to external CFI.

 Availability of CFI is unknown.

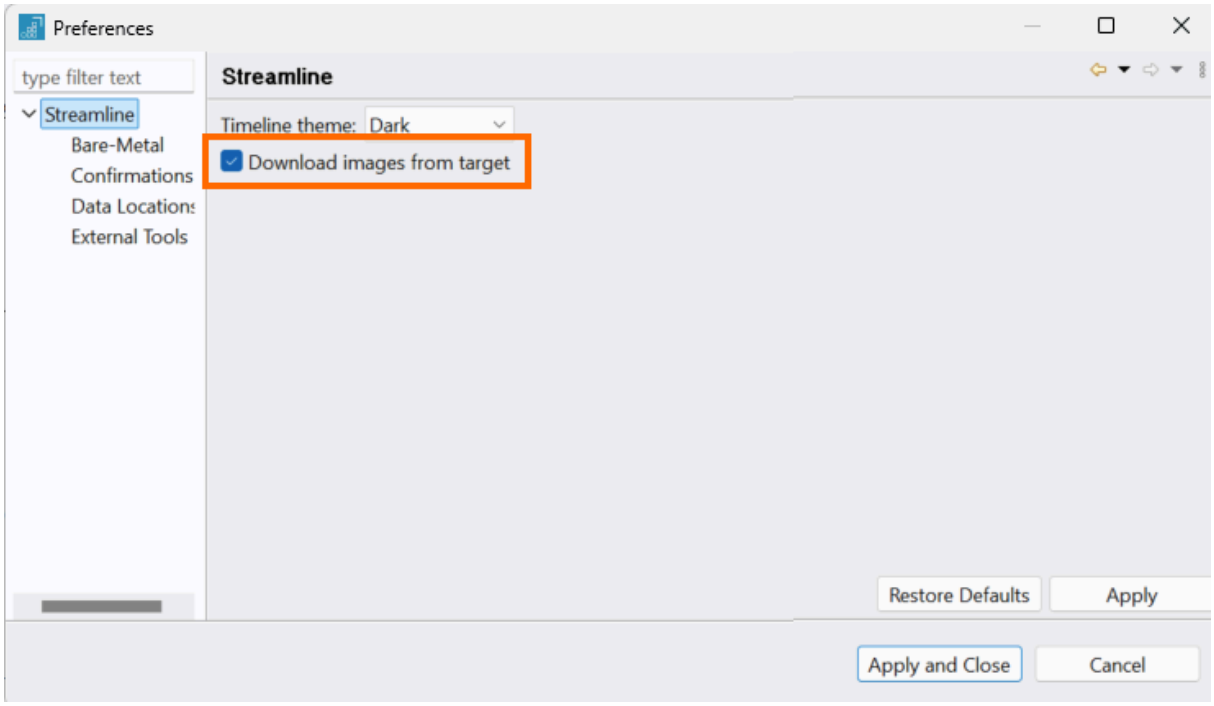
2.6 Download process images

Download process images from the target during the capture session using the image download functionality.

Procedure

1. Before starting a capture, open **Window > Preferences > Streamline**, and select the **Download images from target** checkbox.

Figure 2-6: Select Download images from target.



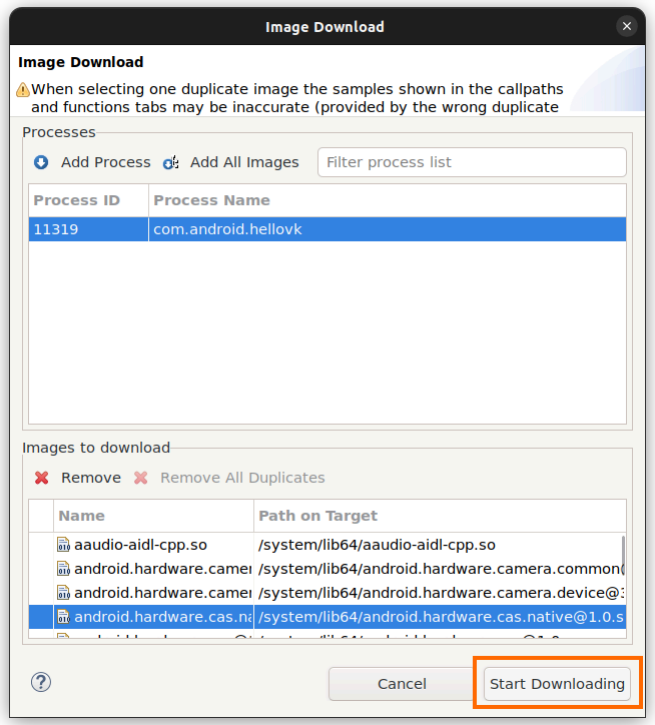
Note

You can also set this preference from **Use advanced mode > Capture settings**.

When the capture ends, the **Image Download** dialog box opens. This dialog box lists the processes that are running on the target, and the process images that you have selected to download.

2. Select a process then click **Add Process** to add it to the list of images to download.
3. Remove all duplicate images before continuing. To remove an image from this list, select the image then click **Remove**. You can remove all of the selected duplicate images by clicking **Remove All Duplicates**.
4. To download the selected images, click **Start Downloading**.

Figure 2-7: Start downloading images from your capture session.



Results

You can now analyze the process images that you selected in the **Image Download** dialog box.

Related information

Separate debug files on page 29
Starting a capture on page 14

2.7 Separate debug files

To limit the size of the executable image file, you can save debug information in a separate file. Streamline supports the separate debug file linking process that is used in many ELF images.

You can locate the separate debug file in two ways:

- The executable includes the name of the debug file, which usually is derived from the executable file name and suffixed with `.debug`.
- The debug file name is derived from the build ID in the executable.

For more information about debugging information in separate files, see https://sourceware.org/gdb/onlinedocs/gdb/Separate-Debug-Files.html.

2.8 Counter Configuration

Streamline uses hardware performance counters to collect the data you need to analyze your applications. Instead of using the default set of counters, you can create your own custom counter configuration.

Arm® Mali™ GPUs implement a comprehensive range of performance counters, that enable you to closely monitor GPU activity as your application runs. The charts in Streamline visualize this performance counter activity, to help you identify the cause of heavy rendering loads or workload inefficiencies that cause poor GPU performance. For detailed descriptions of all the performance counters available for each Mali™ GPU, refer to the Mali GPU Counter references on the Arm Developer website.

2.8.1 Add counters from a template

Counter templates are pre-defined sets of counters. They provide a quick way to review performance of both CPU and GPU behavior, in particular to analyze Arm® Mali™ GPU performance.

Before you begin

Set up your host machine and connect to a target device as described in the relevant Streamline target setup guide.

About this task

To use the most appropriate counter template for your Arm GPU, select the **Capture Arm GPU profile** checkbox. If you would rather choose a different template, or configure your own, follow this procedure.

Procedure

1. In the **Start** view, select the **Use advanced mode** checkbox.
2. Click **Select counters**.
   The **Select Counters** dialog box opens.
3. Click **Add counters from a template**  to see a list of available templates.
4. Select a template from the drop-down list, then **Save** your changes.
   - Streamline notifies you if the target device does not support all the counters that are defined in the selected template.
   - If a template shows a number with a +, such as (55/55+), the template contains wildcards that cannot be resolved until the counters are dynamically generated. After you have captured your profile, if you do not see the results that you expect, then you must add different counters to your template.

Results

The template populates the **Events to collect** list, and the template file is added to the **Selected Templates** list in the drop-down list.

Next steps

If the template does not include some events that you want to profile, then you must add them manually. See Create a custom counter configuration for more information about how to create a custom counter configuration.

2.8.2 Create a custom counter configuration

Streamline uses the counters in the **Events to collect** list to determine what data it collects during a capture session. If an event you want to profile is not shown, you can add it to the list and create a custom counter configuration.

Before you begin

Set up your host machine and connect to a target device as described in the relevant Streamline target setup guide.

Procedure

1. In the **Start** view, select the **Use advanced mode** checkbox.
2. Click **Select counters**.
3. Double-click an event, or click and drag an event to move it from **Available events** to the relevant category in the **Events to collect** list.
   - The **Available events** list contains categorized events that are offered for each core on your target, and other hardware and OS-specific events. Events that are contained in the processor lists are based on the PMU counters of the core. They can vary depending on the type of processor, as can the number of events that you can add.
   - To show advanced events for your target in the **Available events** list, click **Toggle advanced view** .

   The added events appear in gray in the **Events to collect** list under their category name, and are used by the **Timeline** view for its graphs. Each event that is listed here is available for display in a chart in the **Timeline** view.

   Note
   - Selecting a high number of counters might affect performance of the application on the target.
   - To automatically collect the set of counters that are required to generate the charts defined in a chart configuration template, see Add counters from a template. To setup chart configuration templates, click **Switch and manage templates** in the **Live** view and **Timeline** view.
   - To reset the **Events to collect** list to default, click **Reset to default** .
4. If a counter can be collected from different cores or interfaces, select a specific core or interface from the menu next to the counter in the **Events to collect** list.
5. Click **Save**.

6. (Optional) To save your custom counter configuration for future reuse, export the configuration to a file, which you can edit and import later. To learn how to export your configuration, see: Importing and exporting counter configuration files.
   For example, you can reuse an exported configuration file to profile other applications on the same device. Another example is to run Streamline in headless mode (see Generate a headless capture), where you must specify a configuration file to use on your command line.

Next steps

The number of hardware-specific events that you can collect during a capture session is limited. To make space to add new counters, you can remove unwanted counters from the **Events to collect** list. In the **Select Counters** dialog box, select all the events that you want to remove from the **Events to collect** list, then click **Remove selected** .

Related information

Create a chart configuration template on page 78
Generate a headless capture

2.8.3 Setting up event-based sampling

The default behavior of Streamline is to capture data samples based on a regular time step. Alternatively, event-based sampling (EBS) enables you to take samples when a user-defined counter reaches a threshold value. For example, triggering data samples for every 100,000 level two cache misses.

About this task

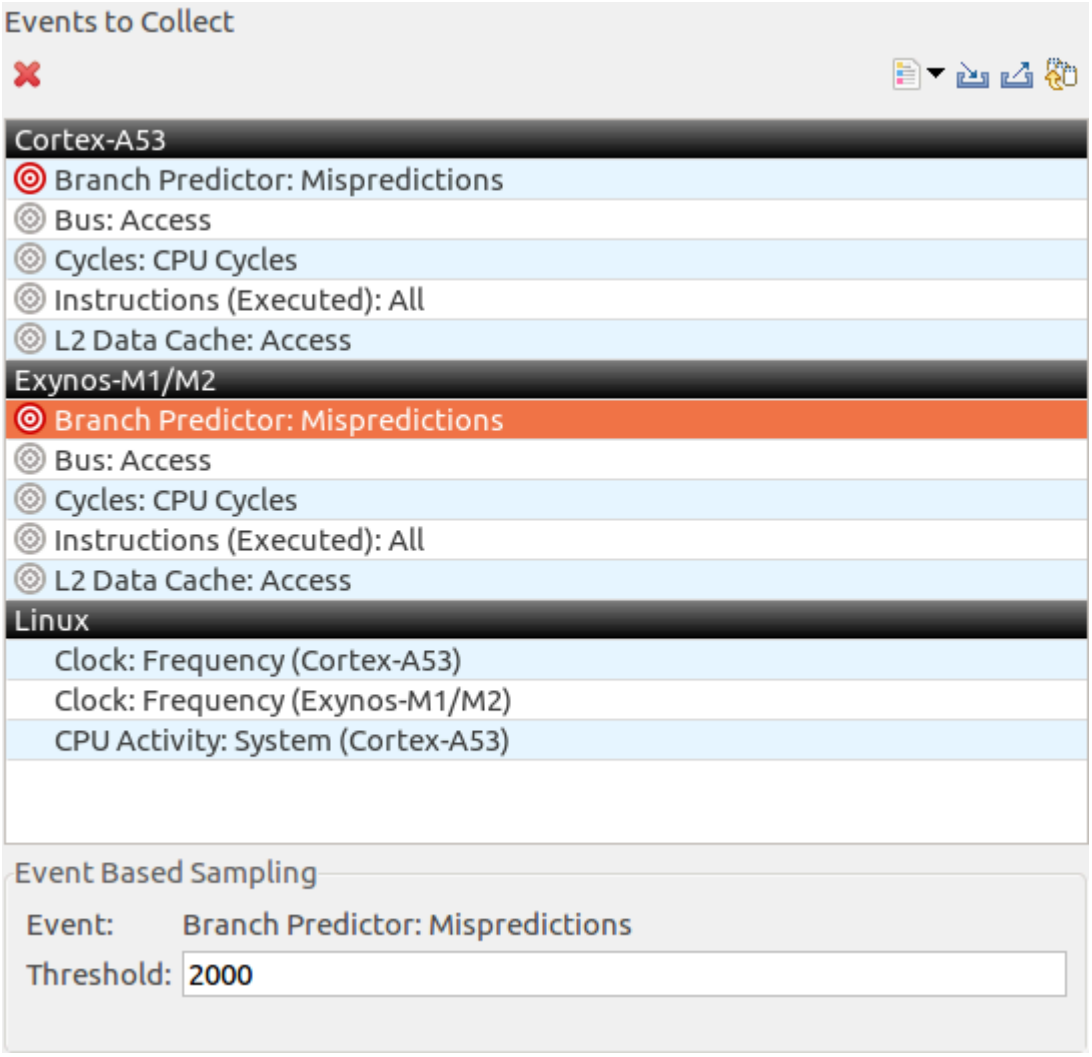
- Only CPU hardware counters and counters that are in the category **Perf Software** support EBS.
- EBS is only possible when the PMU on the target hardware can generate interrupts.

Procedure

1. In the **Start** view, select the **Use advanced mode** checkbox.
2. Click **Select counters**.
3. Click **Toggle event-based sampling** for an event in the **Events to collect** list.
   Event-based sampling is active for all matching events.

   If **Toggle event-based sampling** is not displayed next to an event, then that event does not support event-based sampling.
4. Enter a value in the **Threshold** field.
   Avoid setting a very low threshold for high frequency events, because it might cause a capture to fail. If you enter a threshold value that generates too many samples, the capture could fail, and you might have to restart your target. Your target for the **Threshold** field should be to generate 1000 samples per second. Therefore, if your application typically generates two million counter increments per second, a good value to insert in the **Threshold** field is 2000.

Figure 2-8: Setting up event-based sampling



2.8.4 Importing and exporting counter configuration files

Export the current counter configuration to an XML file so that you can capture data locally. Import a counter configuration file that you previously exported to overwrite your current counter configuration file.

Procedure

1. In the **Start** view, select the **Use advanced mode** checkbox.
2. Click **Select counters**.
3. Choose the relevant action:
   - To search for and import a counter configuration XML file, click **Import**.

- To export the current counter configuration to an XML file, click **Export**. By default, the counter configuration file is called `configuration.xml`, and is saved to the same directory as `gatord` on your target.

4. If you want to capture data locally, you can manually add the exported XML file as an option when running `gatord` on the target.
5. Click **Save** to confirm and close the **Select Counters** dialog box.

2.8.5 Configure SPE counters

Filter sample records that are generated from Statistical Profiling Extension (SPE) counters using the **Select Counters** dialog box. Sample records that do not meet the filtering criteria are discarded and are not written to the profiling buffer.

Before you begin

- You must use hardware which supports the statistical profiling extension.
- You must have a Linux kernel with:
  - The `arm_spe_pmu` module enabled.
  - Support for SPE in the device tree or UEFI.
- Depending on the hardware and kernel configuration, you might need to disable KPTI. To do this, use the kernel command-line argument `kpti=off` when booting the device. You must disable KPTI on Arm®v8.2 processors with SPE, such as the Neoverse™ N1.

About this task

Usually Streamline collects information from hardware counters, which provide aggregate counts. You can therefore only attribute counters to regions of the application. The PC is sampled in software using a timer interrupt, so the sampling rate is limited. SPE is an optional extension to the Arm®v8.2-A architecture that samples the PC periodically in hardware as part of the pipeline of the processor. This form of sampling has a low probe effect so you can set the sampling rate much higher. As it is built into the pipeline, SPE can also collect extra information about each sampled instruction. It therefore allows for a much more detailed analysis of the executed code.

Streamline supports visualizing the following SPE data:

Latency counter packets

Provide issue and total instruction execution latency counts, which help identify execution stalls. They also provide load and store latency counts for memory accesses. Use this data to identify high latency accesses and poor cache use.

Event packets

Provide information about each sampled instruction, including whether:

- It accessed, hit, or missed a cache level.
- It was a mis-predicted or not taken branch.
- An exclusive load or store failed.

Use this data to identify branch prediction problems, poor cache use, and lock contention.

Data source packets

Provide information about the level of the memory hierarchy that a load or store accessed.

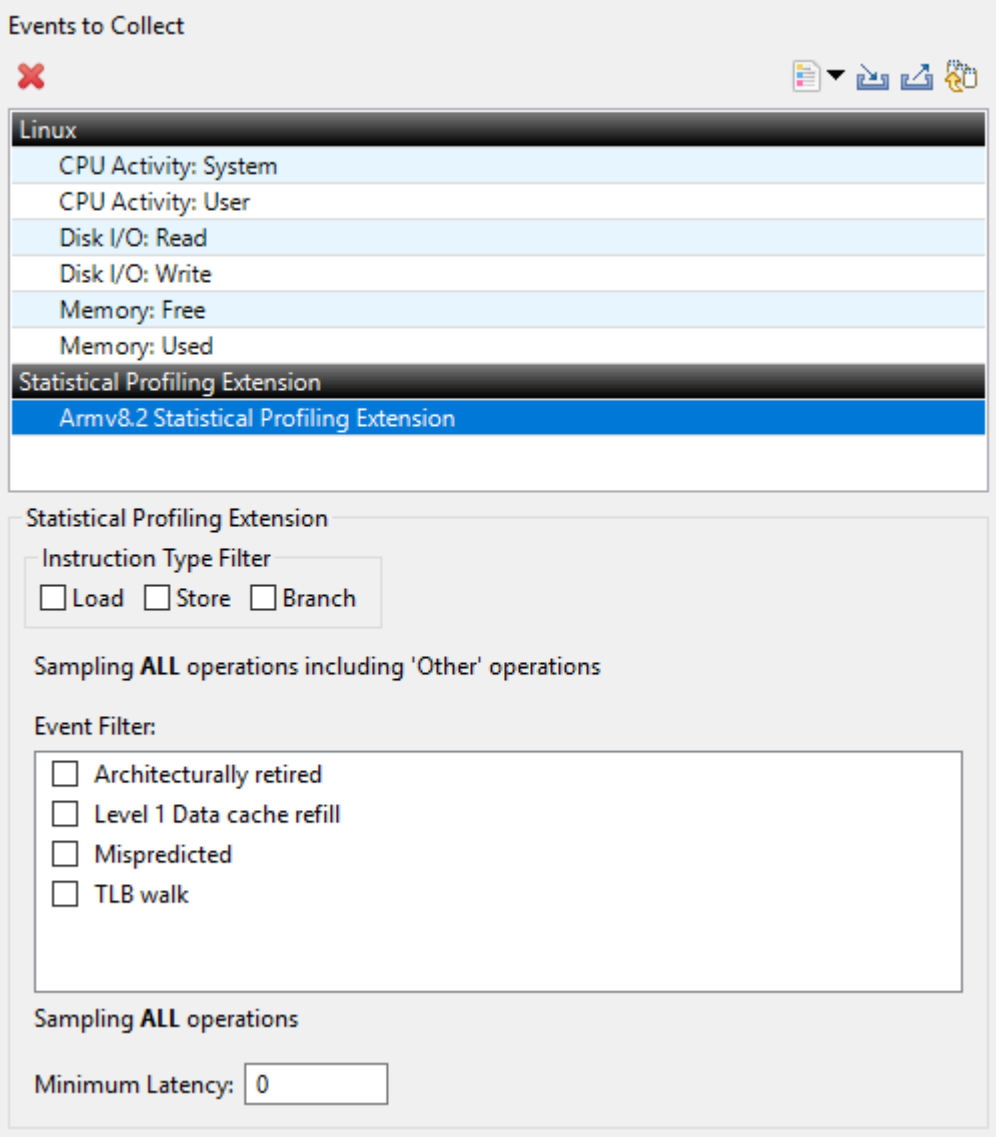
By default, all operations are sampled, but you can filter the samples using the SPE settings in the **Select Counters** dialog box.

The invert event filter and branch not taken filter require the use of a device that supports SPE 1.2 and later (Armv8.7 Statistical Profiling Extension and later).

Procedure

1. In the **Start** view, select the **Use advanced mode** checkbox.
2. Click **Select counters**.
3. In the **Events to collect** list, select a counter in the **Statistical Profiling Extension** category.
4. Configure the SPE counter.

Figure 2-9: The SPE part of the Select Counters dialog box.



For example:

- To identify operations that are slow because they access memory instead of the cache, set a minimum latency.
- To find branches that trigger mispredictions, select the **Mispredicted** event filter.

5. Click **Save** to save your settings and close the **Select Counters** dialog box.

Results

When the analysis is complete, Streamline displays charts for the SPE counters in the **Timeline** view. It also adds SPE data to the **Call Paths**, **Functions**, and **Code** views.

Next steps

Examine the data at the thread, function, source line, or instruction level in the **Call Paths**, **Functions**, and **Code** views.

Related information

Analyze your code on page 99
Analyzing Call Paths on page 99
Analyzing Functions on page 101
Introduction to Statistical Profiling Support in Streamline

2.8.6 Enable Mali Timeline events using the Streamline GUI

Mali™ Timeline events present GPU queue scheduling information through a timeline view. This topic describes how to collect Mali™ Timeline events from the Streamline GUI.

Before you begin

- Ensure your device supports Mali™ Timeline events.

  Mali™ Timeline events requires your device to:

  - Have a release build of Android 11 or later
  - Have a Mali™ device driver version r43p0 or later. To use the GPU Timeline layer driver, you must have a Mali™ device driver version r51p0 or later.
  - Be running the Perfetto service

  
  Note

  Mali™ Timeline events:

  - Are not supported on Midgard architecture GPUs
  - Support OpenCL data from version r43p0 for Android and Linux. If your application contains OpenCL content, select the **Mali Timeline** counter to show the events in the **Compute Queue** Custom Activity Map (CAM) track.

- Connect your device.

  Navigate to the **Start** view in the Streamline GUI, then select your **device type**, **device**, and **application**.

- Linux users must manually set the environment variable `MALI_GPU_RENDERSTAGES_ENABLE=1`. For more information, see System requirements for Mali Timeline support in the Arm Streamline Target Setup Guide for Linux.

About this task

You can use Mali™ Timeline Events to help you:

- Identify scheduling issues where the queues run serially, for all or part of the frame.
- Correlate events with render passes and compute dispatches.

- Understand what is happening in your application by using semantic information.

Procedure

1. Enable data collection using the GPU Timeline layer driver:
   - Android (adb) using the **Capture Arm GPU profile**, the layer driver is deployed automatically. Streamline selects the Mali™ Timeline events counter if that is the most appropriate counter template for your GPU.
   - Android (adb) using the **Capture settings > Use advanced mode** checkbox:
     a. Select the relevant option from the **GPU Timeline layer driver** menu:
        - Android (adb) users:
          - Select **Auto** to automatically deploy and enable the layer driver only if you have enabled the **Mali Timeline events: Perfetto** counter.
          - Select **Enable** or **Force** to automatically deploy and enable the layer driver. To capture Mali Timeline events, you must also enable the **Mali Timeline events: Perfetto** counter.
        - TCP users, select **Force** to automatically deploy and enable the layer driver.

        If you select **Auto**, or **Enable**, you must manually push the GPU Timeline layer to the targeted Android device. For example:

        ```
        adb push bin/android/arm64/libVkLayerGPUTimeline.so/data/local/tmp
        ```

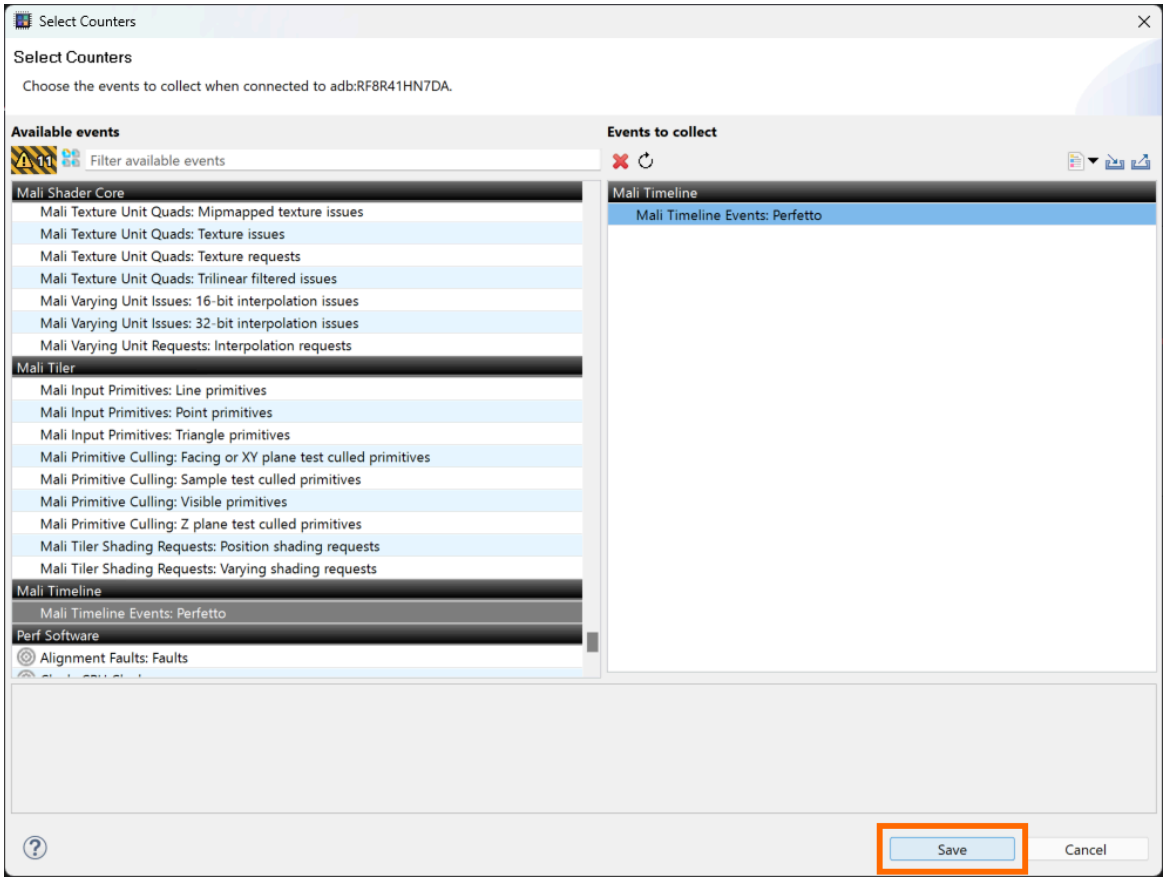
        Note The GPU Timeline layer is only available for Arm64.

2. Enable the Mali™ Timeline Events counter only if you are using advanced mode capture:
   a. In the **Start** view, select the **Use advanced mode** checkbox.
   b. Click **Select counters**.
   c. Drag the **Mali Timeline events: Perfetto** counter from the **Available events** column to the **Events to collect** column.

      Note
      - If many event counters are available for your device, you can use the search bar to filter the available events. In the search bar, search for **Mali Timeline**.
      - If the counter is not displayed, then your device does not support Mali™ Timeline events.

   d. Click **Save**.

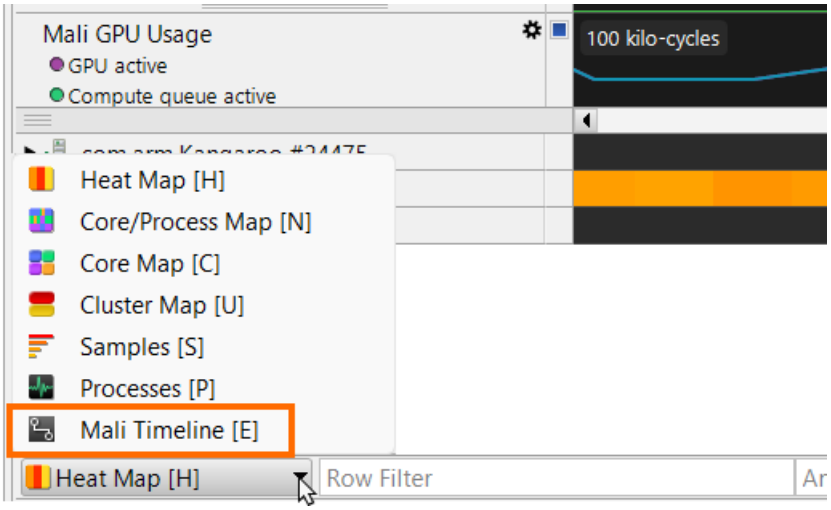
Figure 2-10: Mali Timeline Events: Perfetto under Mali Timeline



Next steps

1. Capture the Mali Timeline data. Click **Start capture**.
2. Open the **Mali Timeline** capture. In the Streamline GUI, open the **Timeline** view. In the details panel, located underneath the charts, open the mode menu, and select **Mali Timeline** mode.

Figure 2-11: Select Mali Timeline mode



Note

If your device does not support Mali™ Timeline Events, the **Mali Timeline** mode is not listed in the menu, and an error message displays in the **Error** view. The device requirements for capturing Mali™ Timeline Events are listed in the **Before you begin** section of this topic.

The **Mali Timeline** view opens which shows the device, process, graphics context, and a list of the queues that were present in your capture.

For more information about Mali™ Timeline, see Mali Timeline mode.

Related information

Enable Mali Timeline Events for headless capture on page 40
Mali Timeline mode on page 89

2.8.7 Enable Mali Timeline events for headless capture

Mali™ Timeline events present GPU queue scheduling information through a timeline view. This topic describes how to collect Mali™ Timeline events for a headless capture.

Before you begin

- Ensure your device supports Mali™ Timeline events.

  Mali™ Timeline events requires your device to:

  - Have a release build of Android 11 or later
  - Have a Mali™ device driver version r43p0 or later. To use the GPU Timeline layer driver, you must have a Mali™ device driver version r51p0 or later.
  - Be running the Perfetto service


Note

Mali™ Timeline events:

- Are not supported on Midgard architecture GPUs
- Support OpenCL data from version r43p0 for Android and Linux. If your application contains OpenCL content, select the **Mali Timeline** counter to show the events in the **Compute Queue** Custom Activity Map (CAM) track.

- Connect to your device.
- Linux users must manually set the environment variable `MALI_GPU_RENDERSTAGES_ENABLE=1`. See System requirements for Mali Timeline support in the Arm Streamline Target Setup Guide for Linux for more information.

About this task

You can use Mali™ Timeline Events to help you:

- Identify scheduling issues where the queues run serially, for all or part of the frame.
- Correlate events with render passes and compute dispatches.
- Understand what is happening in your application by using semantic information.

Procedure

1. Enable the GPU Timeline layer driver:
   - Pass `--gpu-timeline` as a value to the `-g` option on your `gatord` capture command line. The default value is `auto`, which automatically deploys and enables the layer driver only if you have enabled the `MaliTimeline_Perfetto` counter. Set to `yes` to deploy and enable the layer driver when you enable the `MaliTimeline_Perfetto` counter.
   - In `streamline_me.py` using the `--gpu-timeline` command line argument:
     - `Auto` automatically deploys and enables the layer driver only if you have enabled the `MaliTimeline_Perfetto` counter.
     - `Yes` deploys and enables the layer driver. To capture Mali Timeline events, you must also enable the `MaliTimeline_Perfetto` counter.

   Example of enabling the GPU Timeline layer driver in `streamline_me.py`:

   ```
   python streamline_me.py --headless <mali_timeline_capture.apc> --package
    <package_name> --headless-timeout 30 --config <configuration.xml> --lwi-mode
    counters  --gpu-timeline yes --overwrite
   ```

2. If required, specify the path for the GPU Timeline layer driver using `--gpu-timeline-layer-path`. The GPU Timeline layer `libVkLayerGPUTimeline.so` is deployed depending on the setting of the `--gpu-timeline` argument.
3. Enable the `MaliTimeline_Perfetto` counter:

- Add `MaliTimeline_Perfetto` to your `configuration.xml` file, then run `gatord` with `configuration_with_Perfetto.xml`. For example:

```
./gatord --android-pkg com.example1 --android-activity .ExampleActivity
 -c <configuration_with_Perfetto.xml>  MaliTimeline_Perfetto -g -o
 ExampleCapture.apc -t 5
```

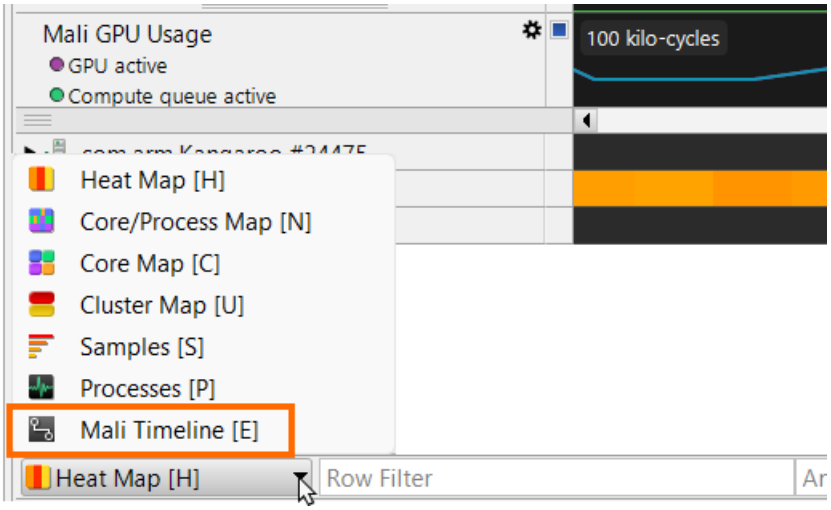
- Pass `MaliTimeline_Perfetto` as a value to the `-C` option on your `gatord` capture command line. For example:

```
./gatord --android-pkg com.example1 --android-activity .ExampleActivity -C
 MaliTimeline_Perfetto -g yes -o ExampleCapture.apc -t 5
```

Next steps

1. Capture the Mali Timeline data using the `gatord` capture command.
2. Open the **Mali Timeline** capture. In the Streamline GUI, open the **Timeline** view. In the details panel, located underneath the charts, open the mode menu, and select the **Mali Timeline** mode.

Figure 2-12: Select Mali Timeline mode



Note

If your device does not support Mali™ Timeline Events, the **Mali Timeline** mode is not listed in the menu, and an error message displays in the **Error** view. The device requirements for capturing Mali™ Timeline Events are listed in the **Before you begin** section of this topic.

The **Mali Timeline** view opens which shows the device, process, graphics context, and a list of the queues that were present in your capture.

For more information about Mali™ Timeline, see Mali Timeline mode.

Related information

Enable Mali Timeline Events using the Streamline GUI on page 37
Creating a configuration.xml file on page 119
Mali Timeline mode on page 89

2.9 Set up a local capture on a device using the Streamline GUI

Run `gatord` to capture data locally on a device when you are experiencing network problems, or when you are running headless Continuous Integration (CI) testing.

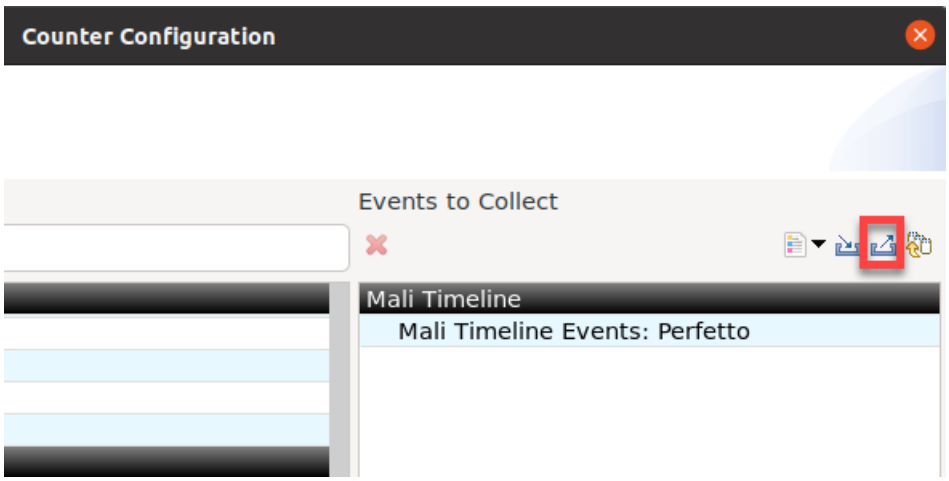
About this task

This topic explains how to set up a local capture on a device using the Streamline GUI. For instructions on how to set up a local capture on a device using the command-line, see Set up a local capture on a device using the command-line.

Procedure

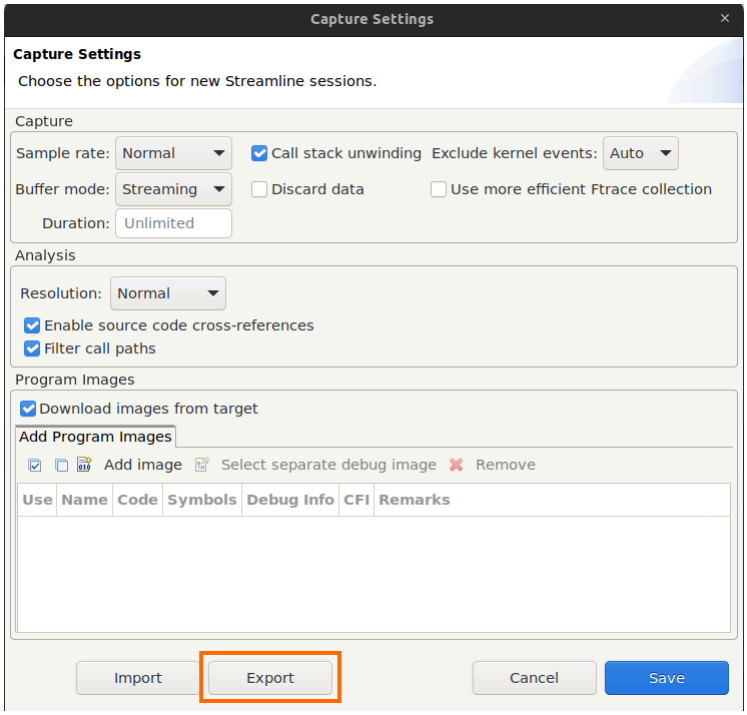
1. Export a `configuration.xml` file.
   a. In the **Start** view, select the **Use advanced mode** checkbox.
   b. Click **Select counters**.
   c. Click **Export** as shown in the following figure. For more information, see Importing and exporting counter configuration files.

Figure 2-13: Export a configuration.xml file from the Select Counters dialog box.



2. Export a `session.xml` file using the Streamline GUI.
   a. In the **Start** view, select the **Use advanced mode** checkbox.
   b. Click **Capture Settings**.
   c. Click **Export** to export the `session.xml`, as shown in the following figure:

Figure 2-14: Export a session.xml file from the Capture Settings dialog box.



3. Run `gatord` with your `configuration.xml` and `session.xml` files, and include the appropriate options for root or non-root devices. This must be done on the command-line:

```
./gatord -s session.xml -c configuration.xml <options> -o <capture-filename>.apc
```

Results

`gatord` captures a profile of all the processes running on your device, and writes the data to the specified location.

Next steps

Copy the `.apc` directory into the Streamline capture directory. This directory is usually `<Home>/Documents/Streamline`.

Related information

Set up a local capture on a device using the command-line on page 44
Importing and exporting counter configuration files on page 33
Gatord command line options (Android targets)
Gatord command line options (Linux targets)

2.10 Set up a local capture on a device using the command-line

Run `gatord` on the command-line to capture data locally on a device when you are experiencing network problems, or when you are running headless Continuous Integration (CI) testing.

About this task

This topic explains how to set up a local capture on a device using the command-line. For instructions on how to set up a local capture on a device using the Streamline GUI, see Set up a local capture on a device using the Streamline GUI.

Procedure

1. Create a `configuration.xml` file.
   Modify an existing `configuration.xml`, or create a new one. To learn more about creating a new `configuration.xml` file, see Creating a configuration.xml file.
2. Create the `session.xml` file with your required arguments. For example:

```
<session version="2" call_stack_unwinding="yes" parse_debug_info="yes"
 resolution_mode="normal" buffer_mode="streaming" sample_rate="normal"
 duration="0" target_address="x.x.x.x" live_rate="100" stop_gator="no">
   <image path="<path/to/image>"/>
</session>
```

   You can set the following arguments:

   `call_stack_unwinding="yes|no"`

   Records call stacks to increase the amount of raw data that the device sends to the host.

   To support call stack unwinding for binaries that are created using GCC or LLVM, enable frame pointers using the `-fno-omit-frame-pointer` compiler option.

   For Arm®v7 software applications, you must also use the `-marm` option to disable the Thumb® instruction set.

   `parse_debug_info="yes|no"`

   Processes DWARF debug information and line numbers for a more detail in your capture session.

   To process debug information, you must have built the image using the `-g` compiler option.

   `resolution_mode="summary|normal|high|ultrahigh"`

   Controls the resolution used to analyze and visualize the data after it has been captured.

   Higher resolutions allow you to view data with a finer time granularity, but result in longer analysis times and higher memory usage.

   `buffer_mode="large|medium|small|streaming"`

   - `large`: 16M

- `medium`: 4M
- `small`: 1M
- `streaming`: Enables Live view and streaming of device data directly to your host during online capture.

`sample_rate="high|low|none"`

- `high`: Uses a sample frequency of 10kHz and enhances visibility, but also incurs an increase in overhead.
- `low`: Uses a sample frequency of 100Hz. Use `low` if the device is slow or heavily loaded.
- `none`: Disables periodic sampling. Use `none` if you have the lowest level of intrusion on your code. Any report columns that rely on sampling show zero values.


Note

`none` still samples on context switches.

`duration=""`

To set a duration for the capture, enter the time that you want the capture session to last, either in seconds or in minutes and seconds.

For example, use `60` to set a capture session of 60 seconds, or use `1:20` to set a capture session of 1 minute and 20 seconds.

If you do not enter a duration, the capture session continues until you stop it manually.

If you do not specify `buffer_mode="streaming"`, the capture session continues until the buffer is full.

`use_efficient_ftrace="yes|no"`

The Live view does not update charts that use `Ftrace` data, but the data is in the analyzed report.

3. Run `gatord` with your `configuration.xml` and `session.xml` files, and include the appropriate options for root or non-root devices:

```
./gatord -s session.xml -c configuration.xml <options> -o <capture-filename>.apc
```

Results

`gatord` captures a profile of all the processes running on your device, and writes the data to the specified location.

Next steps

Copy the `.apc` directory into the Streamline capture directory. This directory is usually `<Home>/<User>/Documents/Streamline`.

Related information

Set up a local capture on a device using the Streamline GUI on page 43
Creating a configuration.xml file on page 119
Gatord command line options (Android targets)
Gatord command line options (Linux targets)

2.11 Overview of CPU profiling

In Streamline, you can capture and analyze data from up to 6 CPU-related performance counters to help you identify inefficient functions in your code. Use the **Call Paths** and **Functions** views in Streamline to find your most expensive functions, and identify ways to optimize them.

To get a comprehensive view of how your functions behave, it is useful to take 2 captures. One using the regular time-based capture method, and one using the event-based capture method, using an event such as branch mispredictions, to sample the data. You can then compare the results to see if the functions that took the most time also caused a large number of branch mispredictions.

A time-based capture shows the percentage of time a function uses. An event-based capture shows how many times the event sample occurred in that function, as a percentage of all samples. Ideally, these 2 percentages would be almost equal. If the percentages are significantly different, it might mean that the code is triggering a microarchitecture behavior that is inefficient. For example, a function that is 5% of runtime in the time-based profile and 15% of branch mispredicts in the event-based profile. This indicates that the function has a disproportionally high number of branch mispredicts relative to its runtime. Modify the code or data to improve predictability or to avoid the branch completely, should improve performance.

Capturing data for CPU profiling

For Streamline to display your code, you must provide debug symbol information together with your application. Some game engines allow you to embed this information in the APK, and other game engines export these files separately.

You can capture data from either:

- A debuggable Android APK with associated debug symbol files exported from a game engine or Android Studio:
  - Install a debuggable version of the Android APK on your device and follow the normal Streamline capture process. However, you must add the debug symbol files in Streamline so that you can see the function names in the call paths and functions views.
- A native binary of an application:
  1. Use Android Debug Bridge (adb) to install both the application and `gatord` on the device, then set permissions to make them executable.
  2. Run `gatord` and specify the application together with any input files it requires.
  3. Open Streamline and connect to the device using TCP.

Related information

- Starting a capture

3. Annotate your code

Add event annotations and software-generated counters to your code, which are propagated into the **Timeline** view and **Log** view.

Annotation macros are defined in `streamline_annotate.h` and `gator_annotate.h`.


Note

Examples of annotations are available in `<install_directory>/streamline/examples/annotations/`. See the `readme` file in the same directory for more information.

There are two types of annotations, user space annotations and kernel space annotations:

User space annotations

User space annotations are for annotating your application. Streamline supports the following types of user space annotations:

- String
- Visual
- Marker
- Custom counters
- Groups and channels
- Custom Activity Maps (CAM)

Kernel space annotations

Kernel annotations send annotations to `gatord` from a kernel context, using Linux ftrace as the transport mechanism. Kernel annotations are defined in `<install_directory>/sw/streamline/gator/trace/event/gator_annotate.h`.

3.1 View annotations

You can view code annotations in the **Timeline** and **Log** views. The annotation type determines where the annotations are presented.

Timeline view

Annotations appear in several parts of the **Timeline** view:

- **String** annotations appear as an annotation on the thread that generated the annotation
- **Visual** annotations appear in their own chart
- **Marker** annotations appear as bookmark icons above the charts
- **Custom counters** annotations appear in their own chart

- **Groups and channel** annotations appear as an annotation on the thread that generated the annotation
- **Custom Activity Map (CAM) counter** annotations appear in their own CAM view

Streamline can show details of annotations within a specified period:

1. Drag each end of the cross-section marker so that it includes the period that you are interested in.
2. Click the **Select annotation log closest to the cross-section marker index** . Alternatively, press the L key.

   The **Log** view opens. All annotations within the selected period are highlighted.

Log view

The **Log** view lists all the annotations in the capture in chronological order. It also gives the thread and process that generated the annotation. If you assigned the annotation to a group and channel, the group and channel numbers are given. If you did not assign it to a group and channel, the **Log** view displays Group 0 and Channel 0 in these columns.

Marker annotations are indicated with a bookmark icon . Similarly, visual annotations are indicated with a camera icon . To see the image for a visual annotation, select the row for that annotation.

To filter the annotations, use the fields at the top of the view. You can enter regular expressions in these fields, except for the **When** and **Duration** fields. Use a dash to enter a range of values in the **When** and **Duration** fields. For example, 4-5 for four to five seconds inclusive, -5 for up to five seconds, or 5- for five seconds or later.

To see annotations in the **Timeline** view, select the annotations that you are interested in, right-click, and select the appropriate option. The **Timeline** view opens with the cross-section marker covering the selected annotations. To see functions that are related to annotations, select the annotations in the **Log** view, right-click, then select **Select in Call Paths**. The **Call Paths** view opens with the related functions selected.

Related information

- User space annotations
- Kernel space annotations
- Add annotations to your code

3.2 User space annotations

User space annotations are for annotating your application. This section describes the user space annotations Streamline supports.

Before you can use any user space annotations, insert the `ANNOTATE_SETUP` macro in your code to set up annotations:

`ANNOTATE_SETUP`
: Set up annotations. Call this macro before calling any other annotate macros.

3.2.1 String annotations

This topic describes the string annotations supported by Streamline.

String annotations write user-defined strings into the **Log** view and place framing overlays in the details panel in the **Timeline** view.

Annotation macros

`ANNOTATE(string)`
: Add a string annotation. Streamline adds the annotation to channel 0.

`ANNOTATE_COLOR(color, string)`
: Add a string annotation with a display colour. See `streamline_annotate.h` for the defined colors.

`ANNOTATE_END()`
: Terminate the annotation in channel 0.

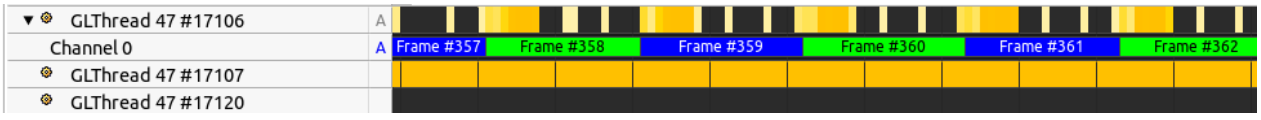
Example 3-1: Example

The following example uses the `ANNOTATE_COLOR` string annotation to add a blue and green framing overlays in your capture timeline:

```
ANNOTATE_COLOR(ANNOTATE_BLUE, label);
ANNOTATE_COLOR(ANNOTATE_GREEN, label);
```

The blue and green overlays render in your capture timeline as:

Figure 3-1: String annotations in example capture



Note

The ANNOTATE_COLOR annotation is a simplified interface for the ANNOTATE_CHANNEL_COLOR group and channel annotation (see Group and Channel annotations), but uses a default channel and group.

Related information

View annotations on page 49
Add annotations to your code on page 62

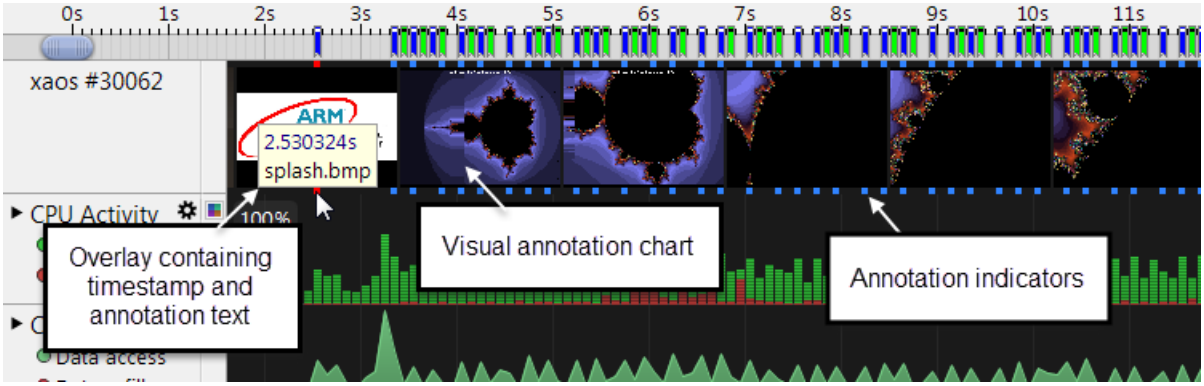
3.2.2 Visual annotations

This topic describes the visual annotations supported by Streamline.

Visual annotations add images to the visual annotation chart in the **Timeline** view. The images are also displayed in the **Log** view.

If you add images to the capture data, they appear in a chart in the **Timeline** view. This chart enables you to track the visuals of your application with the other chart data.

Figure 3-2: Visual annotation in the Timeline view



Visual annotation charts have the following features:

- If there is more than one image in the time period that a thumbnail covers, the first image from the range is displayed. 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 indicators at the top and bottom of the chart identify the time bins in which your code produced each image. Yellow indicators indicate that the annotation contains text in addition to the image. Blue indicators indicate that there is no additional text.
- If you hover over an image in the **Timeline** view, the indicators at the cursor position turn red. The red indicators identify the time bin of the image currently being displayed. Hovering over a red indicator displays an overlay that contains the timestamp and, if present, the text annotation associated with the image.

- Clicking an image switches the Details panel to **Images** mode and shows an enlarged version of the image. To view an animation of the images in the details panel, click and drag the mouse to the left or right over the image.


Note

Generating a visual annotation is expensive because of the CPU overhead of creating the image and the bandwidth overhead of transferring the image off the device. To avoid adversely impacting the performance of the application being profiled, Arm recommends that you rate limit the creation of visual annotations.

Images for visual annotations can be in the following formats:

- GIF
- PNG
- JPEG
- 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 that is sent to the host in this way increases the probe effect for the applications you are profiling.

Annotation macros

`ANNOTATE_VISUAL(data, length, str)`

Record an annotation that includes an image in one of the following formats:

- GIF
- PNG
- JPEG
- ICO
- BMP +RLE

Specify the image in `data`, the amount of data to write to the annotate file in `length`, and optionally a descriptive string to include with the image in `str`.

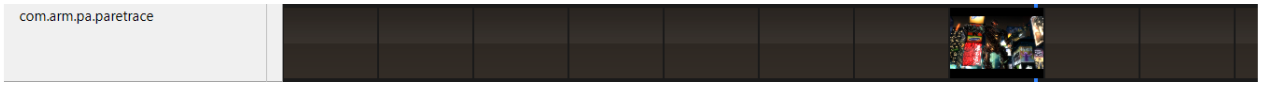
Example 3-2: Example

The following example uses the `ANNOTATE_VISUAL` visual annotation to display an image in a track:

```
std::vector<unsigned char> png_data; // Assume this is populated
ANNOTATE_VISUAL(png_data.data(), png_data.size(), nullptr);
```

Which renders the image in the track as:

Figure 3-3: Image from visual annotation in example capture



Related information

View annotations on page 49
Add annotations to your code on page 62

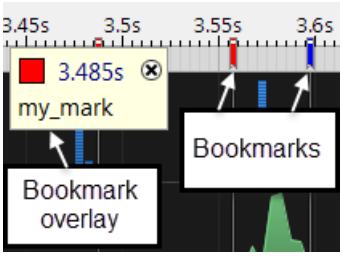
3.2.3 Marker annotations

This topic describes the marker annotations supported by Streamline.

Marker annotations add bookmarks to the **Timeline** view and the **Log** view, optionally with a text string. Bookmarks enable you to label and quickly return to points of interest.

Hovering over a bookmark displays an overlay that contains the timestamp and text string of the bookmark.

Figure 3-4: Bookmarks



Annotation macros

`ANNOTATE_MARKER()`

Add a bookmark to the **Timeline** view and the **Log** view.

`ANNOTATE_MARKER_STR(string)`

Add a bookmark with a string to the **Timeline** view and the **Log** view. Streamline displays the string when you hover over the bookmark in the **Timeline** view, and in the **Message** column in the **Log** view.

`ANNOTATE_MARKER_COLOR(color)`

Add a bookmark with a color to the **Timeline** view and the **Log** view.

`ANNOTATE_MARKER_COLOR_STR(color, string)`

Add a bookmark with a color and a string to the **Timeline** view and the **Log** view.

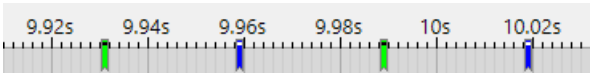
Example 3-3: Example

The following example shows uses the `ANNOTATE_MARKER_COLOR_STR` annotation to add blue and green markers annotations to your capture:

```
ANNOTATE_MARKER_COLOR_STR(ANNOTATE_BLUE, label);
ANNOTATE_MARKER_COLOR_STR(ANNOTATE_GREEN, label);
```

Which renders the blue and green markers in the capture timeline:

Figure 3-5: Marker annotations in example capture



Related information

View annotations on page 49
Add annotations to your code on page 62

3.2.4 Custom counter annotations

Application software can use counter annotations to generate custom data series that are displayed as charts in the **Timeline** view. Streamline automatically displays a chart for each custom counter that an application creates.

Each custom counter is presented as a single data series. For each counter, the implementation tracks an independent value for each application thread. The per-thread values are accumulated into a single series value, which is displayed in the **Timeline** view. Like hardware counters, you can filter custom counters using the **Details panel** to select which threads to include in the accumulated value.

Annotation macros

The following macros define integer counters, where the numeric value is in the range 0...1<<63-1:

`ANNOTATE_DELTA_COUNTER(uid, title, name)`

Define a custom delta counter. Specify:

`uid`

An integer that uniquely identifies the counter in the application process.

`title`

A string for the chart title.

`name`

A string for the series name.

`ANNOTATE_ABSOLUTE_COUNTER(uid, title, name)`

Define a custom absolute counter. The parameters are the same as for `ANNOTATE_DELTA_COUNTER`.

ANNOTATE_COUNTER_VALUE(uid, value)

Emit an integer value for a custom delta or absolute counter. `uid` identifies the counter.

The following macros define fractional (float or double) counters:


Note

The float value is converted to an integer value in the above range using some fixed multiplier, so for example to send a float value in the range 0.000...1.000, with 3-decimal places accuracy, use a modifier value of 1000.

ANNOTATE_DELTA_COUNTER_SCALE(uid, title, name, modifier)

Define a custom fixed-point delta counter with a scaling modifier. Specify:

uid

An integer that uniquely identifies the counter.

title

A string for the chart title.

name

A string for the series name.

modifier

An integer scaling factor.

Floating point counter samples reported through `ANNOTATE_COUNTER_VALUE_SCALE` are encoded as `(int)(sample * modifier)` for transport, and converted back to unscaled float for presentation. For example, to present a float with two decimal places of precision, set `modifier` to 100.

ANNOTATE_ABSOLUTE_COUNTER_SCALE(uid, title, name, modifier)

Define a custom absolute counter with a scaling modifier. The parameters are the same as for `ANNOTATE_DELTA_COUNTER_SCALE`.

ANNOTATE_COUNTER_VALUE_SCALE(uid, value, modifier)

Emit a value for a custom scaled delta or absolute counter. `uid` identifies the counter and `value` specifies a float value. The scaling value `modifier` must be an integer with the same value as the modifier that was used when the scaled counter was declared.

Example 3-4: Example

The following example uses the `ANNOTATE_ABSOLUTE_COUNTER` and `ANNOTATE_COUNTER_VALUE` custom counter annotations to create a custom “Draw Calls / Frame” chart:

```
// Setup code
ANNOTATE_ABSOLUTE_COUNTER(draw_chart_id, "Draw Calls / Frame", context_label);

// Counter generation code
ANNOTATE_COUNTER_VALUE(draw_chart_id, draw_call_count);
```

Which enables the custom “Draw Calls / Frame” graph to be displayed in your capture timeline:

Figure 3-6: Custom counter annotation example capture

Note

To view a list of the graphs available, including custom graphs created using the custom counter annotations, select the **Add charts with default settings...** button. The available graphs are listed in the drop-down.

Related information

View annotations on page 49
Add annotations to your code on page 62

3.2.5 Group and Channel annotations

This topic describes the group and channel annotations supported by Streamline.

The group and channel 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** view and the **Log** view. 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.

Channels and groups are defined per thread. This means that if the same channel number is used on different threads they are separate channels. A channel can belong to only one group per thread. This means channel 1 cannot be part of both group 1 and group 2 on the same thread.

Annotation macros

`ANNOTATE_CHANNEL(channel, string)`
: Add a string annotation and assign it to a channel. The `channel` parameter is the channel number.

`ANNOTATE_CHANNEL_COLOR(channel, color, string)`
: Add a string annotation with a display colour, and assign it to a channel.

`ANNOTATE_CHANNEL_END(channel)`
: Terminate the annotation in the given channel.

`ANNOTATE_NAME_CHANNEL(channel, group, string)`
: Define a channel and attach it to an existing group. The channel number, `channel`, must be unique within the thread.

`ANNOTATE_NAME_GROUP(group, string)`
: Define an annotation group. The group identifier (`group`) must be unique within the thread.

Related information

View annotations on page 49
Add annotations to your code on page 62

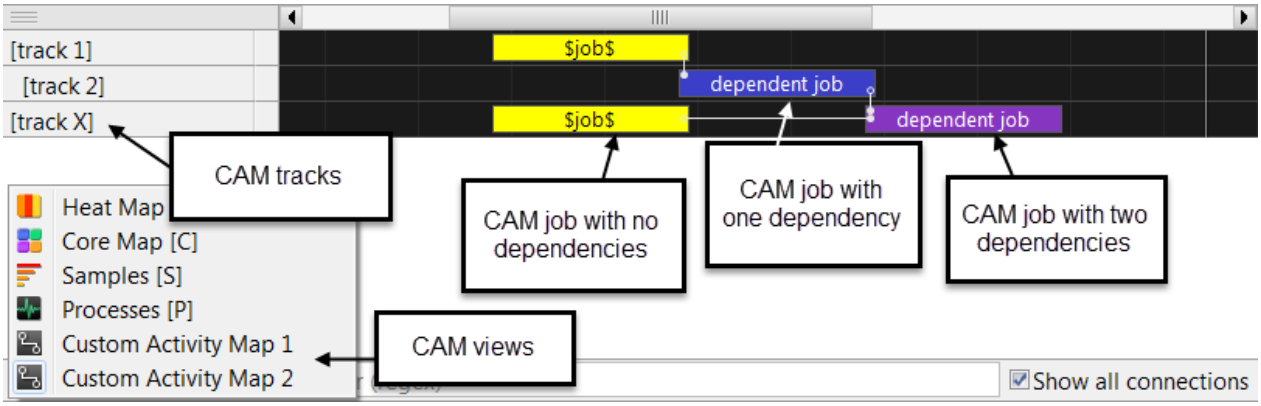
3.2.6 Custom Activity Map annotations

This topic describes the Custom Activity Map (CAM) annotations supported by Streamline.

CAM annotations allow you to define and visualize the execution of a complex dependency chain of jobs. Each CAM view contains one or more tracks and each track contains one or more jobs. Tracks can also be nested hierarchically.

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.

Figure 3-7: Custom Activity Maps



Annotation macros

`CAM_VIEW_NAME(view_uid, name)`

Create a named CAM view. `view_uid` must be unique in the application process.

`CAM_TRACK(view_uid, track_uid, parent_track, name)`

Add a track to a CAM view. To make the track a child of another track, specify the parent track ID in `parent_track`. To make it a root track, set `parent_track` to -1. `track_uid` must be unique within the view.

`CAM_JOB(view_uid, job_uid, name, track, start_time, duration, color)`

Add 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. To get the current timestamp in nanoseconds, use `gator_get_time()`, which is declared in `streamline_annotate.h`.

CAM_JOB_DEPS(view_uid, job_uid, name, track, start_time, duration, color, dependency_count, dependencies)

Add 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_DEP(view_uid, job_uid, name, track, start_time, duration, color, dependency)

The single-dependency version of `CAM_JOB_DEPS`.

CAM_JOB_SET_DEPS(view_uid, job_uid, time, dependency_count, dependencies)

Set 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_SET_DEP(view_uid, job_uid, time, dependency)

The single-dependency version of `CAM_JOB_SET_DEPS`.

CAM_JOB_START(view_uid, job_uid, name, track, time, color)

Add a job with a start time to a CAM track. End the job using `CAM_JOB_STOP`.

CAM_JOB_STOP(view_uid, job_uid, time)

End a CAM job that was previously started using `CAM_JOB_START`.

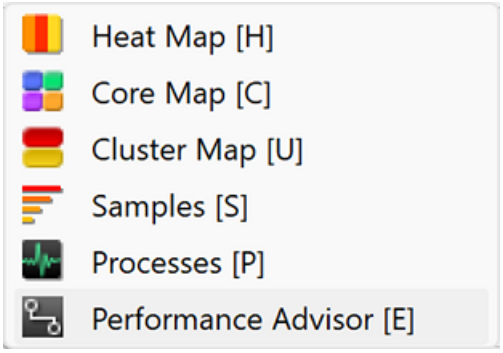
Example 3-5: Example

The following code is an example that uses the `CAM_VIEW_NAME`, `CAM_TRACK`, `CAM_JOB`, `CAM_START`, and `CAM_STOP` annotations:

```
// Setup
cam_view = cam_next_id++;
CAM_VIEW_NAME(cam_view, "Performance Advisor");

cam_track_thread_context = cam_next_id++;
CAM_TRACK(cam_view, cam_track_thread_context, -1, "Active Context");

cam_thread_track = cam_next_id++;
CAM_TRACK(cam_view, cam_track, cam_track_thread_context, thread_label);

// Per frame
if (last_job)
{
        CAM_JOB_STOP(cam_view, last_job, gator_get_time());
}

last_job++
CAM_JOB_START(cam_view, last_job, job_name, cam_thread_track, gator_get_time(),
 ANNOTATE_LTGRAY);
```

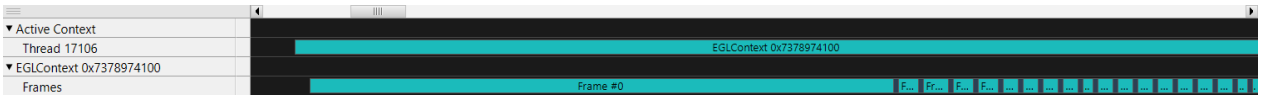
The example code adds the **Performance Advisor** CAM view to the details panel mode:

Figure 3-8: Example CAM annotations mode



To view the tracks, select the **Performance Advisor** CAM view, and select a high zoom:

Figure 3-9: Example CAM annotations tracks view at high zoom



Related information

View annotations on page 49
Add annotations to your code on page 62

3.3 Kernel space annotations

Kernel annotations are for annotating kernel code or kernel modules, and allow you to profile any system calls. Streamline supports string, marker, and custom counter kernel annotations. For more information about how to use kernel annotations, see `README.md` and the annotation examples located in `<install_directory>/streamline/gator/trace/events`:

- `example_standalone`, where annotations are added in the same module
- `example_shared`, where annotations are exported from a different module

Warning

`gator_annotate.h` and `and gator_annotate.c` are intended for use in a development environment. Arm does not recommend including these files in a released product without performing a security audit of the source code first.

Note

Examples of annotations are available in `<install_directory>/streamline/examples/annotations/`. See the `readme` file in the same directory for further information.

Annotation macros

Use the following macros to send annotations to `gatord` from a kernel context, using Linux ftrace as the transport mechanism. They are defined in `<install_directory>/streamline/gator/trace/event/gator_annotate.h`.

The following macros indicate that an event is associated with the whole kernel:

`GATOR_KERNEL_WIDE_PID`
: For bookmark and counter TRACE_EVENTS only. To indicate that the event is associqted with the whole kernel rather than a specific `tid`, you can pass this value as the `tid`. Annotation texts must have a `tid` of at least zero. `tid == 0` is the idle process. This value can used only for bookmark and counter TRACE_EVENTS.

The following macros create and configure gator bookmark annotations:

`GATOR_BOOKMARK(label)`
: Add a bookmark with a label.

`GATOR_BOOKMARK_COLOR(color, label)`
: Add a bookmark with a color and a label.

The following macros start (`START`) and stop (`STOP`) kernel space function annotations:


Note

`START` and `STOP` macros enable you to call the same function but offer a distinction between when the text annotation is intended to be the start of a text annotation, or the end.

`GATOR_TEXT_START(tid, channel, label)`
: Start a text annotation.

`GATOR_TEXT_START_COLOR(tid, color, channel, label)`
: Start a text annotation with a color.

`GATOR_TEXT_STOP(tid, channel)`
: End a text annotation. This macro sends an empty string as the label, which Streamline recognizes as the end of a text annotation with the given channel and `tid`.

The following macros output counter values:

`GATOR_DELTA_COUNTER_VALUE(title, name, units, value)`
: Output delta counter values.

`GATOR_ABSOLUTE_COUNTER_VALUE(title, name, units, value)`
: Output absolute counter values.

Related information

- View annotations
- Add annotations to your code

3.4 Add annotations to your code

You can add annotation macros to your code to add various types of visual indicators to your application analysis report, or to break down threads of activity into custom groups and channels.

Before you begin

Do one of the following:

- Include the files `streamline_annotate.h`, `streamline_annotate.c`, and `streamline_annotate_logging.h` in your project.
- Build a `libstreamline_annotate` library with the Makefile in `<install_directory>/streamline/gator/annotate`, and include this library in your project.

About this task

Annotations are a useful way to add application-aware semantic information about application execution, which might otherwise not be visible to the profiler.

Note

The CPU overhead of emitting annotations is relatively expensive. Arm recommends that you emit fewer than 10000 annotations per second.

Annotations can be seen in the **Timeline** and **Log** views in your analysis report.

Procedure

1. Add the `ANNOTATE_SETUP` macro to your code.
   This macro must be called before any other annotate macros are called.
2. Add annotations to your code using the macros defined in `streamline_annotate.h`. For a list of supported annotations, see Annotate your code.
3. Build your code.
4. Ensure `gatord` is running and run your application. Execute the area of the code that emits the annotations.

Results

These annotations appear in the **Timeline** view and the **Log** view.

For a more information about what to expect in each of the **Timeline** and **Log** views, see the information per annotation type in View annotations.

Example 3-6: Adding annotations to your code

The following examples demonstrate how to use string, visual, or channel and group annotations to your code.

- Example 1: Add a string annotation to your code

String annotations write user-defined strings into the **Log** view and place framing overlays in the details panel in the **Timeline** view.

This example uses the `ANNOTATE_COLOR` string annotation macro to add a colored annotation with a textual string.

1. Add the `ANNOTATE_SETUP` macro to your code.
2. To add a purple annotation with the string “render”, use the `ANNOTATE_COLOR(ANNOTATE_PURPLE, "render")` annotation macro.

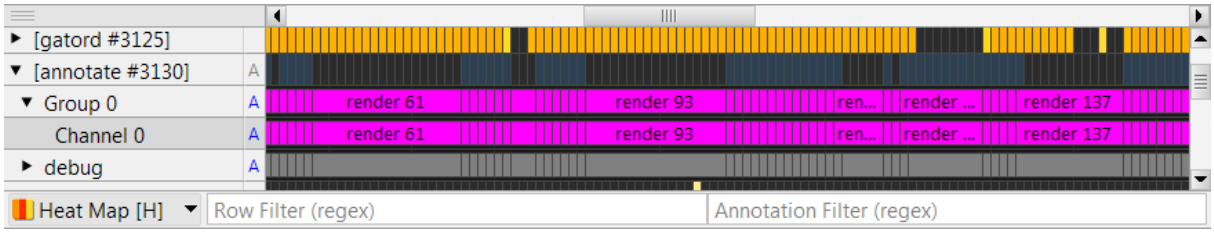
   Note

   `ANNOTATE_COLOR(color, string)` supports color constants that are defined in `streamline_annotate.h` or ASCII escape codes followed by a 3-byte RGB value.

3. Use `ANNOTATE_END()` to clear the annotation message for the thread.
4. Build your code.
5. Ensure `gatord` is running and run your application. Exercise the area of the code that emits the `ANNOTATE_COLOR` annotation.

String annotations are displayed as text overlays inside the relevant channels in the details panel of the **Timeline** view, see inside **Channel 0** in the following screenshot:

Figure 3-10: String annotation overlays.



The letter **A** displays in the process list to indicate the presence of annotations. String annotations are also displayed in the Message column in the **Log** view.

- Example 2: Add a visual annotation to your code

  You can add images to the **Timeline** view and the **Log** view in your analysis report by adding macros to your code.

  This example records an annotation that includes an image. the example uses the `ANNOTATE_VISUAL` visual annotation.

  1. Add the `ANNOTATE_SETUP` macro to your code.
  2. To insert the ‘image1' image with the string “My image 1” as a visual annotation to your code, use `ANNOTATE_VISUAL(image1, sizeof(image1), "My image 1")`.

     Where:

- `image1` is the data (your image)
- `sizeof(image1)` is the size of the `image1` being written to the annotate file
- `My image 1` is an optional descriptive string to be included with `image1`

3. Build your code.
4. Ensure `gatord` is running and run your application. Exercise the area of the code that emits the annotations.

The **Timeline** view now includes a chart that displays 'image1'. In the **Log** view, any annotation event that includes an image has a camera icon in the message field. To see the image, select the row containing the camera icon.

- Example 3: Add a group or channel annotation to your code

  Group and channel macros enable 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 group and channel has a name for display in the **Timeline** view and the **Log** view.

  This example uses the `ANNOTATE_NAME_GROUP` annotation to name a group, and uses the `ANNOTATE_NAME_CHANNEL` to name a channel and link it to a group.

  1. Add the `ANNOTATE_SETUP` macro to your code.
  2. To name group '1' as "DEBUG", insert `ANNOTATE_NAME_GROUP(1, "DEBUG")` macro into your code.

     Note

     Groups can be named after use.

  3. Use `ANNOTATE_END()` to clear the annotation message for the group.
  4. To name channel '1' as "DisplayImage", and link it to the "DEBUG" group insert the `ANNOTATE_NAME_CHANNEL(1, 1, "DisplayImage")` macro into your code.

     Note

     - Channels can be named after use
     - A channel can belong to only one group per thread. In other words, channel 1 cannot be part of both group 1 and group 2 on the same thread

  5. Use `ANNOTATE_CHANNEL_END(1)` to clear the annotation message for the thread.
  6. Build your code.
  7. Ensure `gatord` is running and run your application. Exercise the area of the code that emits the annotations.

Timeline view and the **Log** view show the “DisplayImage” channel, assigned to the “DEBUG” group.

Related information

User space annotations on page 50
Kernel space annotations on page 60
View annotations on page 49

4. Analyze your capture

When you have captured a Streamline profile of your application, view the data in the **Timeline** view.

4.1 Timeline overview

The first view in Streamline is the **Timeline** view, which shows the captured performance data on a time-based visualization. You can zoom and scroll this view to show regions of interest.

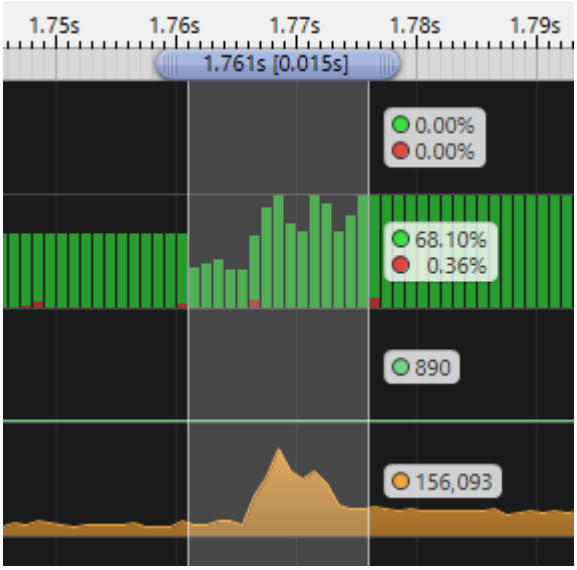
A hierarchical data model stores the captured data at multiple resolutions that match the zoom levels available in the **Timeline** view. If you zoom in beyond the data capture sample frequency, the presented data is interpolated between the nearest data samples.

Hardware counters are sampled, by default, using a 1 millisecond sample rate. Other data sources are event-based and are captured and timestamped as they occur. Counter samples or event-based counters that occur less often than the 1 millisecond level must be mapped to the 1 millisecond data view for presentation. This mapping depends on the type of data being shown. See Counter classes for more details.

Cross-section marker

You can use the cross-section marker to highlight a time interval and show a summary of the counter statistics for the highlighted region. The summary shows a sum or an average per series, depending on the data source, see Chart configuration series options .

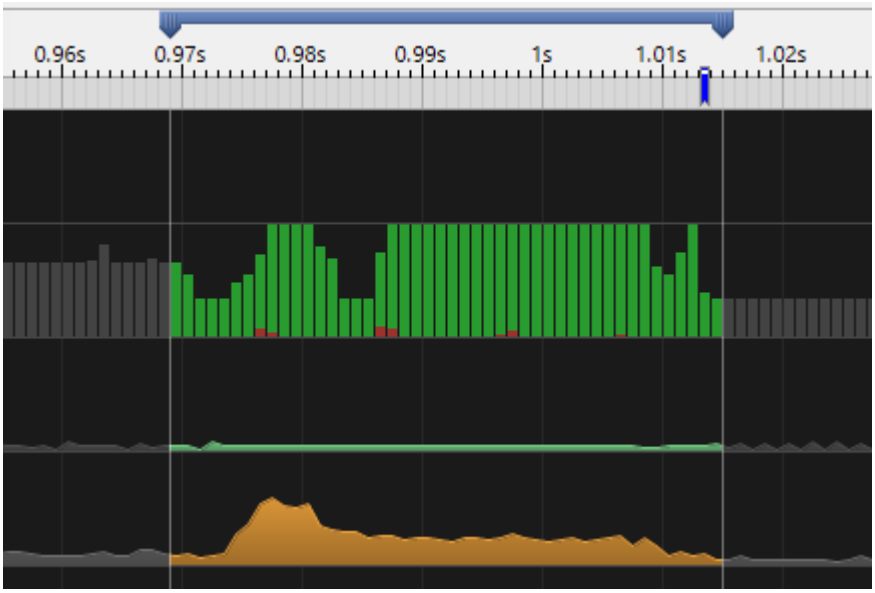
Figure 4-1: The cross-section marker



Calipers

You can use the calipers to filter the data range that is provided to the other analysis visualizations, such as the **Call Paths** view. When you select a region of interest with the calipers in the **Timeline** view, the other views only consider that region.

Figure 4-2: The calipers



Note

You can also filter by process or thread by clicking the processes or threads of interest in the **Timeline** view.

Activity and Counters charts

There are two main chart types that the **Timeline** view shows by default. By convention we call these 'Activity charts' and 'Counter charts', but you can customize the presentation and these names are not enforced.

Activity charts are generated from kernel or driver scheduling information. They show the percentage of time, at the current hierarchy level, that the indicated hardware unit was busy. Note that high utilization is not always the same as high load. A processor might run all the time, but at a low operating frequency due to low workload demand.

Counter-based charts are generated from actual hardware activity. For example, counting the number of instructions executed by a processor, or the number of cycles a block was active. These charts give a more direct measure of the workload, but do not indicate how long the processor was actually running for.

Combining both activity and counter-based charts allows a view of execution time and workload, both of which are needed for a complete performance analysis.

Expressions

To supplement the raw data that is captured from the hardware, you can use expressions to define custom data series. Expressions are simply equations, defined in a light-weight macro syntax, which define a derived data point to plot. For example, you can use expressions to scale a 'bus beats' counter into a 'bus bytes' counter, or to generate a hit ratio from cache hit and miss event counts.

Data points for expressions are re-evaluated for each zoom level in the timeline to ensure correct results for equations that are not distributive.

Templates

Templates store the current **Timeline** view layout - charts, series, and styling - as a reusable visualization that you can apply to other data captures. You can customize your **Timeline** view and save it as a template for future use, or for sharing with other team members.


Note

- Templates are designed to work with captures made with the current version of Streamline. Applying a template to a capture made with an older version of Streamline might result in some series being omitted. If a data series uses a source counter which has changed name since the capture was made, it is omitted.
- When you configure your counters, you can select a template to automatically collect data from all the sources that the template requires.

Built-in templates are included for most of the Arm® Mali™ GPU family. These templates do not include every Mali™ data source, but are the recommended starting point for Mali™ GPU profiling. For detailed descriptions of all the performance counters available for each Mali™ GPU, refer to the Mali GPU Counter references on the Arm Developer website.

4.2 Arm NN Timeline

This topic describes the Arm NN Timeline in Streamline.

Arm NN is an open source Machine Learning (ML) inference engine for Android and Linux which accelerates ML on Arm CPUs and GPUs.

If you are using Streamline with Arm NN and timeline profiling is enabled there is an Arm NN Timeline view for running process that uses Arm NN. Each row in the view is a neural network, which you can expand to see the layers of the neural network. Expanding a layer shows the workloads in that layer. Workload names are in the form `workload:<Backend>(<GUID>)`, where `<Backend>` is the hardware or mechanism responsible for the workload, and `<GUID>` is a Globally Unique Identifier (GUID) provided by Arm NN. The jobs on each row indicate how long each inference of the network, each layer, or each workload ran for.

Related information

Arm NN GitHub

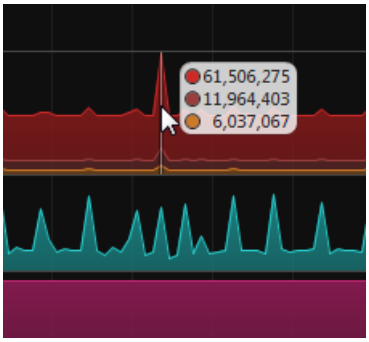
View annotations on page 49

Add annotations to your code on page 62

4.3 Working with charts

Hover over a chart to see the values at that point on the timeline.

Figure 4-3: Values at a specific point on a timeline.



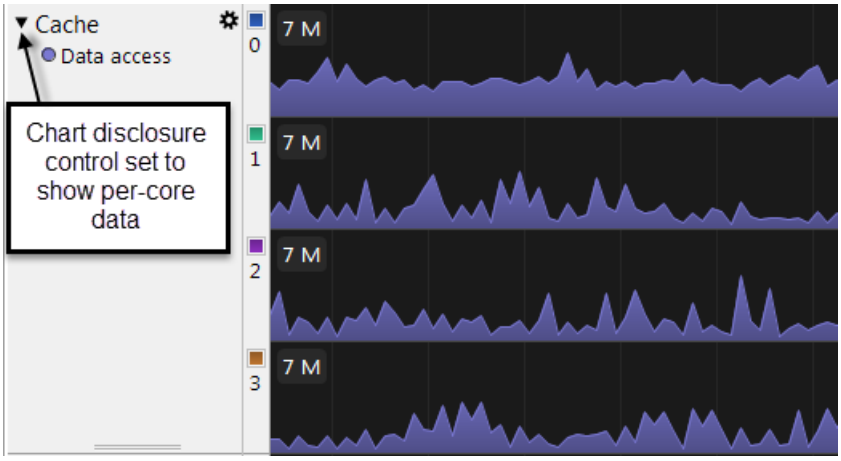
Increase the size of a chart to see a higher level of graphical detail, making it easier to see the variance in values. Resize a chart by clicking and dragging the handle control at the bottom of the box that displays the chart title.

Figure 4-4: Control for resizing the chart.

You can reorder charts by clicking elsewhere within the box and dragging the chart to the new location.

If you have collected data from a target with multiple cores, the chart shows an aggregate of all cores. Expand the chart name to see the per-core data, and hover over a core number to see the name of the core.

Figure 4-5: Chart disclosure control set to show per-core data.



The charts scale according to the filter selected in the chart configuration series options. For example, if you select Average, the y-axis scales to the maximum average value for the time units in the selected range. Refer to Customizing your charts for instructions on how to change the configuration.

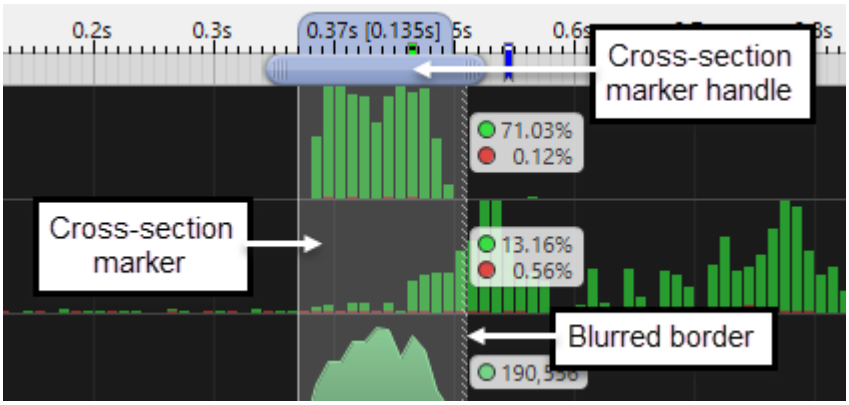
4.4 Analyze a region of time with the cross-section marker

To select a range of time to investigate more closely, click anywhere on the timeline and drag the handles on the cross-section marker. The information shown in the details panel when in **Processes** mode or **Samples** mode updates to show data for the window of time you have defined.

To move the view so the selected range is in the center, click **Center the display on the cross-section marker** on the toolbar of the **Live** view or the **Timeline** view.

If you select a time range, then change the scale of the view so the cross-section marker cannot sit precisely, the border appears as a blurred line.

Figure 4-6: Cross-section marker with blurred border.



Note

Unlike the filter controls, moving and expanding the cross-section marker does not affect the data in the other report views. To focus the data in the other views, use the calipers instead.

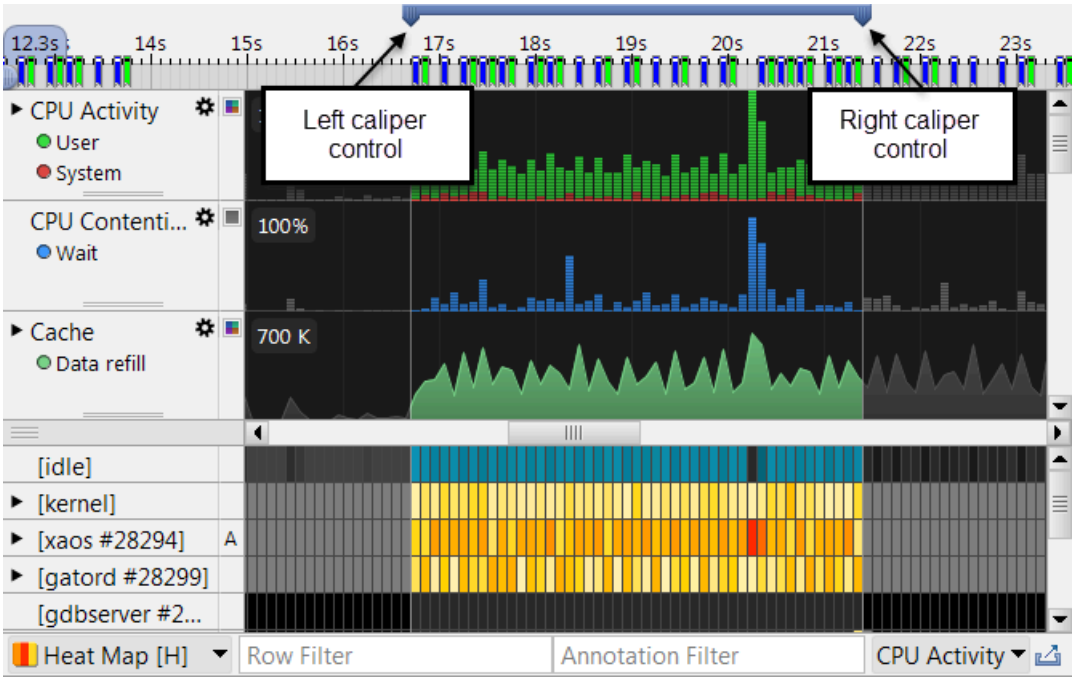
You can also control the cross-section marker with contextual menu options and keyboard shortcuts:

- Set the cross-section marker, or one end of it, to a specific time index with the corresponding contextual menu option.
- Move the cross-section marker with the left and right arrow keys on your keyboard.
- Adjust the range that the cross-section marker covers by holding down Shift while pressing the arrow keys.

4.5 Analyze a region of time with the calipers

Use the calipers on the timeline to select a region of time. The **Call Paths**, **Functions**, and **Code** views update to show data for the time period set with the calipers.

Figure 4-7: Caliper controls on the timeline.



Drag the calipers to set the required time region, or right-click on the timeline and select **Set Left Caliper** or **Set Right Caliper**.

To reset the calipers so they include all the data, click **Reset Calipers** on the toolbar of the **Live** or **Timeline** view.

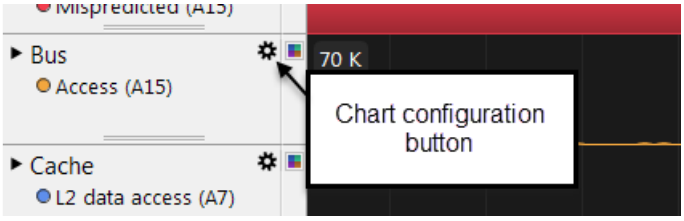
4.6 Customizing your charts

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

4.6.1 Chart configuration panel

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

Figure 4-8: Chart configuration button

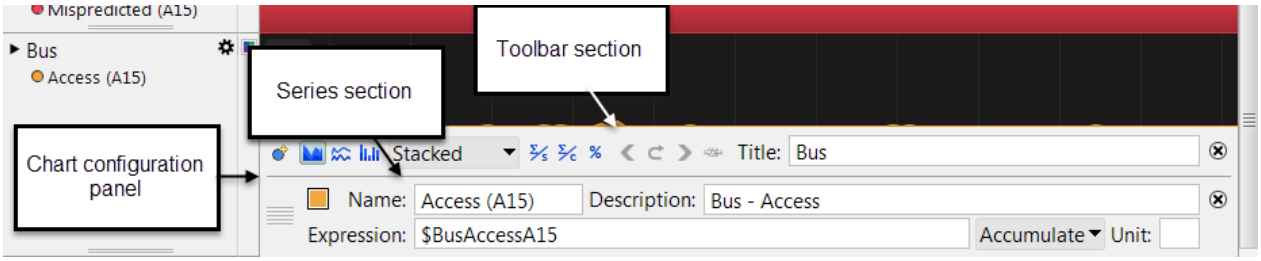


The chart configuration panel, in the following figure, contains the following sections:

- 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 defines the chart title as **Bus**, and the chart type as Stacked. The series section defines the **Access (A15)** series.

Figure 4-9: Chart configuration panel



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

4.6.2 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.

Create new wildcard series

Adds an empty wildcard series to the chart.

Chart Type

To choose one of the following chart types, use the Type buttons on the left side of the toolbar:

Filled

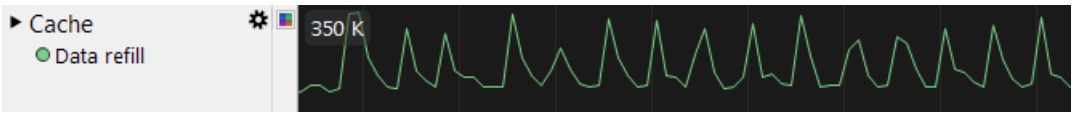
In a filled chart, each series is displayed as an area filled with the color that is specified in the chart configuration.

Figure 4-10: Filled chart

Line

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

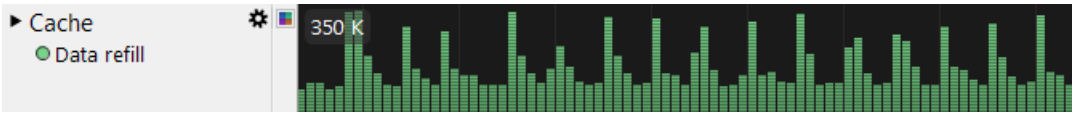
Figure 4-11: Line chart



Bar

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

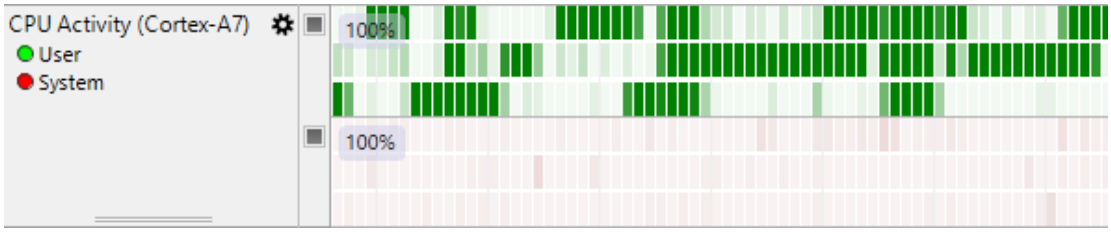
Figure 4-12: Bar chart



Heatmap

In a heatmap chart, each series is displayed as a heat map. Each counter is shown on a separate line and the bin color intensity increases in proportion to the number of samples.

Figure 4-13: Heatmap 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 the series that are 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.

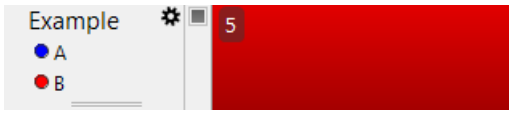
Figure 4-14: Stacked chart

Stacked charts are appropriate when the events are counted in exactly one of the series in a chart. For example, they are 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. Their position in the chart control determines the front-to-back ordering. To prevent series from obscuring data, as shown in the following figure, place series with larger values above series with lower values in the chart control. Drag and drop series controls using the chart configuration panel to reorder them.

Figure 4-15: 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. For example, Data Read Requests and Data Read Hits.

Logarithmic

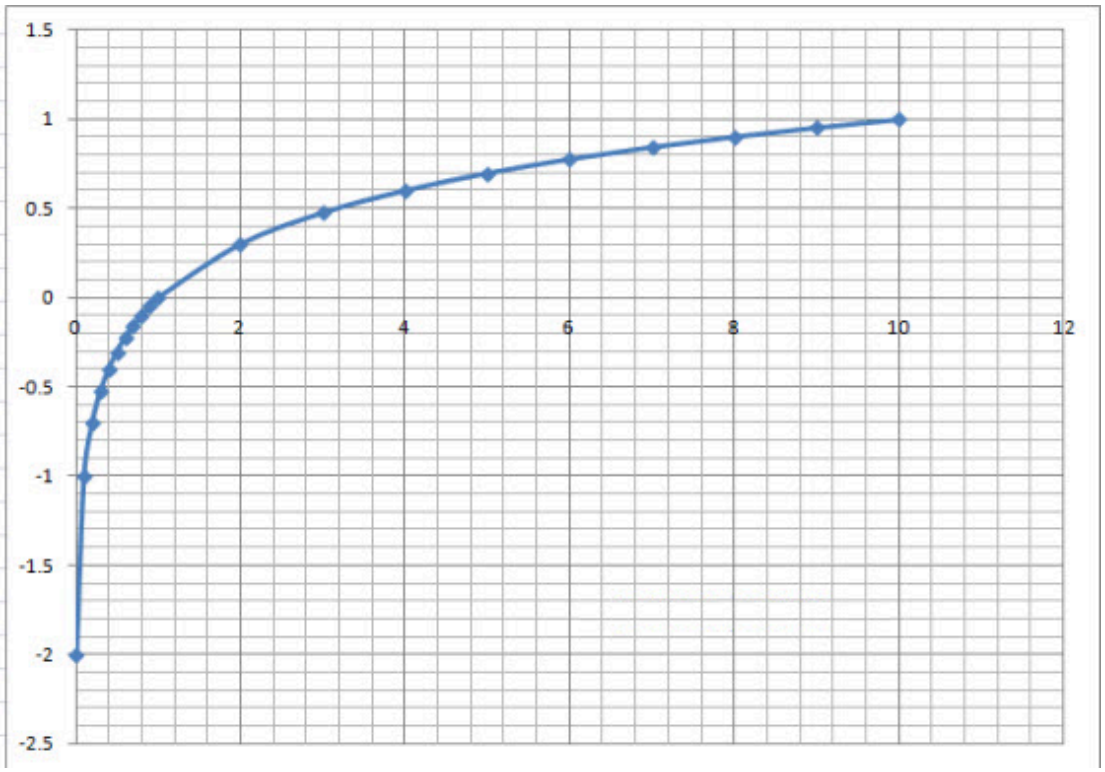
Shows all data points on a log10(y) scale where y is each Y-axis data point in the chart. This Y-axis modifier is useful when there is a huge difference between the minimum

and maximum values in a chart. A logarithmic chart displays with a series of horizontal lines, which increase in value by a power of ten.

Note

The charts cannot show negative numbers, so values less than one, which have a negative value in log space, appear as zero.

Figure 4-16: Logarithmic scale



Σs Average Selection

The cross-section marker overlay shows the average value of all bins that are included in the selection. If not selected, the overlay shows the total value of all bins in the selection.

Σc Average Cores

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

Plot values as a percentage of the largest value in the chart.

Title

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

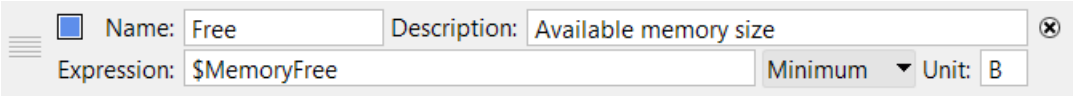
Remove Chart

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.

4.6.3 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 that enable you to customize the data that is used to plot the chart, and its look and feel.

Figure 4-17: 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 that 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 the color box in its series control to open the **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. Open a drop-down list of counters by pressing Ctrl+Space or the $ symbol. To see a description of a counter, hover over it in the list. To add a counter to the **Expression** field, select it in the list. 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 parentheses 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. If `x` is nonzero, the result is `y`. If `x` is zero, the result is `z`. Only one of `y` or `z` is evaluated.

`abs`

Returns the absolute value of the numeric expression that is 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 that is rounded to an integer.

Usage: `round(x)`, where `x` is a numeric expression.

You can also use the following built-in variables:

`$ZOOM`

This variable is the bin size of the current zoom level, in seconds, given as a floating-point value. For example, when showing the 200ms zoom level, this variable has the value 0.2.


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. To select the filter, use 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 that the time bin represents. It then converts the result 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 that was recorded within that second.

Unit

Enter the unit type for the series. The value that you enter in this field appears when you use the cross-section marker to select one or more bins.

Remove Series

To remove the current series from the chart, click **Remove Series**.

Related information

Chart configuration panel on page 72
Counter classes on page 124
Chart configuration toolbar options on page 72

4.6.4 Create a chart configuration template

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. A set of built-in templates is provided but you can also create your own.

Procedure

1. Configure the charts in the report as required. See Chart configuration toolbar options for information about the options available.
2. Click **Switch and manage templates**.
3. Select **Save as...** and enter a name for your template.

Results

The template is saved to the following location:

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

Related information

Toolbar options in the Live and Timeline views on page 95

4.6.5 Apply a chart configuration template

A chart configuration template is a set of pre-configured charts that you can apply to the report that is displayed in the **Live** view and the **Timeline** view. A set of built-in templates is provided to accelerate analysis of Arm® Mali™ GPU rendering operations.

About this task

You can apply templates to existing reports. You can also apply templates to new captures using the **Select Counters** dialog box. This feature ensures that the counters that the charts in the template require 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.

Procedure

1. Click **Switch and manage templates**  .

   Note

   To append the charts in the template to the currently displayed charts, rather than replacing them, hold down Shift when you click **Switch and manage templates**  .

2. Select the template from the drop-down list.
   To revert the report to its default chart configuration, select **<Default Template>**.

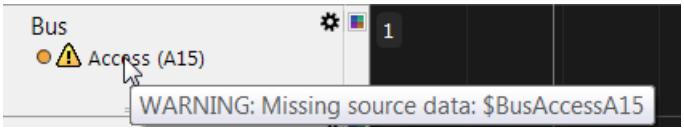
   To add a template to the list, select **Install and Load...**.

Results

Streamline displays the charts that the template specifies.

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 4-18: Warning message for a missing counter.



4.6.6 Match dynamic counter sources

When capturing a profile of your application, the application, or other sources such as the Performance Advisor layer driver, can generate software counters. Software counters sometimes require namespacing, for example reporting unique values per graphics context. To include counters using namespaced names in your templates, create a wildcard series to match a counter using a fixed prefix string and a variable post-fix.

Procedure

1. Create a chart configuration template, as described in Create a chart configuration template. Click the **Create new wildcard series** button in your chart, then enter part of a counter name with a wildcard * for the new series to match with intercepted counters.

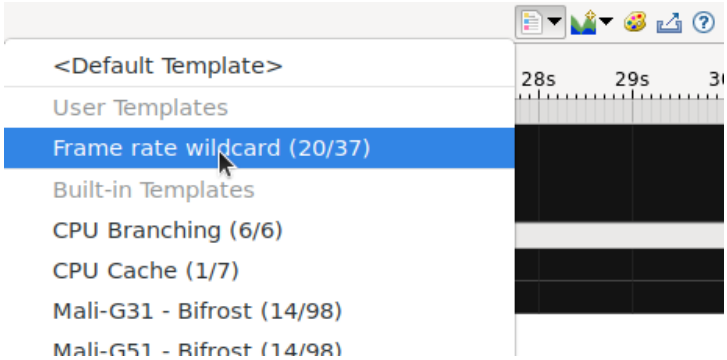
   Note

   The supported wildcard format is `$<Prefix>*`, where `<Prefix>` is the start of the counter name that you want to match. The wildcard creates a `match_with` attribute in your template. The `match_with` attribute specifies the variable name pattern that is used to match variables by the template. For example, it can be used to match the `EGL Context` suffix (a hex number) that is appended to variable names by the Performance Advisor layer driver. This example shows how a wildcard (*) is used to specify all counters that start with `DrawCallsFrameEGLContext`.

   ```
   <chart title="Draw calls" series_composition="stacked" rendering_type="filled"
    average_selection="yes">
   <series name="Draw calls EGLContext ${*}" match_with="$DrawCallsFrameEGLContext*"
    \
     description="The number of draw calls." display="AVERAGE" color="246,138,51"
   units="draw calls"/>
   </chart>
   ```

2. Apply your wildcard configuration template to your charts in **Live** view or **Timeline** view. Open the **Select Counters** dialog box, click **Switch and manage templates**, and select your template containing the wildcard name.

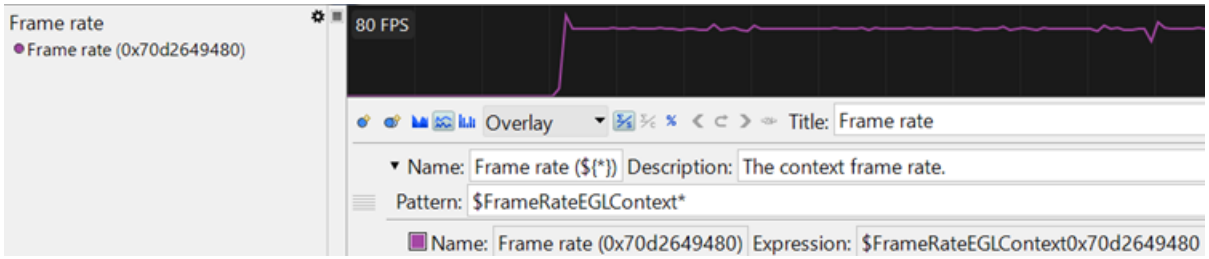
Figure 4-19: Select wildcard template.



3. Expand your chart and see the matched chart series with their expressions.

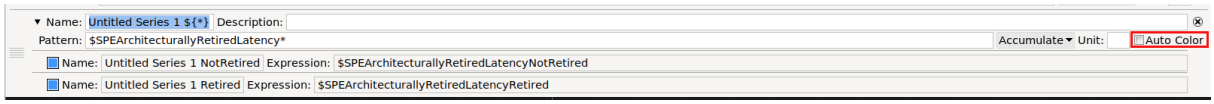
The `expression` for the chart series now shows the full matched variable name, instead of the wildcard pattern. You can change the color of the chart series to help distinguish between different series, but the color is not persistent. The `Name` and `Expression` cannot be edited.

Figure 4-20: Expanded wildcard chart series.



4. To change the color of the wildcard series, select the **Auto Color** checkbox, or click the color box to open the **Color** dialog box.

Figure 4-21: Wildcard chart series Auto Color checkbox.



When you save your template, the color of the wildcard series is saved.

4.7 Viewing application activity

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.

4.7.1 Details panel modes

The details panel has the following modes:

Heat Map mode

The **Heat Map** shows 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** shows a list of threads and processes that were active during the capture session and a color-coded activity map. The activity map shows which core was responsible for most 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. Jump to the relevant row in the **Functions** view by double-clicking on a function.

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 depends on the current selection of the cross-section marker.

Mali timeline mode

The **Mali Timeline** mode shows the device, process, graphics context, and a list of the queues that were present in your capture. You can analyze the events data shown in **Mali Timeline** mode at a variety of zoom levels and get the **Initiated** and **Duration** times for an event.

Use **Mali timeline** mode if your job manager GPU is version r43p0 or later. To use the GPU Timeline layer driver, you must have a Mali™ device driver version r51p0 or later.

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. This mode is only available on Arm® Mali™ devices with OpenCL timeline support compiled into the driver.

Use **OpenCL** mode if your job manager GPU is version r42 or earlier.

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. Filter the data in the details panel by entering a regular expression in the field. 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 information

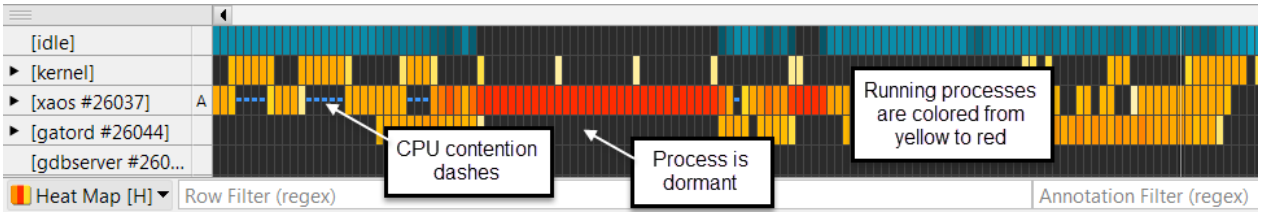
- Add annotations to your code
- Keyboard shortcuts in the Live and Timeline views

4.7.2 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 4-22: Heat Map mode



Here is what each of the colored bins 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. CPU contention can happen if there are too many processes and not enough cores to handle them.

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. To filter the list of processes and threads, add a regex to the row filter. To filter the string annotations displayed in the **Heat Map**, add a regex to the annotation filter.

Figure 4-23: Filtering annotations

To export the **Heat Map** data to a text file, click **Export Heat Map Data to a text file** to the right of the filter fields. 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.

Note

Annotations are not exported.

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

Related information

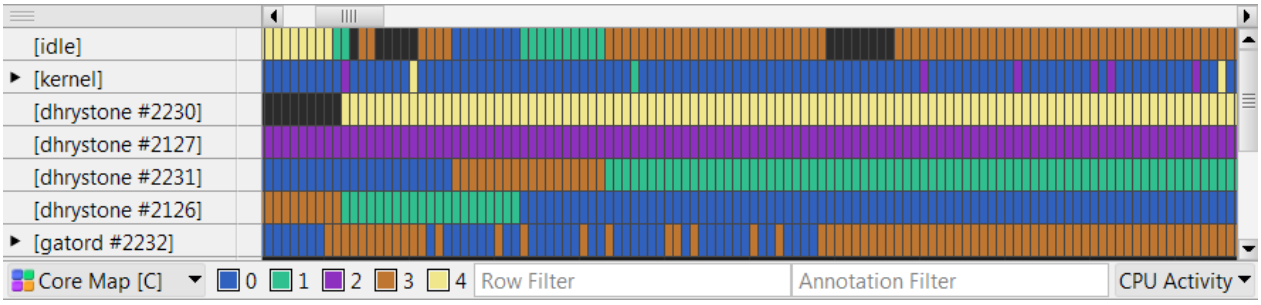
- Selecting the activity source

4.7.3 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.

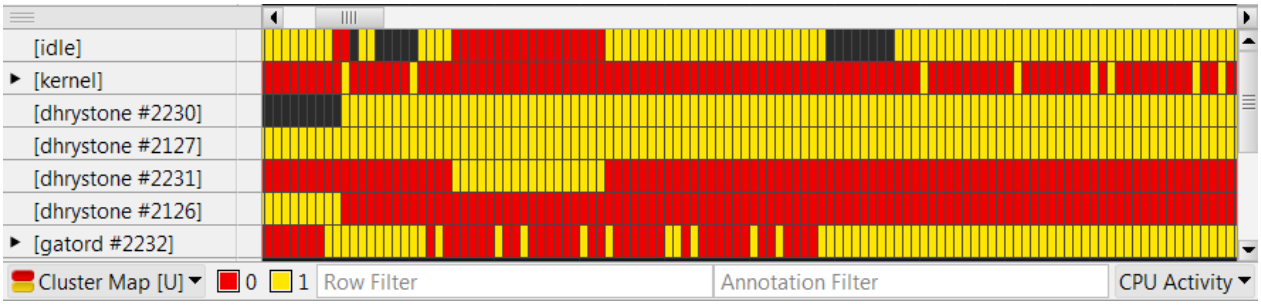
Core Map mode with five cores:

Figure 4-24: Core Map mode with five cores



Cluster Map mode with two clusters:

Figure 4-25: Cluster Map mode with two clusters



Note

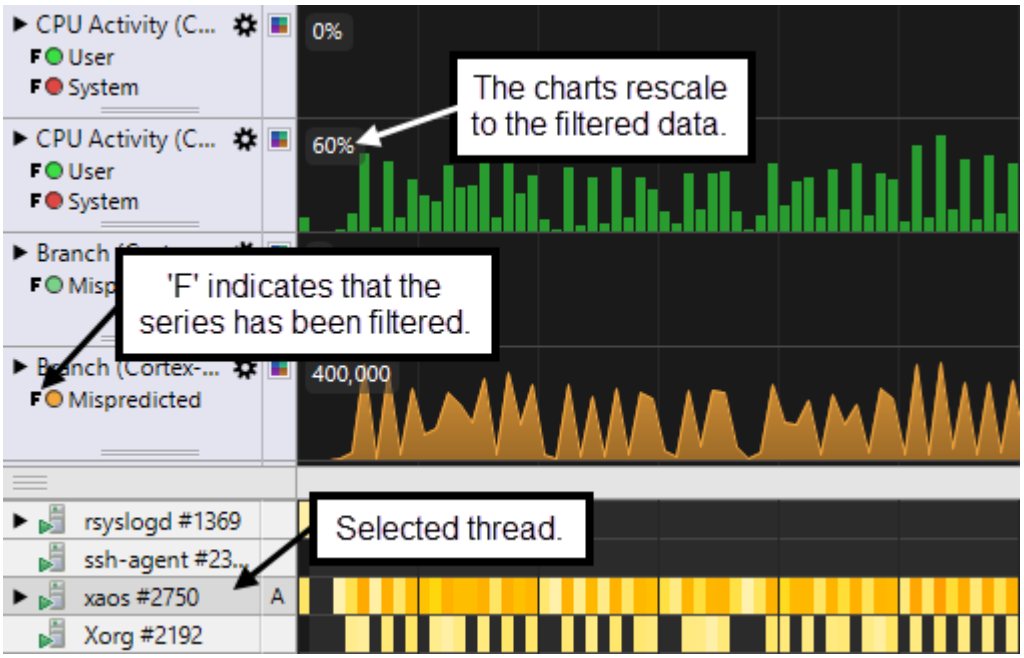
Captures using SMP systems support **Core Map** mode. Hardware targets with more than one cluster of cores support **Cluster Map** mode. These modes do not appear in the mode menu for captures that do not support them.

4.7.4 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 only show activity resulting from the selected processes or threads. The charts rescale to the limit of the filtered data. The letter **F** next to the series name indicates that the series is filtered.

Figure 4-26: 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 the selected processes or threads initiate. You can then differentiate between GPU activity that your application causes and activity resulting from other applications or system services.

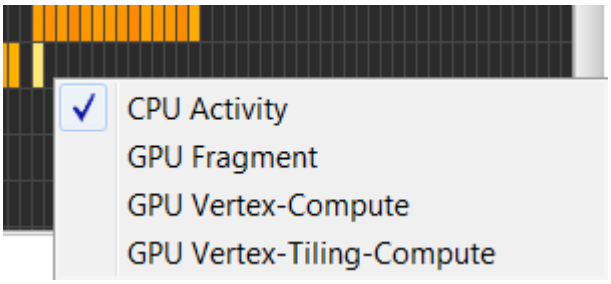
4.7.5 Selecting the activity source

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.

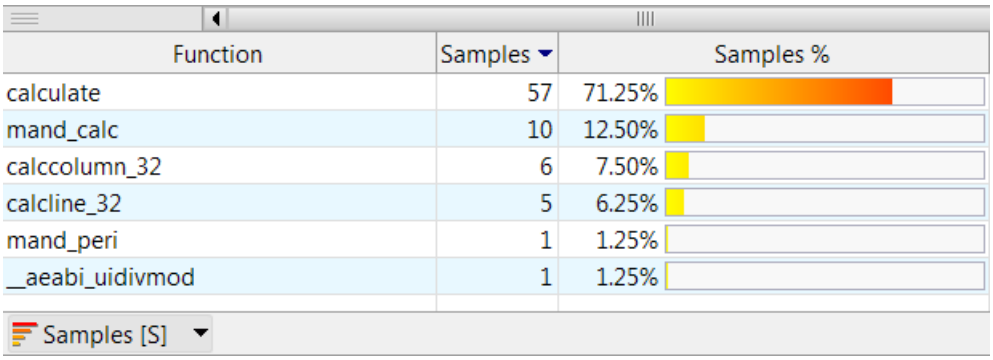
Figure 4-27: Processes focus menu



4.7.6 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 4-28: 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.

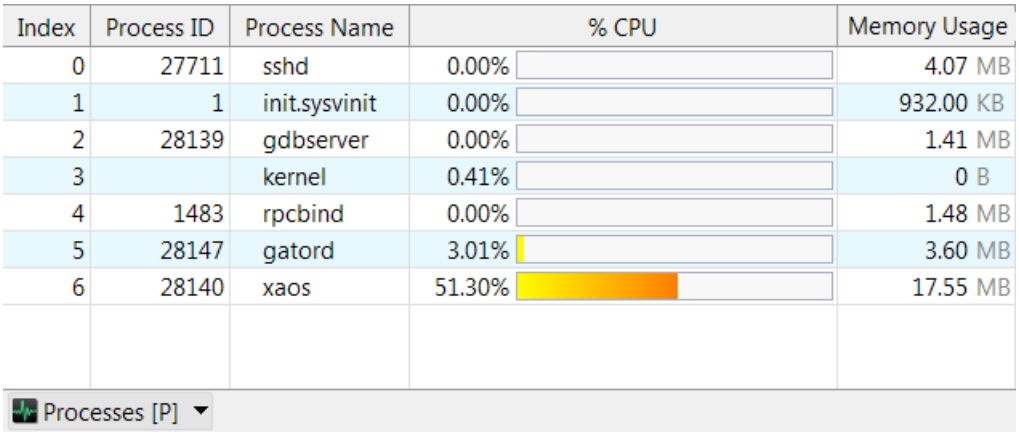
Edit Source
: Opens the source files for the selected functions in your default code editor.

Jump into the relevant row in the **Functions** view by double-clicking on a function.

4.7.7 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 4-29: 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 the cross-section marker covers.

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

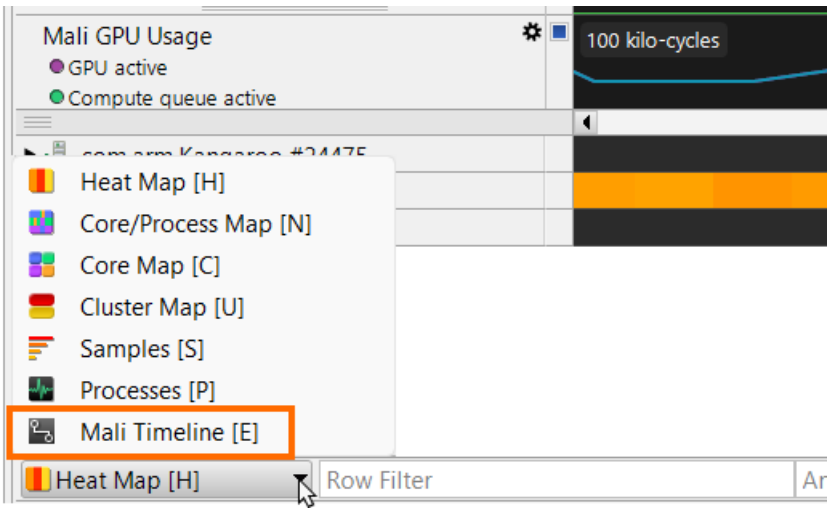
4.7.8 Mali Timeline mode

The **Mali Timeline** mode in the **Timeline** view presents GPU queue scheduling information from your capture, in the form of Mali™ Timeline event data. **Mali Timeline** mode enables you to understand more about performance and scheduling problems for the workloads in your captured frames.

Mali Timeline details panel

When you are viewing your captured frames in the **Timeline** view, you can select **Mali Timeline** mode from the details panel menu.

Figure 4-30: Select Mali Timeline mode



Note

If your device does not support Mali™ Timeline events, the **Mali Timeline** mode is not listed in the menu, and an error message displays in the **Error** view. The requirements for capturing Mali™ Timeline events are listed in Enable Mali Timeline Events using the Streamline GUI and Enable Mali Timeline Events for headless capture.

4.7.8.1 Mali Timeline data

In the **Mali Timeline** mode, the Mali™ Timeline event data shows the device, process, graphics context, and a list of the queues that were present in your capture:

Device ID

The numeric ID assigned to each device.

Process ID

The PID that Linux assigned to the process.

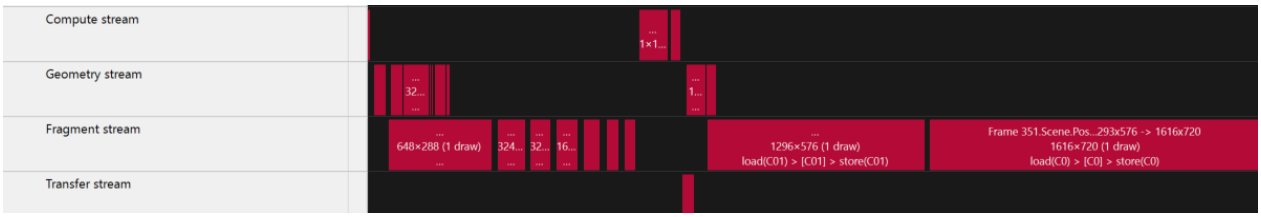
Contexts

A list of the graphics contexts.

Streams

Each stream is a workload queue that is present in your capture. The queues help you to identify scheduling problems, for example when the queues are processed serially for all or part of the frame.

Figure 4-31: Example Mali Timeline data using GPU Timeline



Note

- The data presented in the **Mali Timeline** mode is provided by the Perfetto service. The information received from the driver by Perfetto combines the “Execution”, “Sync”, and “Memory” stage tracks into a single track for each compute queue.
- To collect Mali™ Timeline Event data, your device and architecture must meet a number of prerequisites. If your device does not support Mali™ Timeline Events, you cannot select the **Mali Timeline** mode in the details panel menu, and an error message displays in the **Error** view. For more information about these prerequisites and how to capture Mali™ Timeline Event data, see the **Before you begin** section in: Enable Mali Timeline Events using the Streamline GUI.

To see the **Mali Timeline** events in the CAM track clearly, you must use a high level of zoom. If the events are not clear, re-analyze the capture with **High** or **Ultra-High** resolution mode so that you can access higher zoom levels and see more details. To learn how to re-analyze your capture with higher resolution modes, see Re-analyze stored capture data.

4.7.8.2 Mali Timeline events

To find workloads in your captured frames that are causing scheduling problems, zoom in and navigate the workloads using the labels on the CAM events. When you zoom in, the labels are progressively revealed as the CAM box size increases to show the label details.


Note

You can control what is shown in the debug label on the CAM boxes from the API label stack, and specify how contracted labels display when they are longer than the CAM box size. To change the preferences, open **Window > Preferences > Mali Debug Labels**. Preferences are applied in the same order as they appear in the dialog box.

To see information about an event, hover your mouse pointer over an event on the CAM track. A tooltip shows metadata relating to that event, any debug label information obtained from the layer driver, as well as the **Initiated** and **Duration** times for the event.

Click on an event that you want to investigate further. Detailed properties about the selected event are displayed in a panel. To see summarized properties for multiple events, press Ctrl then click on the CAM boxes that you want to include in the selection. Alternatively, right-click and use the selection options from the context menu:

- Select the complete workload, frame, or command buffer.
- Select all events in a time range if you are analyzing events inside a region of time using the cross section marker.
- Select all events matching the **Activity Filter**.


Note

To clear all selected events, right-click and choose **Clear selection** from the context menu.

To export relevant CAM data for further post-processing, use the `-report` command line option:

```
streamline-cli -report -cam <GPUTimelineCapture.apc>
```

Related information

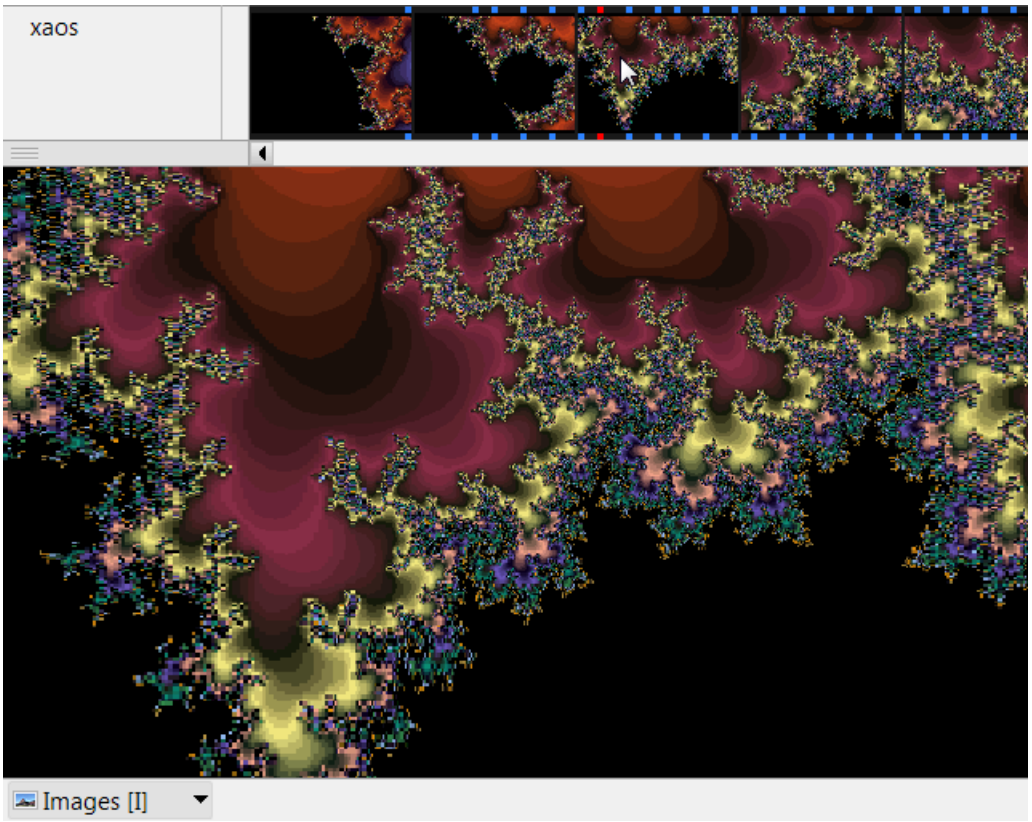
Enable Mali Timeline Events using the Streamline GUI on page 37
Enable Mali Timeline Events for headless capture on page 40

4.7.9 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 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 4-32: Images mode



4.7.10 OpenCL mode

The Open Computing Language, or OpenCL, is a framework for parallel execution of jobs or kernels using task-based and data-based parallelism. **OpenCL** mode provides a visual representation of OpenCL code running on Arm® Mali™ devices.

Note

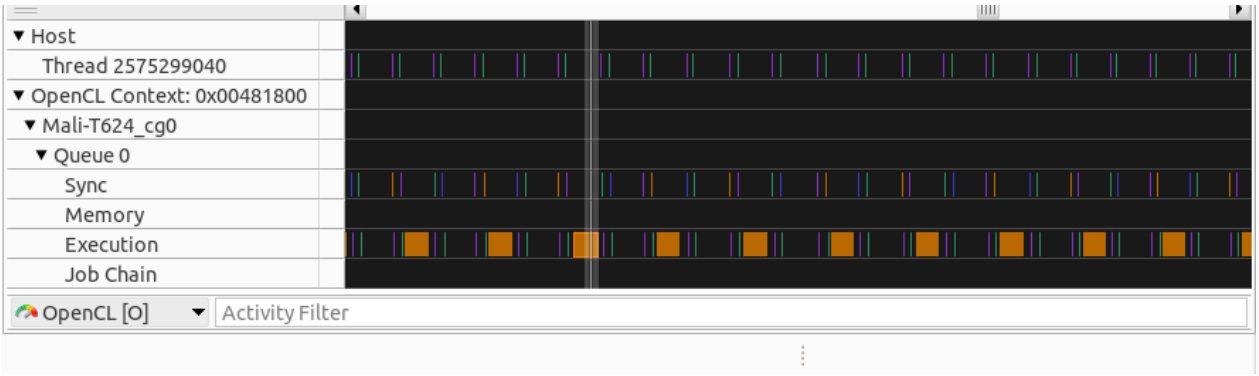
- **OpenCL** mode is only available on platforms with an OpenCL timeline compiled into the Mali™ driver. Contact your platform provider or Arm Technical Support for more details.
- The information in this topic applies to job manager GPUs up to version r42. CSF-based GPUs from DDK version r43p0 use Mali™ Timeline events to capture

OpenCL data. Jobs are placed in one Custom Activity Map (CAM) track for the compute queue instead of categorizing them as shown in **OpenCL** mode. For more information, see Enable Mali Timeline Events using the Streamline GUI.

This mode shows which command is being run on each thread over the course of the capture session.

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 Mali™ Driver Development Kit (DDK).

Figure 4-33: OpenCL mode



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

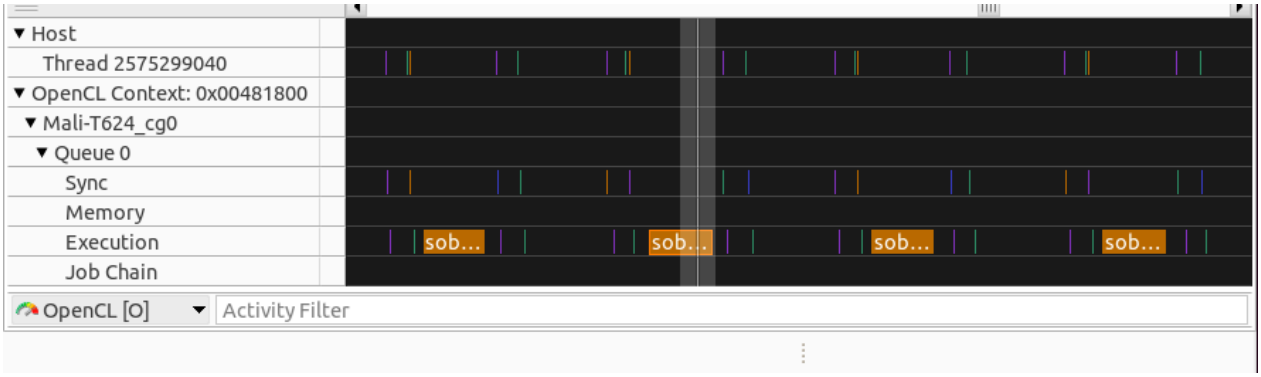
Command names are shown inside colored areas, representing the duration of their execution. If there are two or more commands in a bin, and there is enough room, the number of commands in the bin appears in light gray.

Clicking a command gives it focus and hides any non-relevant information. The following figure shows a selected command that is highlighted with a yellow border.

Figure 4-34: Selecting a command in OpenCL mode

Zoom out to see more commands and the points at which they were enqueued.

Figure 4-35: Zooming out in OpenCL mode



Hovering over a command displays a tooltip that shows the command name, the time that it was initiated, and its duration.

Figure 4-36: OpenCL mode tooltip

There is a filter field at the bottom of the chart in **OpenCL** mode. Enter a regex in the field and **OpenCL** mode updates, showing only matching commands.

Figure 4-37: Filtering in OpenCL mode

4.8 Toolbar options in the Live and Timeline views

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

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

Toggle bookmark markers

 Activates or deactivates the bookmarks. If selected, vertical lines appear 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

 Sets the calipers so they include all the data in the capture. The calipers are the blue arrow controls at the top of the view. You can use them to limit the statistics in the reports to an area of interest.

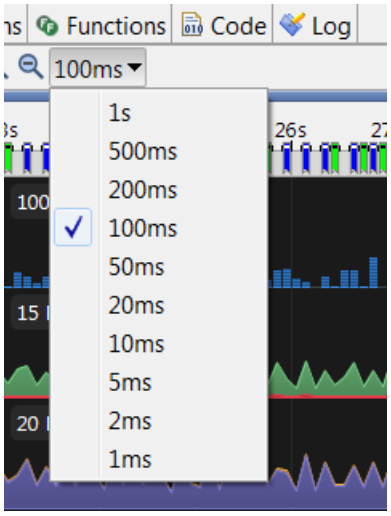
Select Annotation log... (Timeline view only)

 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. You can also use the combination of Ctrl and the scroll wheel to zoom in and out on the area around the mouse cursor.

Figure 4-38: Zoom level drop-down list



Time index

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)

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

Switch and manage templates

Enables you to do the following:

- Select a chart configuration template to apply to the current display.
- Manage saved templates.
- Create and load templates.

See Create a chart configuration template for more information.

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 has been collected during the capture session.

Cycle through themes

 Switches between the dark and light color schemes.

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.

Another export button below the details panel allows you to **Export Heat Map Data**. See Heat Map mode for further information.

Related information

Annotate your code on page 49
Create a chart configuration template on page 78
Contextual menu options in the Live and Timeline views on page 97
Keyboard shortcuts in the Live and Timeline views on page 127

4.9 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. This menu enables you to add bookmarks and to control the calipers and the cross-section marker.

The contextual menu options are:

Create Bookmark at...

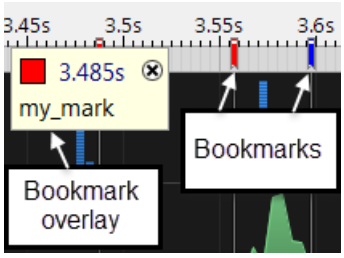
Creates a bookmark at the time index of the current position of the mouse cursor. The bookmark is displayed in the **Live** or **Timeline** view rule as a colored mark.

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 4-39: Bookmarks



To turn on or off vertical lines across the charts from the bookmarks, click **Toggle bookmark markers** on the toolbar of the **Live** or **Timeline** view.

Set Left Caliper

Sets the left caliper control to the current position of the mouse cursor. 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. Filters out all data after the right caliper from all views.

Reset Calipers

Resets the calipers to their default locations, selecting all data in the capture.

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 information

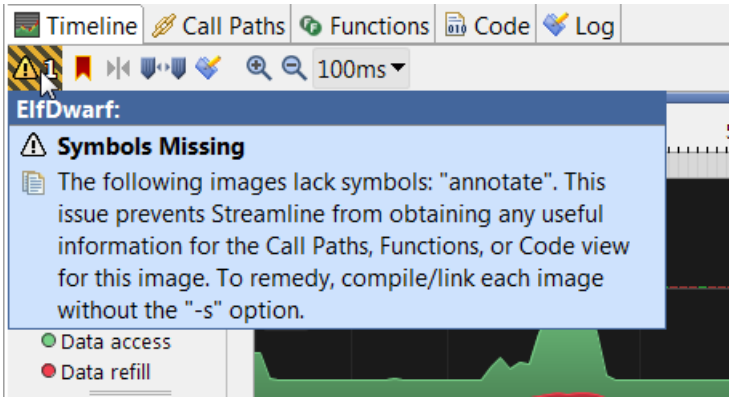
Viewing application activity on page 81
Toolbar options in the Live and Timeline views on page 95
Keyboard shortcuts in the Live and Timeline views on page 127

4.10 Warnings tag

If there is a problem with the reports, warnings appear in the **Live** and **Timeline** views. Clicking 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. If there are no issues, the tag is not displayed.

Figure 4-40: 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, an issue parsing ELF or DWARF information.

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.

4.11 Analyze your code

After you have analyzed your capture and determined that the CPU is causing performance issues in your application, you can analyze your code. Identify function-level hotspots in the **Call Paths** view and **Functions** view, then open the **Code** view to debug the source code and disassembly.

4.11.1 Analyzing Call Paths

The **Call Paths** view shows sample, size, and location data for each process, thread, and function called in your code. If you have used the caliper controls to filter data in the **Timeline** view, the data in the **Call Paths** view is limited to this selection.

Data is presented in a hierarchy of processes, threads, and functions below the one that called them.


Note

If call stack unwinding is not available, for example because it is not selected in the dialog box, the sampled functions appear as a flat list for each thread.

To see more information about a called process, thread, or function, expand the hierarchy to show its subordinates. To show only the immediate subordinates, press Shift+Left arrow to hide all the subordinate call chain links, then expand again.

In the data table, the name of each process is enclosed in square brackets. The names of threads are enclosed in curly braces.

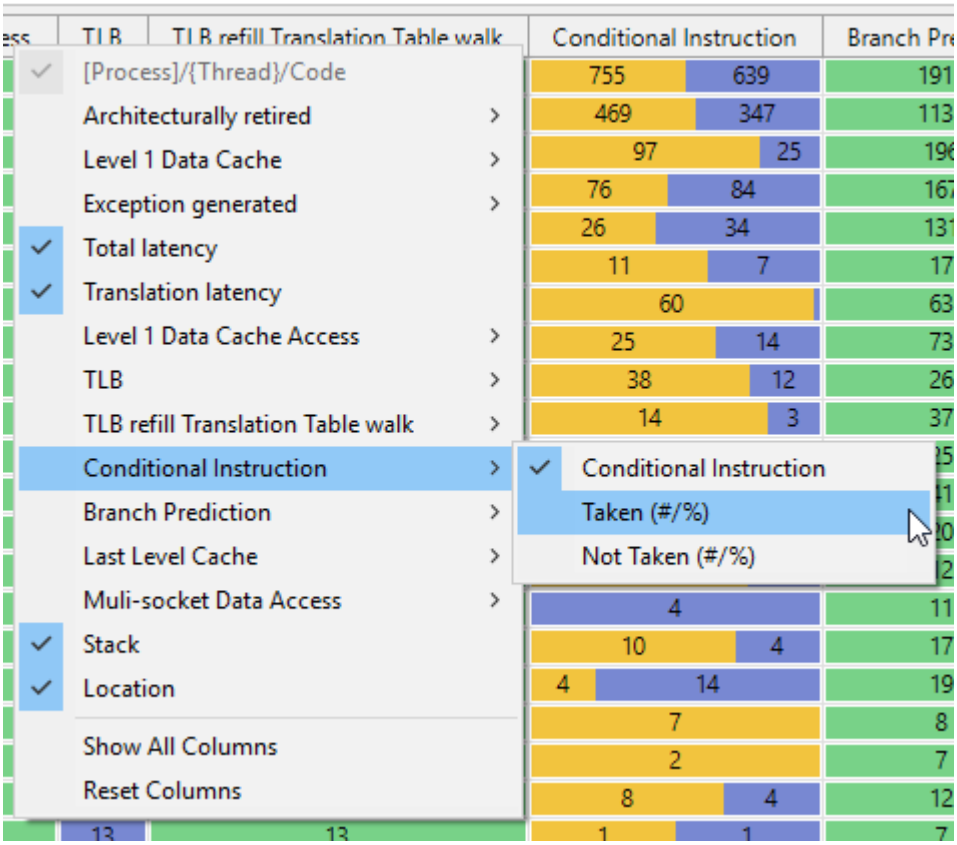
If the stack usage cannot be determined for one or more functions in the call path, the **Stack** value is a dark red color.

Right-click on any row in the table report to open a context menu where you can:

- Open the selected process, thread, or function in different views.
- Collapse rows that are not part of the current selection.
- Expand the hierarchy to expose hidden instances of the selected function.

When the source is set to **SPE**, the **Call Paths** view also has columns for the events that were captured. To add or remove columns, right-click on a column header, then select the column name. For columns that display a ratio, you can use this menu to show part of the ratio in a separate column.

Figure 4-41: Select the column headers from the menu.



Related information

- Set capture options
- Configure SPE counters
- Troubleshooting report issues
- Keyboard shortcuts in the table views

4.11.2 Analyzing Functions

The **Functions** view lists all functions that were called during the capture session alongside sample, instance, and usage data. If you used the caliper controls to filter data in the **Timeline** view, the data in the **Functions** view reflects this selection.

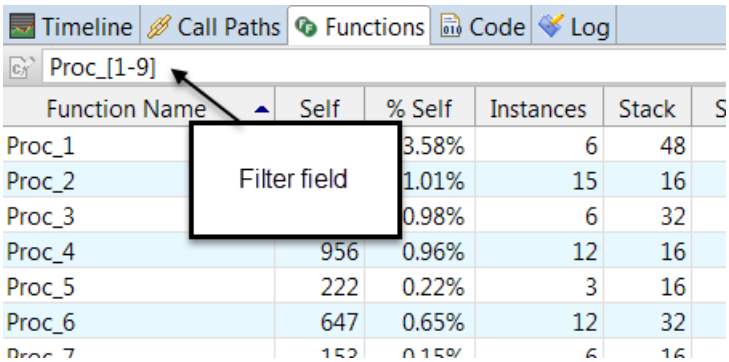
Note

The data shown in the **Functions** view depends on the text that is entered in the filter field, and the filtering selection in the **Timeline** view.

In the data table, the **Stack** value is a dark red color if the stack usage cannot be determined.

Filter the rows that are displayed in the view by entering a regular expression in this field. Only processes, threads, and functions whose name matches the regular expression are displayed.

Figure 4-42: Filter field in the Functions tab.



When the source is set to Periodic Sampling, or EBS, the **Functions** view also shows the number of samples collected by all instances of the function, and the percentage of the total samples collected by all functions.

Note

Samples are excluded if any functions that were called by the function collected them.

Right-click on any row in the table report to open a context menu where you can:

- Open the selected function in different views.
- View the top ten processes and threads that the selected function was called from.

When the source is set to **SPE**, the **Functions** view also has columns for the events that were captured. To add or remove columns, right-click on a column header, then select the column name as shown in Analyzing Call Paths.

Related information

- Configure SPE counters
- Keyboard shortcuts in the table views

4.11.3 Sorting table reports

To reorder data in the table views and bring functions to the top of the table, perform a multi-level search.

Procedure

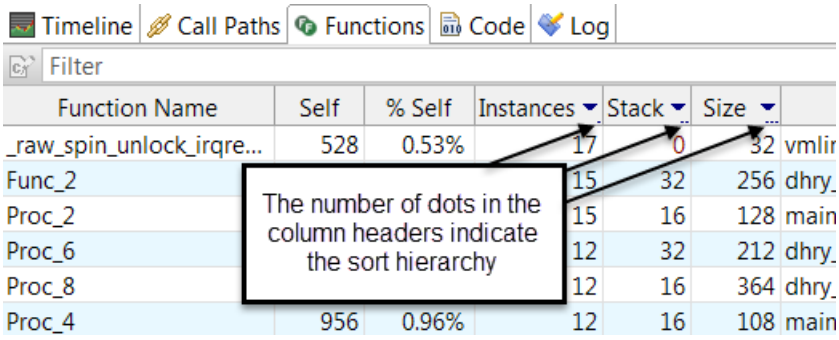
1. Select a column header to reorder data in that column. Select the column header again to reverse the sort order.

2. Hold down the shift key and select the next column header that you want to sort data by.
3. Repeat step 2 as necessary to sort your data by multiple levels.

Results

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

Figure 4-43: Multi-level sort



4.11.4 Viewing your code

The **Code** view helps you to identify function-level hot spots, at source and disassembly levels. If you have used the caliper controls to filter data in the **Timeline** view, the sampling data in the **Code** view reflects this selection.

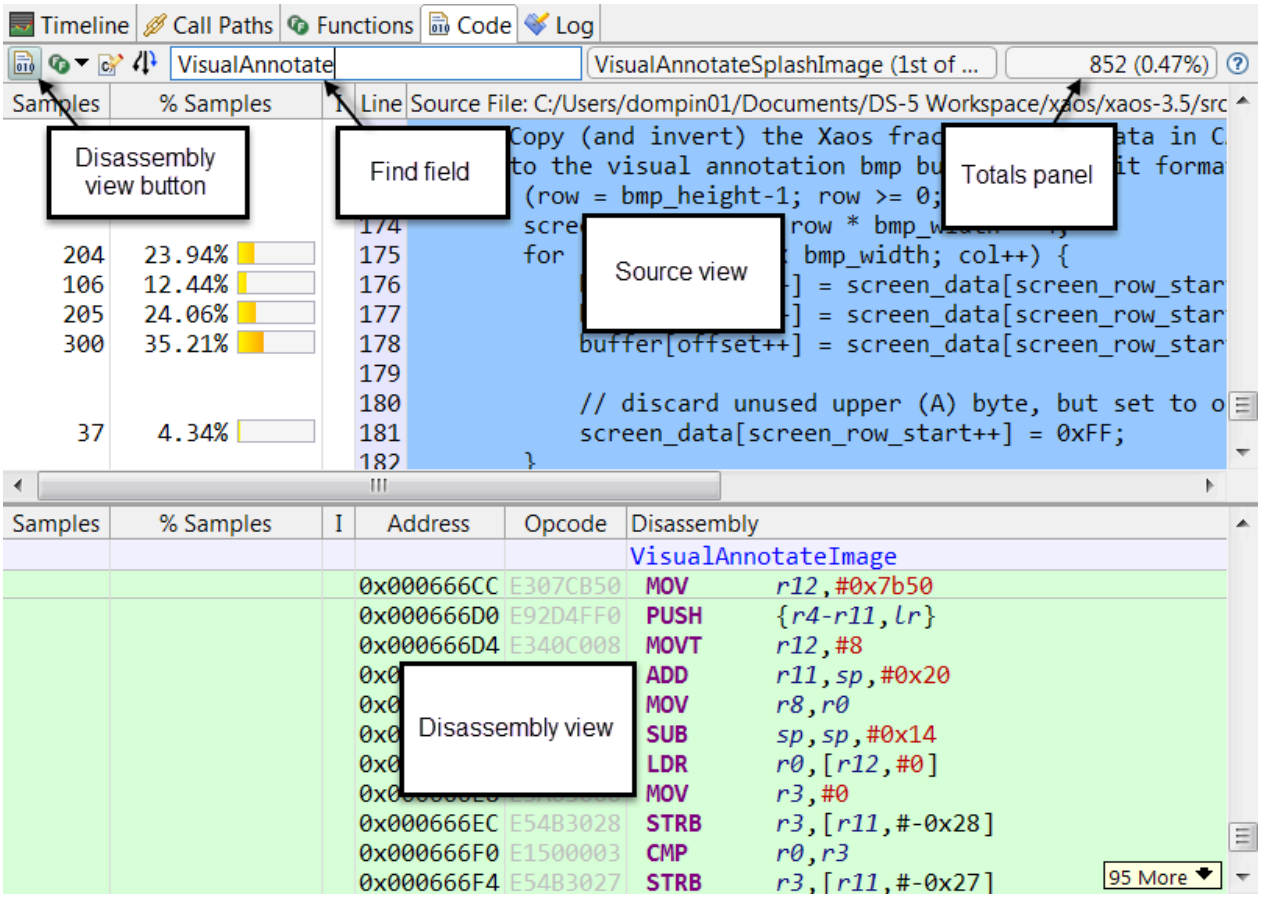
Before you begin

- Source code is only available for images in which the selected function is located, and if the image contains debug symbols. See Generate debug information for more information.
- In **Call Paths** view or **Functions** view, right-click on a function and choose **Select in code** from the context menu.
- If the source code is located in a different directory than when your application was compiled, you must redirect the file path so that Streamline can find the code. See Troubleshooting missing source files for more information.

About this task

When you open the **Code** view, your source code is shown with some color-coded statistics to help you identify functions that are causing performance issues.

Figure 4-44: Code view showing source code.



Procedure

1. Select your source code to highlight related instructions in the disassembly panel.
   - Click the first row, then drag your mouse across a range of rows.
   - To select a single row of code, simply click it.
   - To select multiple rows of code, click the first row and press Ctrl to select more individual rows. Alternatively, click the first row, then press Shift and click the last row of the series.

   To view the disassembly, click **Toggle** to open the disassembly panel.

   The instructions that are related to the code you selected in the **Code** view are shown in the disassembly panel.

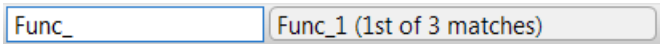
Note

- Selecting a single line of disassembly selects all the instructions that relate to that single line of source code. You can use function tags to highlight all instruction lines that are associated with a single function.

- Sometimes, the selected lines of source code or disassembly contain too many rows to fit in the current window. To see the hidden rows of code, select the **More** indicators.

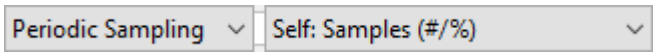
2. Find and select a specific function to analyze it further:
   - Click **Recently selected functions**, and choose a function from a list.
   - Enter a string in the **Find** field located just below the toolbar, and press Enter to cycle through the matching functions.

     Figure 4-45: Find field below the toolbar.

3. If your capture contains PC sample sources, select a source from the first list, and then choose a counter to focus your analysis on that area.

   Figure 4-46: Selecting a counter for analysis.

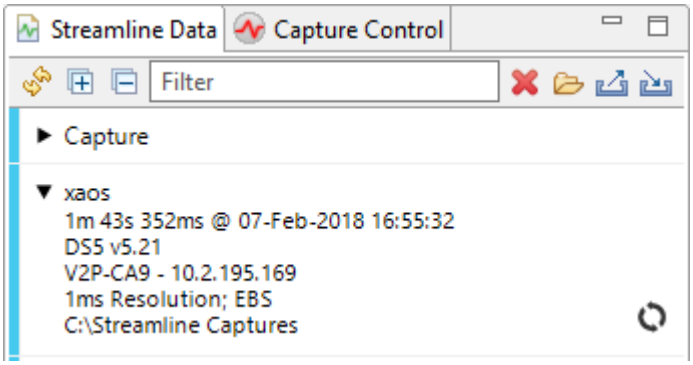
   

4. To open the source file in your preferred editor and update your source code, click **Edit Source**.
   After you have edited your source code, capture the profile again to compare the performance of your application. See Starting a capture for help with capturing a profile.

4.12 Streamline Data view

Use the **Streamline Data** view to manage your capture files and select the captures that you want to see or analyze.

Figure 4-47: Streamline Data view.



Note

You can only open a report that was generated with a version of Streamline that is no greater than two years old. See the Getting started with Streamline chapter for

compatible versions for your project. You must first re-analyze the report using the instructions in Re-analyze stored capture data.

4.12.1 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. To open the report for a blue capture, double-click on it.

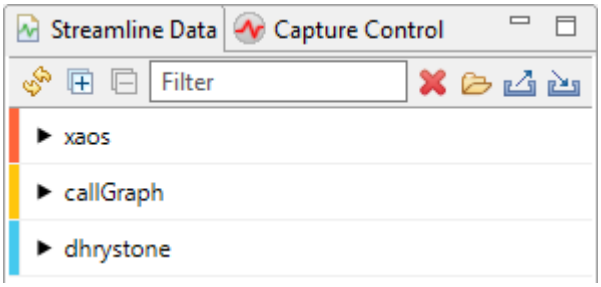
Amber

Amber indicates a capture that needs to be re-analyzed, as its report is missing or incompatible with the current version of Streamline. To re-analyze the capture, click **Analyze** or double-click on the capture.

Red

Red indicates that the capture does not contain valid data and cannot be re-analyzed. It might indicate a capture error, in which case the error message is displayed in a tooltip.

Figure 4-48: Color-coded captures



4.12.2 Re-analyze stored capture data

To add additional debug information, see a higher level of detail, or use a capture that was generated by an incompatible version of Streamline, you must re-analyze your capture. Re-analyzing a capture replaces your existing report.

About this task

There are several reasons why you might want to re-analyze a capture, for example:

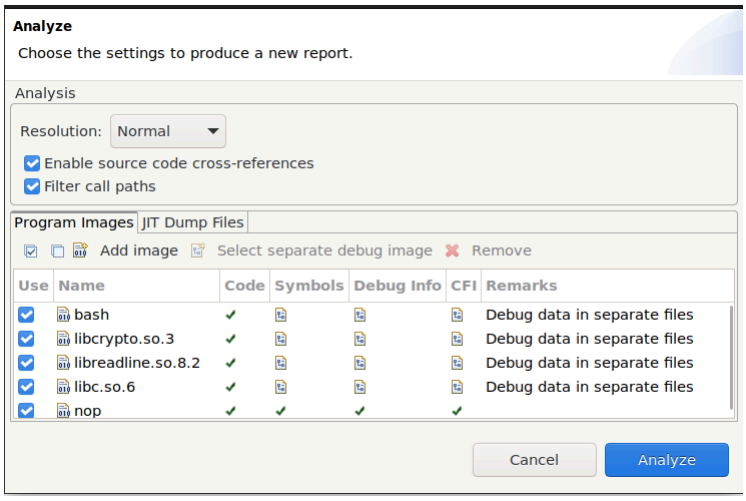
- To change your analysis options to include debug information
- To see events more clearly at a higher resolution.
- To use sources that were not available when the data was captured.

- To use a new version of Streamline that is incompatible with the existing report.

Procedure

1. In the **Streamline Data** view, right-click on the capture then select **Analyze**. The **Analyze** dialog box opens.
2. Change the required settings.

   Figure 4-49: Analyze dialog box.




Note

The options for re-analysis are a subset of the options available in the **Capture Settings** dialog box and they work the same way.

3. Click **Analyze**.

Results

Streamline re-analyzes the capture data with the new settings and generates a new report, which replaces the existing report.

4.12.3 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.

Procedure

1. Right-click on the capture and select **Duplicate**.
2. In the **New Name** dialog box, give the **Duplicate Streamline Document** a new name.
3. Click **OK**.

Results

A copy of the capture with the new name appears in the **Streamline Data** view.

5. Use Streamline from the command line

Use Streamline outside of the user interface to perform automated captures and generate text-based reports that you can import into a spreadsheet application.

The `streamline-cli` command has different modes that enable you to use most features outside of the graphical user interface.

Use the `streamline-cli` command in Streamline-enabled shells with the following syntax:

```
streamline-cli <mode> [options] <files>
```

This section describes the available modes.


Note

Some options are common to all modes:

`-h, -?, -help`
Output help information to the console, listing each mode and option, and the required syntax.

`-v, -version`
Display the program version information.

5.1 Open a Streamline-enabled command prompt or shell

Open a command prompt or shell from which you can use the `streamline-cli` command to access most of the functionality of Streamline.

Procedure

1. Add the appropriate location to your `PATH` environment variable:
   - For Arm® Performance Studio, add `<install_directory>/streamline`.
   - For Arm® Development Studio, add `<install_directory>/bin`.
2. Open a command prompt or shell.

Related information

List modes on page 109
Generate-config mode on page 110
Capture mode on page 111
Analyze mode on page 113
Report mode on page 113
Import modes on page 117

5.2 List modes

List the connected devices, or the packages, on a connected device.

Syntax

```
streamline-cli <mode> [options]
```

Modes

`-list-devices`
List the connected devices.

`-list-packages`
List the packages that are available on the connected device.

Options

Options for `-list-devices` :

`-w, -wait <seconds>`
Specify the maximum time that Streamline waits for responses from any devices.

Options for `-list-packages`:

`-a, -all`
List all packages that are installed on the device, including packages that are not debuggable.

`-t, -activity`
List all the main activities of each package.


Note

To profile non-debuggable packages in the list, you must be running a development device with root access.

`-d, -device <device>`
If multiple devices are connected, specify the ID of the required device. To obtain the ID, use the `adb devices` command.

5.3 Generate-config mode

Generate a `configuration.xml` file with the available events from a template.

Streamline searches for the template name or path in the built-in templates, then in the `Templates` directory, then as a file path (`*.st`). The target address is the IP address (optionally suffixed with " `:<port>`") or" `adb:<serial>`" of a running instance of `gatord`.

Syntax

```
streamline-cli -generate-config [options] -template <name_or_path> <target_address>
```

Options

`-template <name_or_path>`

Additional template to apply. Streamline searches each template in the following order:

1. In the built-in templates.
2. In the `Templates` directory.
3. As a file path (`*.st`).

You can apply more than one `-template` option. To create a chart configuration template, see Create a chart configuration template.

`-include-default`

Include a default set of events from the target, in addition to the events referenced in the `-template` arguments.

`-o, -output <filename>`

The file to write the `configuration.xml` to. The default is to send the `configuration.xml` to the target.

Returns

A `configuration.xml` file.

5.4 Capture mode

Initiate a capture session.


Note

For Android, Arm recommends using the `streamline_me.py` script instead of this command. Refer to Generate a headless capture in the Arm Streamline Target Setup Guide for Android for more details.

You must specify a valid `session.xml` file for this mode. This file defines your target hardware and the parameters of the capture session.

To create a `session.xml` file, enter your settings in the **Capture Settings** dialog box then click **Export...**.

Syntax

```
streamline-cli -capture [options] <session.xml>
```

Options

`-c, -config <configuration.xml>`

The path to the configuration file to send to the target.

`-clean-gatord`

Stop and restart gatord if gatord is already running on target.

`-duration <seconds>`

The number of seconds that you want the capture to last. The capture automatically terminates after that duration.

`-e, -events <events.xml>`

The path to your `events.xml` file.

`-include-libs`

Include the relevant libraries when retrieving images from the target.

`-o, -output <filename>`

The name of the APC capture file to create.

`-package <package_name> [-activity <activity name>] [-activity-args <arguments>]`

Use with Android targets to specify the package to capture. Use `-activity` to specify the main activity to be launched. If `-activity` is not specified, the main activity from the package is used. If there is no activity or multiple main activities in the package, the capture will not be started. Use `-activity-args` to launch the activity with the specified arguments passed to the activity manager (am). The arguments must be provided as a single string.

`-password <string>`

The password to use when retrieving images from the target.

`-p, -pmus <pmus.xml>`

The path to your `pmus.xml` file.

`-retrieve-image <regex>`

A regular expression to match the processes to retrieve from the target.

`-template <name_or_path>`

The template to apply. Streamline searches each template in the following order:

1. In the built-in templates.
2. In the `Templates` directory.
3. As a file path (`*.st`).

You can apply more than one `-template` option. To create a chart configuration template, see Create a chart configuration template.

`-username <string>`

The username to use when retrieving images from the target.

Returns

An APC capture directory.

Example 5-1: 60 second capture session

Initiate a capture session that lasts for 60 seconds with the following command:

```
streamline-cli -capture -duration 60 session.xml
```

5.5 Analyze mode

Analyze a capture.

Syntax

```
streamline-cli -analyze [options] <capture.apc>
```

Options

`-resolution <mode>`

The resolution mode to use. Set `<mode>` to `summary`, `normal`, `high`, or `ultra`.

5.6 Report mode

Output data from a capture to the command line or to a file. You can use options to filter and format the data.

Syntax

```
streamline-cli -report [options] <capture.apc>
```

Options

`-all`

Output the contents of the **Timeline**, **Call Paths**, **Functions**, and **Log** views, including bookmarks and custom activity map tables. This option is the default.

`-bookmarks`

Output all bookmarks that are stored in a capture, listing the time index location of the bookmark and its text.

`-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.

`-callpath`

Output the contents of the **Call Paths** view. Subordinate functions are indented.

-cam

Output the contents of custom activity map tables.


Note

To export custom activity map data, the report must contain custom activity map annotations.

-disasm[=<name>] <images> [<source>]

Output the disassembly view part of the **Code** view as a CSV file.

`<images>` is a comma-separated list of ELF images (as were attached to the capture for analysis) to include in the output.

`<source>`, if specified, is the name for the data to output that is given in the **Code** view. For example, 'Periodic Sampling' or 'SPE'.

-format <space|tab|csv>

The format of the output. Format the tables using spaces, tabs, or comma-separated values. This option is useful if you want to easily convert output text files to a spreadsheet application. The default format is spaces, making the tables easy to read when printed on the command line.

-function

Output the contents of the **Functions** view.

-heatmap "<source>"

Export the contents of the **Heat Map** to a text file. `<source>` must match the name of the activity source for the **Heat Map** in the capture. Default = `"CPU Activity"`.

-individual-threads

Use this option with `-timeline`, `-callpath`, or `-all`, to produce report files specific to each thread. The thread ID is included in each generated filename. You must specify the process that the threads belong to using `-process`.

-log

Output the contents of the **Log** view.

-o, -output <output_directory>

Send report data to files in the specified directory. Files are saved with the same name as the capture, unless you specify a filename. If you specify multiple tables for export, each table is written to `<output_directory>` in a new file with an appropriate name. If you specify multiple captures, each capture is output to a subdirectory of `<output_directory>` with a name matching the capture.

-opencl

If there is an OpenCL table, output the contents.

-per_core

Output per-core data when used with the option.

-process <regex>#<PID>

A string specifying the process to inspect. The first part of the string is a regular expression that matches the name of the process. The second part is a '#' character followed by the numeric ID of the process. Both parts can be used together or individually.

Examples of valid arguments are:

- `-process=.sampleApp`
- `-process=.sampleApp#1234`
- `-process=#1234`

-scale <number>

Set the number of bins per second to use in the report. Default = 1,000.

-start <seconds>

Filter output data to start at the specified time in the timeline. For example, if you enter `-start 0.005`, all data before the 5 millisecond mark is not included in the output.

-stop <seconds>

Filter output data to stop at the specified time in the timeline.

-template <name_or_path>

The template to apply. Streamline searches each template in the following order:

1. In the file `captured.xml`.
2. In the built-in templates.
3. In the `Templates` directory.
4. As a file path (`*.st`).

You can apply more than one `-template` option. To create a chart configuration template, see Create a chart configuration template.

-default-template

Ignore the templates from the capture and load Streamline's default template.

-thread <regex>#<TID>

A string specifying the thread to inspect. Like `-process`, it consists of two parts. The first part of the string is a regular expression that matches the name of the thread. The second part is a '#' character followed by the numeric ID of the thread. Both parts can be used together or individually. To use thread filtering, you must specify the process that the threads belong to using `-process`.

Examples of valid arguments are:

- `-thread=Thread(.*)`
- `-thread=Thread(.*)#1234`
- `-thread=#1234`

-timeline

Output the contents of the **Timeline** view.

-warnings[=<name>]

Produce the warnings table. `<name>` specifies the filename to use for the output. Default = `warnings.csv`.

Returns

A capture analysis report.

Example 5-2: Output multiple report types

To output the **Call Paths** and **Functions** data from the capture `thread_001.apc`, enter:

```
streamline-cli -report -callpath -function thread_001.apc
```

Example 5-3: Output timeline data to a CSV file

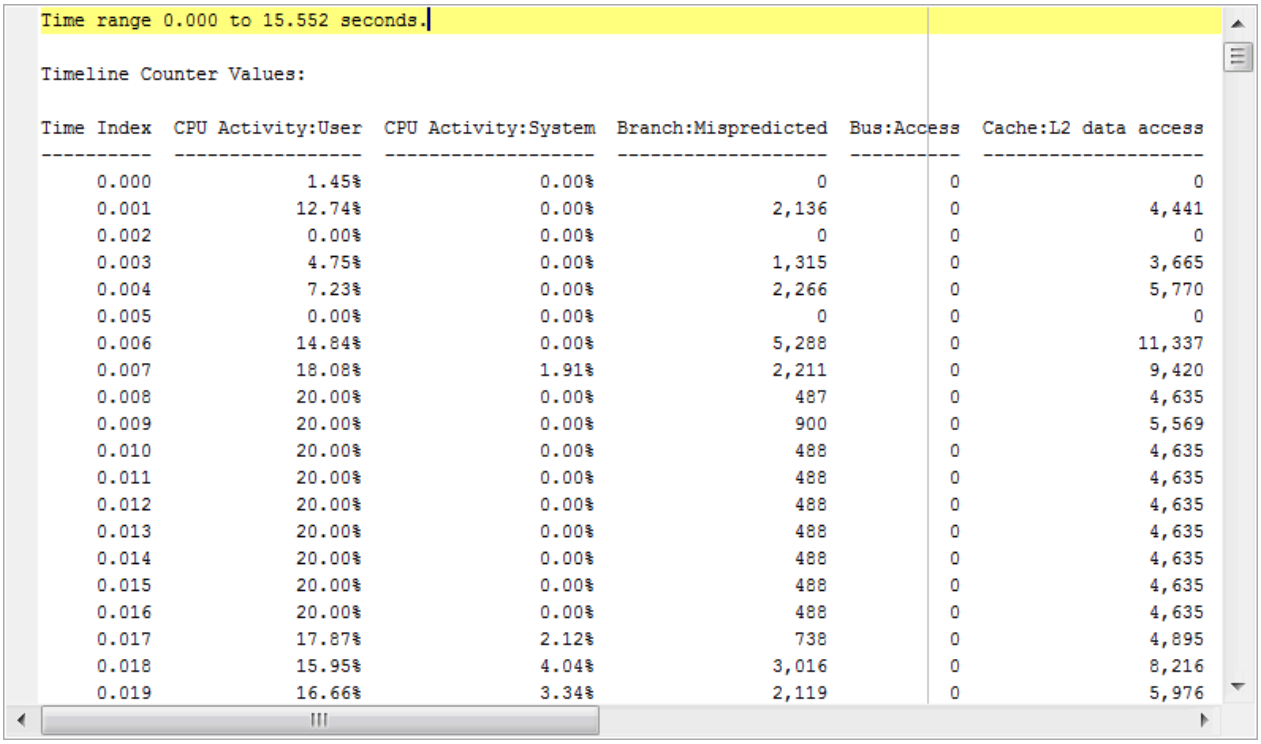
You can output very large reports to a file, then import the data into a spreadsheet application.

To output the timeline data from `capture_001.apc` to a file called `my_timeline.csv` in a directory called `capture_001_reports`, enter:

```
streamline-cli -report -timeline=my_timeline.csv capture_001.apc -o
 capture_001_reports
```

If `my_timeline.csv` already exists in the specified directory, the file is overwritten with the new data.

Figure 5-1: Timeline view data output to a text file



```
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
```

5.7 Import modes

Import a bare-metal trace file and create a capture from it.

Syntax

```
streamline-cli <mode> [options] <files_to_import>
```

Modes

`-import-apc-zip`

Import zipped APC files.

`-import-perf`

Import perf data files.

`-import-barman-raw`

Import a bare-metal raw file.

`-import-stm`

Import a bare-metal trace that was captured through STM. Streamline converts the trace file into a bare-metal raw file.

-import-itm

Import a bare-metal trace that was captured through ITM. Streamline converts the trace file into a bare-metal raw file.

Options

-o, -output <output_directory>

The directory where the capture is created.

-agent-image <elf_image>

The path to the ELF image of the bare-metal code that contains the bare-metal agent code.

-barman-xml <barman.xml>

The path to the `barman.xml` file that the Barman Generator Wizard generated. Required when importing STM and ITM traces.

-assume-sync

Assume that the ITM trace is synchronized from the start.

Returns

An APC capture directory.

Related information

Using the bare-metal generation mechanism from the command line

6. Add custom counters from files and trace events

Customize the more advanced collection and reporting features of Streamline.

6.1 Creating a configuration.xml file

A `configuration.xml` file defines the set of counters and their attributes that you want to collect from the target.

About this task

You normally configure counters using the **Select Counters** dialog box. From this dialog box, you can also export a `configuration.xml` file to use later in a headless workflow. 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 set is sufficient, you do not need to create a `configuration.xml` file.

Procedure

1. Run `gatord --print counters.xml <flags>`, where `<flags>` is the flags that you will use when capturing.
   This command lists all the counters that your target supports.

   Example:

   ```
   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 the output of `--print events.xml` .
   Example:

   ```
   $ gatord --print events.xml
   ...
   ```

```
  <event counter="Linux_block_rq_rd" title="Disk I/O" name="Read" units="B"
 description="Disk I/O Bytes Read"/>
...
```

This example shows that the `Linux_block_rq_rd` counter is the Disk I/O: 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 counter set name in the output of `--print events.xml` .
Example:

```
$ gatord --print events.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 example 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 many events, including Software: Increment, Cache: Instruction refill, and Cache: Inst TLB refill.

After following this step, you now know which events are available for your target.

4. Create an empty `configuration.xml` file. Copy and paste into it the counter and event (if any) attributes from the `<event .../>` nodes for the events you are interested in.
Example:

To add the Disk I/O: Read event, copy the required attributes from `events.xml` into your `configuration.xml` file, so it looks like this:

```
<configurations revision="3">
   <configuration counter="Linux_block_rq_rd"/>
</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 into a new `<configuration>` node:

```
<configurations revision="3">
   <configuration counter="Linux_block_rq_rd"/>
   <configuration counter="ARMv7_Cortex_A9_ccnt" event="0xff"/>
</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"`. Therefore 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 now looks like this:

```
<configurations revision="3">
   <configuration counter="Linux_block_rq_rd"/>
   <configuration counter="ARMv7_Cortex_A9_ccnt" event="0xff"/>
   <configuration counter="ARMv7_Cortex_A9_cnt0" event="0x01"/>
</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.

Results
You now have a `configuration.xml` file for a specific target to capture the events you are interested in.

6.2 Filesystem and ftrace counters

`gatord` supports reading data from the host filesystem and from `ftrace` events.

6.2.1 Filesystem counters

This topic describes the data that `gatord` reads from the host filesystem counters.

The data in an instrumented filesystem node is read once every 100 milliseconds. The default behavior attempts to interpret the contents of the file as the string representation of a positive decimal integer, with an optional trailing newline. This value is turned directly into the counter value. If the value is not a valid string representation of a decimal integer, the counter value is discarded.

For more complex files, you can optionally specify a regular expression using the extended POSIX syntax. The first capture group of the first match of this regex pattern is interpreted as an integer. If this interpretation fails, either because there are no matches or because the data is not an integer, the counter value is discarded.

Filesystem counters use the event XML syntax that is shown in this example:

```
<category name="Filesystem">
  <event counter="filesystem_loginuid"
         path="/proc/self/loginuid"
         title="The loginuid chart"
         name="The loginuid series"
         description="The description of the loginuid series"
         regex="^(\d+)"/>
</category>
```

- The `counter` attribute must be a unique identifier code, and must begin with the `filesystem_` prefix.
- The `path` attribute is the location on the filesystem to be read.
- The `class` attribute is the counter class, which explains how to interpret the data.
- The `title` attribute is the name of the chart in the **Timeline** view that shows the data series. Multiple event counters can have the same title, which means they are all plotted on the same chart.
- The `name` attribute is the name of the data series in the chart in the **Timeline** view.
- The `description` attribute provides some documentation of the counter.
- The `regex` attribute is optional. It contains the extended POSIX regular expression to use to search and extract the counter integer value from the file content. The content of the first match group of the first hit in the search is interpreted as an integer to determine the counter value.

6.2.2 ftrace event counters

This topic describes the data that `gatord` reads from the host `ftrace` event counters.

The data that is stored in an `ftrace` tracepoint event can also be read as each event is emitted. Each trace event accessible by perf has a self-defining binary structure, which is defined by `/sys/kernel/debug/tracing/events/<group>/<event>/format`. The counter is read from the field in this binary structure that is defined by the `arg` attribute in the event XML.

`ftrace` event counters use the event XML syntax that is shown in this example:

```
<events>
<category name="Ftrace">
  <event counter="ftrace_power_cpu_idle"
         tracepoint="power/cpu_idle"
         arg="state"
         title="Power"
         name="cpu_idle"
```

```
          regex="cpu_idle"
          description="Number of times cpu_idle is entered or exited" />
</category>
</events>
```

To filter multithreaded applications at thread level, add `per_cpu="yes" proc="yes"` to the event xml.

The XML syntax for `ftrace` events include the following attributes:

`counter`

The `counter` attribute must be a unique identifier code, and must begin with the `ftrace_` prefix.

`tracepoint`

The `tracepoint` attribute is the source `ftrace` event that you want to use. Specify `tracepoint` in the form `<group>/<event>`, which equates to the path `/sys/kernel/debug/tracing/events/<group>/<event>` where the tracepoint is defined.

`arg`

The optional `arg` attribute is used with `tracepoint` to extract a specified counter value from each tracepoint event, otherwise the number of events is counted.

`regex`

The `regex` value is used to either count the number of times a tracepoint is seen, or extracts an argument attribute to use as a counter value.

To filter this event by thread, add `per_cpu="yes" proc="yes"` to the event xml. The tracepoint must correctly report the `pid` associated with the event as part of the tracepoint common fields.

Attribute behavior with the efficient ftrace collection option

The behavior of the `tracepoint`, and `regex` attributes depends on if you are using the efficient ftrace option.

When the **Use more efficient Ftrace collection** checkbox is selected, and `arg` is not set, `tracepoint` identifies the tracepoint name and counts the number of times that it is seen. If `arg` is set, the specified argument attribute is used as the counter value for each data point.

When the **Use more efficient Ftrace collection** checkbox is cleared, `regex` is used to match with the `ftrace` log text. `regex` must be in one of the following formats:

- `tracepoint_name:.*` to count the number of times a tracepoint is seen
- `tracepoint_name:.*arg=([0-9]+).*` to extract an argument attribute as the counter value


Note

- You can only generate one Streamline counter per tracepoint.
- The event XML must contain `tracepoint` and `regex`, but `arg` is optional. If `regex` is used to capture an `arg` value, and `arg` is not specified in your event XML, the output is determined by whether you are using efficient ftrace collection.

6.2.3 Create filesystem and ftrace counters

You can add counters that are based on data that is loaded from files, such as `/dev` , `/sys` , and `/proc` , and record `ftrace` events using perf.

Procedure

1. Create an `events-<xxx>.xml` file.
2. Add an event entry for each counter.


Note

The `gatord` source code comes with an example file that is called `events-Filesystem.xml` and contains commented-out entries.

3. Do one of the following:
   - Rebuild `gatord`, placing the `events-<xxx>.xml` file in the same directory as the `gatord` Makefile before building.
   - Restart `gatord` using the `-E` command-line option to specify the location of your `events-<xxx>.xml` file.

   If you choose to use the separate XML file that is specified on the command line, the file format is slightly different. You must specify the wrapping `events` tag in addition to the category. For example:

```
<events>
  <category name="Filesystem">
    <event counter="filesystem_loginuid"
           path="/proc/self/loginuid"

           title="The loginuid chart"
           name="The loginuid series"
           description="The description of the loginuid series"/>
  </category>
</events>
```

6.3 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 report the current, absolute value. Use the `average`, `maximum`, and `minimum` filters with absolute counters.

Delta

Delta counters 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 are the same as delta counters, except the exact time is known when the data occurs, so no interpolation is calculated.

Activity

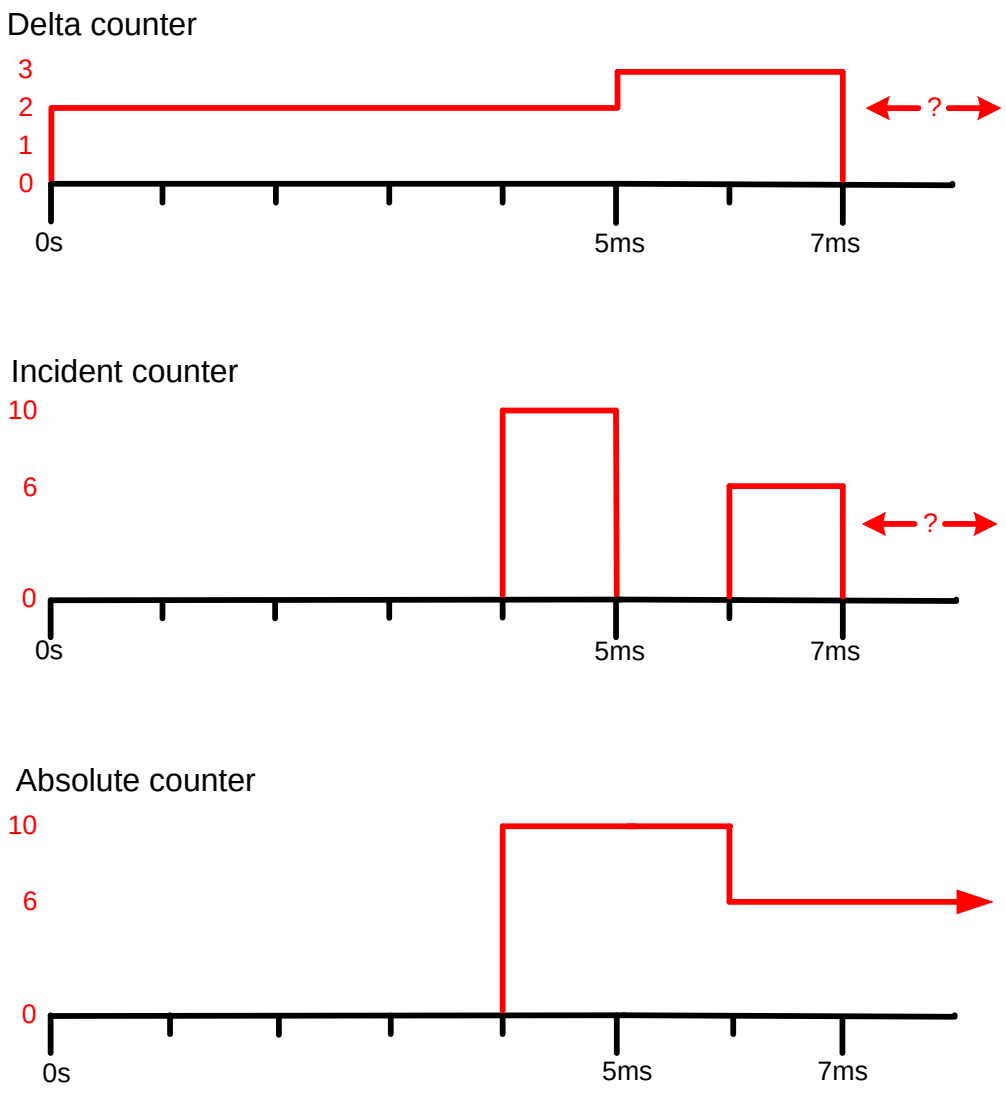
Activity counters report changes in processor activity or state. Use the `average` filter with activity counters.

The following figure illustrates how the same data that is received from `gatord` 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 show 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 6-1: Counter classes



Related information

- Chart configuration series options

7. Keyboard shortcuts

Learn about which keyboard shortcuts you can use to navigate and perform tasks in different views.

7.1 Keyboard shortcuts in the Live and Timeline views

While you can navigate every report 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 lines from bookmarks on and off.

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

L
: Opens the Log view and highlights any annotations that are covered by the cross-section marker.

- Zooms in one level.
- Zooms out one level.

Ctrl+scroll wheel
: Zooms smoothly on the area around the mouse cursor.

Various shortcut keys 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 information

Details panel modes on page 82
Contextual menu options in the Live and Timeline views on page 97

7.2 Keyboard shortcuts in the table views

While you can navigate every table report 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 one page in the current report.

Page Down
: Moves down one page in the current report.

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

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.

7.3 Keyboard shortcuts in the code view

Keyboard shortcuts help you to quickly navigate through your source code and disassembly instructions in the **Code** view.

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 to the top of the source file.

End
: Takes you to the bottom of the function that contains the currently selected row. 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.

Page Down
: Moves down one page.

Related information

Analyze your code on page 99

8. Importing raw perf data

This appendix explains how to import event data created by the Linux perf command-line tools.

8.1 Import raw 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.

To import the event trace, click **Import Capture File(s)...** and select 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 `gatord` recognizes, are configured as though `gatord` captured them. Counters that `gatord` 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 `gatord`, 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** mode and **Core Map** mode, 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>

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:

- Add the `-T` argument to `perf record` to include timestamp information in events.
- Add the `S` modifier to events to read sample values.
- Use the `-P` argument to specify a sample period greater than zero.
- Use the `-F` argument to read the sampling frequency alongside recording the sample period.

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.

9. JIT profiling support

This appendix explains how to profile JIT execution and provides an example of the workflow.

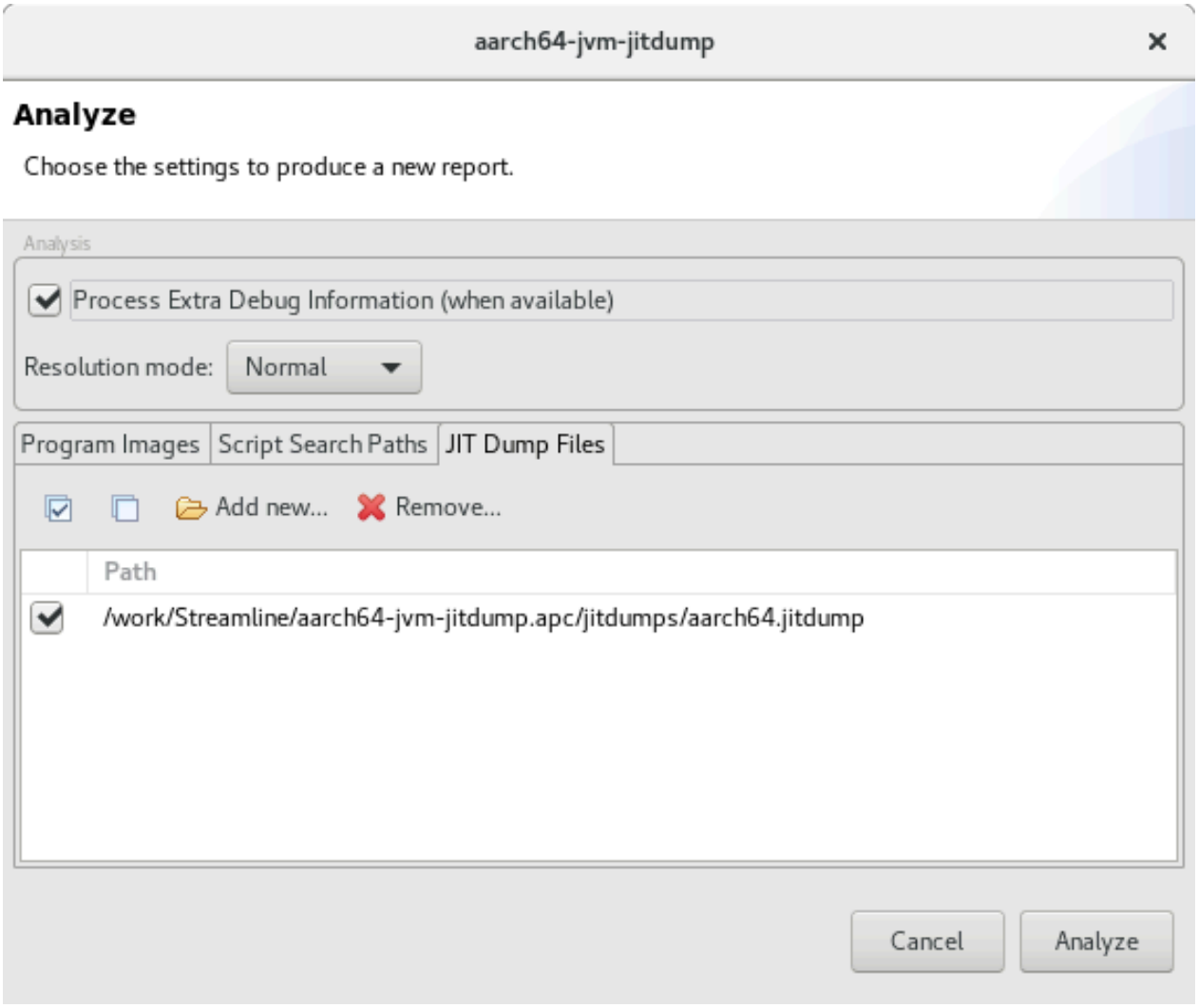
9.1 JIT profiling using gator

Profile Just In Time (JIT) execution by using gatord on a supported JIT engine, to collect trace information when running your program. Then you can import the generated `jitdump` data into Streamline for analysis.

Common supported languages are: Node.js, Chrome V8 Javascript, and Java.

You can specify the `jitdump` file paths in the **Analyze** dialog box, **JIT Dump Files** tab, which is shown in the following figure.

Figure 9-1: Analyze dialog box (JIT Dump Files tab).



Streamline processes the files to extract the source and line information, and shows the generated machine code in the **Code** view.

Tracing program execution

Follow the steps below to configure the JVM and then use gatord to trace execution:

Use the `libperf-jvmti.so` agent library which is provided in source form as part of the perf tools sources. See https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=209045adc2bbdb2b315fa5539cec54d01cd3e7db.

- Download kernel sources and install the JDK on your target.
- Modify the JVM agent in the perf tools in order to use CLOCK_MONOTONIC_RAW instead of CLOCK_MONOTONIC.

In Linux v6.11 this can be done by applying the following patch to the agent source code:

```
diff --git a/tools/perf/jvmti/jvmti_agent.c b/tools/perf/jvmti/jvmti_agent.c
index 526dcaf9f..1286b870e 100644
--- a/tools/perf/jvmti/jvmti_agent.c
+++ b/tools/perf/jvmti/jvmti_agent.c
@@ -102,7 +102,7 @@ get_arch_timestamp(void)
 }

 #define NSEC_PER_SEC   1000000000
-static int perf_clk_id = CLOCK_MONOTONIC;
+static int perf_clk_id = CLOCK_MONOTONIC_RAW;

 static inline uint64_t
 timespec_to_ns(const struct timespec *ts)
```

- Compile perf tools by running `make`, setting the `JDIR` flag to the location of the JDK:

  ```
  cd <KERNEL>/tools/perf
  make JDIR=<PATH_TO_JDK>
  ```

  The generated library is in `<KERNEL>/tools/perf`.
- Run the `java` command using `gatord`. You must add `-agentpath:<ABSOLUTE_PATH_TO_libperf_jvmti.so>` to the JVM arguments.

  For example, to perform a local capture:

  ```
  ./gatord ... --output example-jvm-jit --app java -agentpath:<KERNEL>/tools/perf/
  libperf-jvmti.so ...
  ```

  If the JVM is configured correctly, the following message is shown in the console with the paths to the generated `jitdump` files:

  ```
  Gator ready
  Starting command: java...
  java: jvmti: jitdump in /home/user/.debug/jit/java-jit-20241108.XXvrbtM7/
  jit-2331608.dump
  ```

  Import the generated capture `example-jvm-jit.apc` into Streamline and include the generated `jitdump` files in the **Analyze** dialog box as shown in the figure above.

Related information

Streamline and JITed code

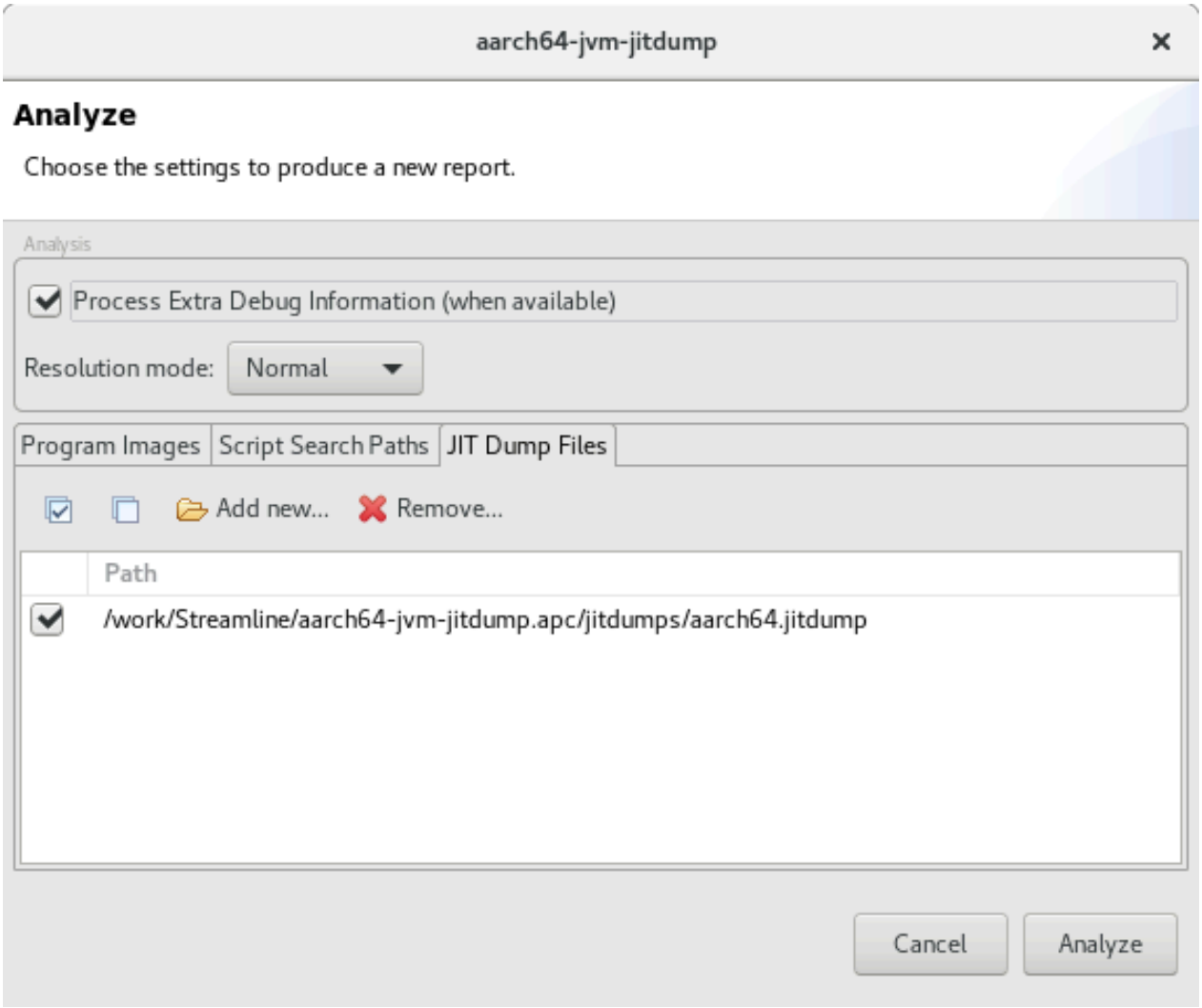
9.2 JIT profiling using perf

Profile Just In Time (JIT) execution by using perf on a supported JIT engine, to collect trace information when running your program. Then you can import the generated `jitdump` data into Streamline for analysis.

Common supported languages are: Node.js, Chrome V8 Javascript, and Java.

You can specify the `jitdump` file paths in the **Analyze** dialog box, **JIT Dump Files** tab, which is shown in the following figure.

Figure 9-2: Analyze dialog box (JIT Dump Files tab).



Streamline processes the files to extract the source and line information, and shows the generated machine code in the **Code** view.


Note

- You must pass `-k CLOCK_MONOTONIC` to `perf record` when profiling using `libperf-jvmti.so`. `libperf-jvmti.so` uses CLOCK_MONOTONIC but, by default, `perf` does not use the same clock.

An example of how to profile and analyze JIT execution could be:

1. To capture profiling data using `perf record`, you must use a CLOCK_MONOTONIC clock:

   ```
   sudo perf record -k CLOCK_MONOTONIC -a -c 1 -e "{sched:sched_switch, cpu-clock/
   period=10000000/, instructions, branch-misses}:S" java -agentpath:<path-to-
   libperf-jvmti.so> <java-main-class>"
   ```

   To include call stacks, pass `--call-graph=fp` to `perf` and pass `-XX:+PreserveFramePointer` to `java`.
2. Copy `perf.data` and `jitdump` files, found in `~/.debug/jit/`, to your host:

   ```
   scp perf.data ~/.debug/jit/<path_to_jitdump_file> hostmachine:
   ```
3. Import `perf.data`.
4. Analyze the imported capture by attaching the `jitdump` file that is transferred in step 2.

Tracing program execution

The steps that you must take to trace execution depends on the particular VM you are using:

- Node.js and V8

  Run `perf` with the `--perf-prof` argument:

  ```
  perf record ... node --perf-prof ...
  ```

  See https://v8.dev/docs/linux-perf.
- JVM

  Use the `libperf-jvmti.so` agent library which is provided in source form as part of the perf tools sources. See https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=209045adc2bbdb2b315fa5539cec54d01cd3e7db.
  - Download kernel sources and install the JDK on your target.
  - Compile perf tools by running `make`, setting the `JDIR` flag to the location of the JDK:

    ```
    cd <KERNEL>/tools/perf
    make JDIR=<PATH_TO_JDK>
    ```
  - The resultant library is in `<KERNEL>/tools/perf`.

- Run the `java` command with the `-agentpath:<ABSOLUTE_PATH_TO_libperf_jvmti.so>` argument:

```
perf record ... java -agentpath:/home/user/linux-src/tools/perf/
libperf_jvmti.so ...
```

Related information

Streamline and JITed code

10. Troubleshooting Streamline

Troubleshoot known Streamline issues.

10.1 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.

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

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

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 Settings** dialog box.

1. In the Program Images section, click **Add image**.
2. Navigate to your shared library, and select the required file.
3. Click **Open**.

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 Arm®v7 and earlier architectures. Also, check the **Call stack unwinding** option in the **Capture Settings** dialog box.

If frame pointers are set correctly, the disassembly of the code contains 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.
- Streamline does not walk the stack for statically linked drivers.

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

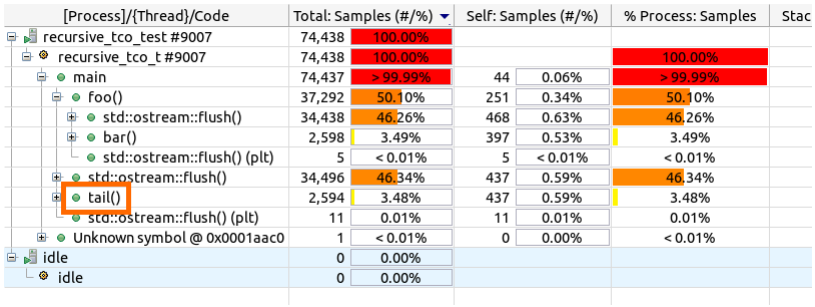
This problem can be because of code inlining that is done by the compiler. To turn inlining off, add `-fno-inline` as an option during compilation.

Also, check that your debug symbols are correct.

Functions are displayed as a separate call path instead of in the thread where they are called.

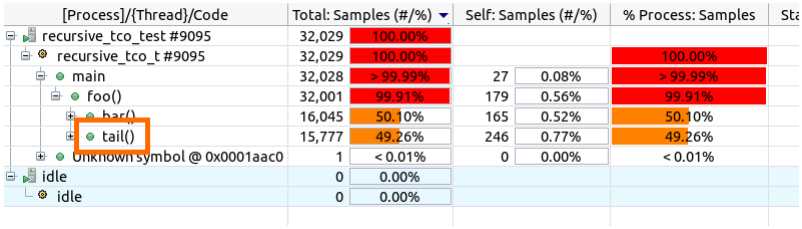
Check if the TailCall Optimization (TCO) compiler flag is enabled. Enabling TCO prevents Streamline from unwinding the call stack correctly. For more information about TCO, see Overview of optimizations in the Arm Compiler Optimization guide.

Figure 10-1: Function displayed in a separate call path when TCO is enabled



To display the functions in the correct thread, disable TCO, then recapture the Streamline profile.

Figure 10-2: Function displayed correctly when TCO is disabled



If you still need help with your capture, send the `gator-log.txt` file, along with your capture, to the support team. The log file is in the same directory as the capture, and is available after the capture has finished. It records basic debugging and support information, such as errors, warnings, and information about the device.

- If you are using Streamline as part of Arm® Performance Studio, contact Arm at performancestudio@arm.com.
- If you are using Streamline as part of Arm® Development Studio, open an Arm Support case.

To create a `gator-log.txt` file for a local capture, enable the `--capture-log` command line option.

Related information

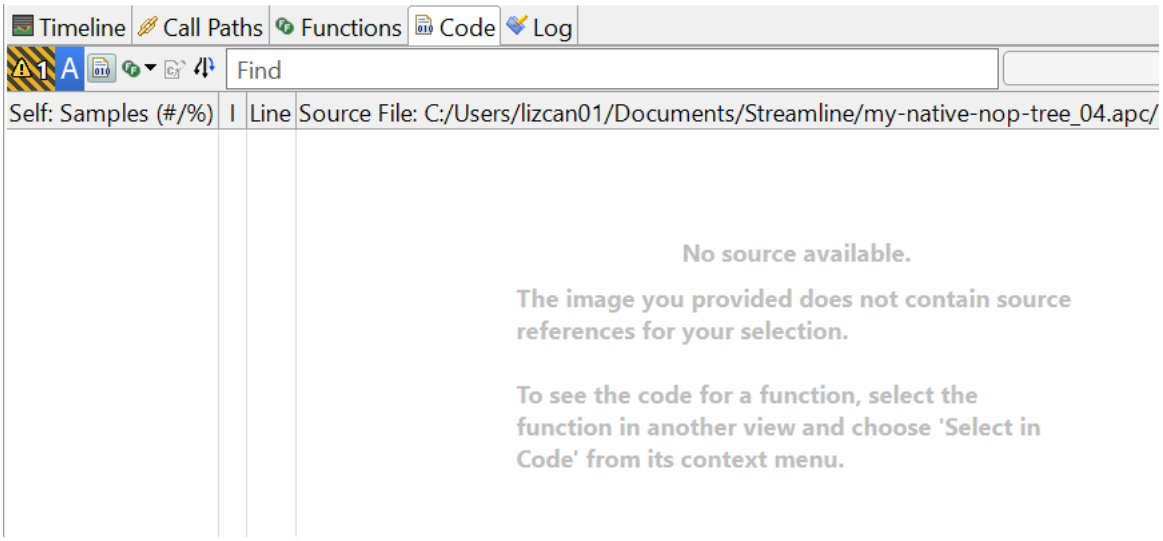
Troubleshooting missing source files on page 139

10.2 Troubleshooting missing source files

If your source files are located in the compilation directory, Streamline automatically locates and displays the source code in the **Code** view. If Streamline cannot locate your source files, a missing source file message is shown.

The missing source file message is shown in the source file section of the **Code** view.

Figure 10-3: Missing source file message



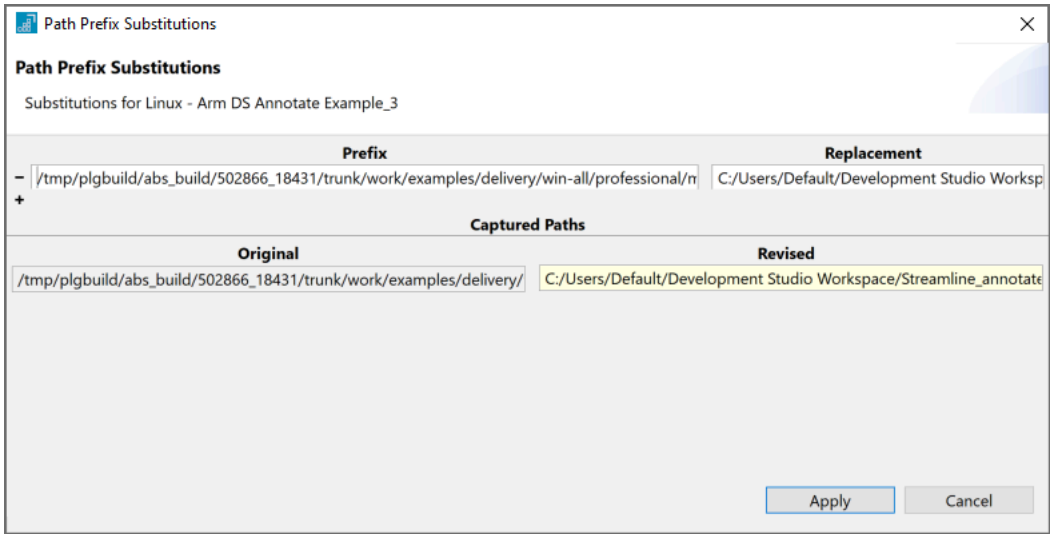
Source files are not located in the compilation directory

Solution

Set up path substitutions to replace the path that Streamline uses to find the source code. You can replace the source path in two ways:

- Click the link under the missing file message, which opens a standard file dialog box. Locating the file from this dialog box creates an entry in the dialog box to appropriately map the file.
- Click **Set Path Prefix Substitutions** on the toolbar, which opens the **Path Prefix Substitutions** dialog box.

Figure 10-4: Path Prefix Substitutions dialog box



1. To substitute a file path, click the plus symbol.
2. Enter valid paths in the **Prefix** and **Replacement** fields.
3. Click **Apply**.

If the path in the **Replacement** field contains code that matches the code used in the capture, Streamline populates the view with your source code and the statistical overlay.

10.3 Troubleshooting perf import issues

This topic describes potential error messages and warnings related to importing perf data.

`Missing cluster map.` warning shown in the Timeline view.

Streamline attempts to determine the cluster configuration based on which processor, or subset of processors, each recorded event is associated with. Streamline also attempts to assign appropriate CPU names, such as Arm®Cortex® A53 and Arm®Cortex® A57, to these clusters.

This warning means that it is not possible to know from the recorded data what the CPU 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 fail, which means that processor PMU counters are not separated into different clusters. Therefore counter values appear as though there is no clustering.

10.4 Troubleshooting Streamline crashes during a capture

If you run a capture session but Streamline crashes, you can locate the Java Virtual Machine (JVM) crash report and seek help from the Arm Support team.

Cause

The are various reasons which could cause Streamline to crash and exit during a capture session. Arm Support can help you to diagnose the cause and identify a solution.

Solution

Locate your JVM crash report file and contact Arm:

- If you are using Streamline as part of Arm® Performance Studio, contact Arm at performancestudio@arm.com.
- If you are using Streamline as part of Arm® Development Studio, open an Arm Support case.

By default, the file name for JVM crash reports is `hs_err_pid<pid>.log` and is created in the working directory, unless:

- You have manually set a different crash log location using the `-XX:ErrorFile=<file>` JVM argument.

  
  Note

  You can check or edit the value for this argument in the file that is appropriate for your OS and usage, either GUI or command line:

  - On Linux: in `/.Streamline-gui.ini` (GUI) or in `./.Streamline-cli.ini` (command line)
  - On MacOS: in `"./Streamline.app/Contents/Eclipse/Streamline-gui.ini` (GUI) or in `./Streamline.app/Contents/Eclipse/Streamline-cli.ini` (command line)
  - On Windows: in `./Streamline-gui.ini` (GUI) or in `./Streamline-cli.in` (command line)

  You can also set the value through the command line, for example, using `streamline-gui -vmargs -XX:Errorfile=<file>`.

- Your working directory cannot be written to. Instead, you can find your crash reports at the default location for your OS:
  - On Linux: in `tmp`
  - On MacOS: in `tmp`
  - On Windows: in the location set by the `TMP` or `TEMP` environment variables. The default location is: `C:Users<username>AppDataLocalTemp`

If you still need help with your capture, send the `gator-log.txt` file, along with your capture, to the support team. The log file is in the same directory as the capture, and is available after the capture has finished. It records basic debugging and support information, such as errors, warnings, and information about the device.

- If you are using Streamline as part of Arm® Performance Studio, contact Arm at performancestudio@arm.com.
- If you are using Streamline as part of Arm® Development Studio, open an Arm Support case.

To create a `gator-log.txt` file for a local capture, enable the `--capture-log` command line option.

Proprietary Notice

This document is protected by copyright and other related rights and the use or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm Limited ("Arm"). No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether the subject matter of this document infringes any third party patents.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

This document may include technical inaccuracies or typographical errors. THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, any patents, copyrights, trade secrets, trademarks, or other rights.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any permitted use, duplication, or disclosure of this document complies fully with any relevant

export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of this document shall prevail.

The validity, construction and performance of this notice shall be governed by English Law.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

Product and document information

Read the information in these sections to understand the release status of the product and documentation, and the conventions used in Arm documents.

Product status

All products and services provided by Arm require deliverables to be prepared and made available at different levels of completeness. The information in this document indicates the appropriate level of completeness for the associated deliverables.

Product completeness status

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

Revision history

These sections can help you understand how the document has changed over time.

Document release information

The Document history table gives the issue number and the released date for each released issue of this document.

Document history

Issue	Date	Confidentiality	Change
0907-00	31 July 2025	Non-Confidential	New document for v9.7
0906-00	1 May 2025	Non-Confidential	New document for v9.6
0905-01	20 March 2025	Non-Confidential	Updated document for v9.5
0905-00	6 February 2025	Non-Confidential	New document for v9.5
0904-00	28 November 2024	Non-Confidential	New document for v9.4
0903-00	5 September 2024	Non-Confidential	New document for v9.3
0902-00	7 June 2024	Non-Confidential	New document for v9.2
0901-00	12 April 2024	Non-Confidential	New document for v9.1
0900-00	15 February 2024	Non-Confidential	New document for v9.0
0809-00	23 November 2023	Non-Confidential	New document for v8.9

Issue	Date	Confidentiality	Change
0808-00	28 September 2023	Non-Confidential	New document for v8.8
0807-00	3 August 2023	Non-Confidential	New document for v8.7
0806-00	8 June 2023	Non-Confidential	New document for v8.6
0805-00	20 April 2023	Non-Confidential	New document for v8.5
0804-00	14 February 2023	Non-Confidential	New document for v8.4
0803-00	17 November 2022	Non-Confidential	New document for v8.3
0802-00	19 August 2022	Non-Confidential	New document for v8.2
0801-00	20 May 2022	Non-Confidential	New document for v8.1
0800-00	18 February 2022	Non-Confidential	New document for v8.0
0709-00	18 November 2021	Non-Confidential	New document for v7.9
0708-00	20 August 2021	Non-Confidential	New document for v7.8

Change history

For information about the functional changes to Streamline, see the Arm Performance Studio Release Notes.

Conventions

The following subsections describe conventions used in Arm documents.

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: developer.arm.com/glossary.

Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
italic	Citations.
bold	Interface elements, such as menu names. Terms in descriptive lists, where appropriate.

Convention	Use
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace underline`	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`<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	Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, **IMPLEMENTATION DEFINED**, **IMPLEMENTATION SPECIFIC**, **UNKNOWN**, and **UNPREDICTABLE**.

Caution — We recommend the following. If you do not follow these recommendations your system might not work.

Warning — Your system requires the following. If you do not follow these requirements your system will not work.

Danger — You are at risk of causing permanent damage to your system or your equipment, or harming yourself.

Note — This information is important and needs your attention.

Tip — A useful tip that might make it easier, better or faster to perform a task.

Remember — A reminder of something important that relates to the information you are reading.

Useful resources

This document contains information that is specific to this product. See the following resources for other useful information.

Arm documents are available on developer.arm.com/documentation.

Confidential documents are only available to licensees, when logged in. Each document link in the tables below provides direct access to the online version of the document.

Arm product resources	Document ID	Confidentiality
Android performance triage with Streamline	102540	Non-Confidential
Arm Compiler Optimization Guide	102654	Non-Confidential
Arm Development Studio Getting Started Guide	101469	Non-Confidential
Arm Streamline Target Setup Guide for Android	101813	Non-Confidential
Arm Streamline Target Setup Guide for Bare-metal	101815	Non-Confidential
Arm Streamline Target Setup Guide for Linux	101814	Non-Confidential
Arm community blogs	Non-Confidential	Unspecified
User-based Licensing User Guide	102516	Non-Confidential

Non-Arm resources	Document ID	Organization
Debugging information in separate debug files for GDB project packages	-	Sourceware.org
Specification for intent arguments	-	adb intent arguments
V8 Linux perf integration	-	V8