## Arm® Streamline Performance Analyzer

### User Guide

Copyright © 2010–2019 Arm Limited or its affiliates. All rights reserved.

### Release Information

<table>
<thead>
<tr>
<th>Issue</th>
<th>Date</th>
<th>Confidentiality</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>19 October 2010</td>
<td>Non-Confidential</td>
<td>ARM Streamline Performance Analyzer 1.0</td>
</tr>
<tr>
<td>B</td>
<td>19 January 2011</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.4</td>
</tr>
<tr>
<td>C</td>
<td>19 April 2011</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.5</td>
</tr>
<tr>
<td>D</td>
<td>19 July 2011</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.6</td>
</tr>
<tr>
<td>E</td>
<td>19 September 2011</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.7</td>
</tr>
<tr>
<td>F</td>
<td>19 November 2011</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.8</td>
</tr>
<tr>
<td>G</td>
<td>19 February 2012</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.9</td>
</tr>
<tr>
<td>H</td>
<td>19 May 2012</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.10</td>
</tr>
<tr>
<td>I</td>
<td>19 July 2012</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.11</td>
</tr>
<tr>
<td>J</td>
<td>19 October 2012</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.12</td>
</tr>
<tr>
<td>K</td>
<td>19 December 2012</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.13</td>
</tr>
<tr>
<td>L</td>
<td>15 March 2013</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.14</td>
</tr>
<tr>
<td>M</td>
<td>14 June 2013</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.15</td>
</tr>
<tr>
<td>N</td>
<td>13 September 2013</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.16</td>
</tr>
<tr>
<td>O</td>
<td>13 December 2013</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.17</td>
</tr>
<tr>
<td>P</td>
<td>14 March 2014</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.18</td>
</tr>
<tr>
<td>Q</td>
<td>27 June 2014</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.19</td>
</tr>
<tr>
<td>R</td>
<td>17 October 2014</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.20</td>
</tr>
<tr>
<td>S</td>
<td>20 March 2015</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.21</td>
</tr>
<tr>
<td>T</td>
<td>15 July 2015</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.22</td>
</tr>
<tr>
<td>U</td>
<td>15 October 2015</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.23</td>
</tr>
<tr>
<td>V</td>
<td>15 March 2016</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.24</td>
</tr>
<tr>
<td>W</td>
<td>31 May 2016</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.25.0.1</td>
</tr>
<tr>
<td>X</td>
<td>11 August 2016</td>
<td>Non-Confidential</td>
<td>Update for DS-5 version 5.25.0.2</td>
</tr>
<tr>
<td>0600-00</td>
<td>30 September 2016</td>
<td>Non-Confidential</td>
<td>Update for Streamline version 6.0. Document numbering scheme changed.</td>
</tr>
<tr>
<td>0601-00</td>
<td>04 November 2016</td>
<td>Non-Confidential</td>
<td>Update for Streamline version 6.1</td>
</tr>
<tr>
<td>0602-00</td>
<td>10 March 2017</td>
<td>Non-Confidential</td>
<td>Update for Streamline version 6.2</td>
</tr>
<tr>
<td>0603-00</td>
<td>12 May 2017</td>
<td>Non-Confidential</td>
<td>Update for Streamline version 6.3. Document title changed.</td>
</tr>
<tr>
<td>0604-00</td>
<td>30 August 2017</td>
<td>Non-Confidential</td>
<td>Update for Streamline version 6.4</td>
</tr>
<tr>
<td>0605-00</td>
<td>08 November 2017</td>
<td>Non-Confidential</td>
<td>Update for Streamline version 6.5</td>
</tr>
<tr>
<td>0606-00</td>
<td>08 February 2018</td>
<td>Non-Confidential</td>
<td>Update for Streamline version 6.6</td>
</tr>
<tr>
<td>0607-00</td>
<td>08 June 2018</td>
<td>Non-Confidential</td>
<td>Update for Streamline version 6.7</td>
</tr>
<tr>
<td>0608-00</td>
<td>16 October 2018</td>
<td>Non-Confidential</td>
<td>Update for Arm Streamline version 6.8</td>
</tr>
</tbody>
</table>
### Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice 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. **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 implementations infringe any third party patents.

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, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

**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.**

This document consists solely of commercial items. You shall be responsible for ensuring that any 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.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. 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 the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at [http://www.arm.com/company/policies/trademarks](http://www.arm.com/company/policies/trademarks).

Copyright © 2010–2019 Arm Limited (or its affiliates). All rights reserved.


110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

### Confidenitality Status

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

Unrestricted Access is an Arm internal classification.

### Product Status

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

### Issue Log

<table>
<thead>
<tr>
<th>Issue</th>
<th>Date</th>
<th>Confidentiality</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>0609-00</td>
<td>28 February 2019</td>
<td>Non-Confidential</td>
<td>Update for Arm Streamline version 6.9.</td>
</tr>
<tr>
<td>0700-00</td>
<td>03 May 2019</td>
<td>Non-Confidential</td>
<td>Update for Arm Streamline version 7.0.</td>
</tr>
</tbody>
</table>
Web Address

http://www.arm.com
Contents

Arm® Streamline Performance Analyzer User Guide

Preface

About this book ..................................................... ..................................................... 13

Chapter 1  Introduction to Arm Streamline Performance Analyzer

1.1 Features of Arm Streamline editions ................................................................. 1-17
1.2 Overview of Arm Streamline Performance Analyzer ...................................... 1-18
1.3 New features and updates .................................................................................. 1-21
1.4 Importing the capture examples ........................................................................ 1-25
1.5 Profile an application on a non-root Android device ....................................... 1-26
1.6 Install the default profiling agent from the UI ............................................... 1-28
1.7 Standards compliance in Arm Streamline ...................................................... 1-29

Chapter 2 Target Setup

2.1 Arm Streamline prerequisites ........................................................................... 2-31
2.2 Setup scenarios for Linux targets ...................................................................... 2-32
2.3 Preparing and building your Linux kernel ....................................................... 2-36
2.4 Required kernel configuration menu options ................................................... 2-38
2.5 Building and installing the gator daemon ........................................................ 2-40
2.6 Building the gator module ............................................................................... 2-42
2.7 Running the gator daemon on your target ...................................................... 2-44
2.8 Profiling applications without root access ...................................................... 2-46
2.9 Capturing data locally on a target .................................................................... 2-47
2.10 Stopping the gator daemon ............................................................................. 2-48
2.11 gatord command-line options ........................................................................ 2-49
**List of Figures**

**Arm® Streamline Performance Analyzer User Guide**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Arm Streamline graphical user interface</td>
<td>1-20</td>
</tr>
<tr>
<td>3-1</td>
<td>Target view</td>
<td>3-58</td>
</tr>
<tr>
<td>3-2</td>
<td>Starting a capture session</td>
<td>3-59</td>
</tr>
<tr>
<td>3-3</td>
<td>Connection Browser in Arm Streamline</td>
<td>3-60</td>
</tr>
<tr>
<td>3-4</td>
<td>Automatically setting up Arm Streamline on the target</td>
<td>3-61</td>
</tr>
<tr>
<td>3-5</td>
<td>Arm Streamline Data view</td>
<td>3-63</td>
</tr>
<tr>
<td>3-6</td>
<td>Color-coded captures</td>
<td>3-64</td>
</tr>
<tr>
<td>3-7</td>
<td>Analyze dialog box</td>
<td>3-65</td>
</tr>
<tr>
<td>3-8</td>
<td>Duplicating a capture</td>
<td>3-67</td>
</tr>
<tr>
<td>3-9</td>
<td>Merge captures dialog box</td>
<td>3-68</td>
</tr>
<tr>
<td>3-10</td>
<td>An imported SoC Designer System Analyzer profile in Heat Map view</td>
<td>3-74</td>
</tr>
<tr>
<td>3-11</td>
<td>An imported SoC Designer System Analyzer profile in CAM view</td>
<td>3-75</td>
</tr>
<tr>
<td>4-1</td>
<td>Capture &amp; Analysis Options dialog box</td>
<td>4-78</td>
</tr>
<tr>
<td>5-1</td>
<td>Target view - Counter Configuration</td>
<td>5-86</td>
</tr>
<tr>
<td>5-2</td>
<td>Counter Configuration dialog box</td>
<td>5-87</td>
</tr>
<tr>
<td>5-3</td>
<td>Interface drop-down menu</td>
<td>5-89</td>
</tr>
<tr>
<td>5-4</td>
<td>Setting up event-based sampling</td>
<td>5-111</td>
</tr>
<tr>
<td>5-5</td>
<td>SPE counter configuration</td>
<td>5-113</td>
</tr>
<tr>
<td>6-1</td>
<td>Live view</td>
<td>6-116</td>
</tr>
<tr>
<td>6-2</td>
<td>Dead process in Live view</td>
<td>6-117</td>
</tr>
<tr>
<td>6-3</td>
<td>Interactive counters panel and Live view</td>
<td>6-117</td>
</tr>
<tr>
<td>6-4</td>
<td>Timeline view</td>
<td>6-120</td>
</tr>
<tr>
<td>6-5</td>
<td>Charts specific to a Mali Midgard-based target</td>
<td>6-122</td>
</tr>
</tbody>
</table>
Preface

This preface introduces the Arm® Streamline Performance Analyzer User Guide.

It contains the following:
• About this book on page 13.
About this book

This book describes how to set up and use the Arm® Streamline Performance Analyzer to capture, analyze, and display real-time performance measurements from your system.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction to Arm Streamline Performance Analyzer
Arm Streamline Performance Analyzer is a tool for visualizing the performance of code executing on an Arm-based target CPU, and rendering workloads running on a Mali GPU. This chapter provides an overview of how Arm Streamline works, and links to the new features and updates in this release.

Chapter 2 Target Setup
To get started using Arm Streamline, you must ensure that gator, the mechanism that Arm Streamline uses to communicate with your target, is running on the target. You can either build and install gator yourself, or you can use a pre-built version. This chapter describes how to set up your target device for Arm Streamline.

Chapter 3 Target and Data Views
Describes how to use the Streamline Target view to select a target, configure, start, and stop a capture session and the Streamline Data view to manage your existing captures.

Chapter 4 Capture and Analysis Options
Describes how to use the Capture & Analysis Options dialog box to change capture session settings, such as duration, sample rate, and buffer size.

Chapter 5 Counter Configuration
Describes how to use the Counter Configuration dialog box to select the events that Arm Streamline collects.

Chapter 6 Live View and Timeline View
Describes the Live and Timeline views, which display charts showing the data collected during the capture session. Live view is displayed while the capture takes place, and charts the data in real time. Timeline view is displayed after the capture session ends and the data has been analyzed. It provides additional information in a details panel.

Chapter 7 Table Views: Call Paths and Functions
Describes the Call Paths and Functions views, which provide tabular data about the capture.

Chapter 8 Code View
Describes the Code view, which provides statistics for lines of source code and for disassembled instructions.

Chapter 9 Code Annotations
Describes the Arm Streamline Annotate feature. It enables you to add annotations to your code, which are propagated into the Timeline and Log views.

Chapter 10 Log View
Describes the Log view, which lists the annotations generated in your code along with information about them.

Chapter 11 Capturing Energy Data
Describes how to set up and use the Arm Energy Probe with Arm Streamline to view the power metrics of code running on target hardware.

Chapter 12 Advanced gator Customizations
Describes how to customize the more advanced collection and reporting features of Arm Streamline.
Chapter 13 Bare-Metal Support
Describes the bare-metal support available within Arm Streamline.

Chapter 14 Fastline
Describes the Fastline Wizard, which automates the process of profiling Fast Models with Arm Streamline.

Chapter 15 Using Arm Streamline on the Command Line
Describes how to use the streamline command to access much of the functionality of Arm Streamline from the command line.

Chapter 16 JIT and Python Profiling Support
Describes how to profile Just In Time (JIT) and Python execution with Arm Streamline.

Chapter 17 Troubleshooting Common Arm Streamline Issues
Describes how to troubleshoot some common Arm Streamline issues.

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

See the Arm® Glossary for more information.

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

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

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

monospace
 Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

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

monospace bold
 Denotes language keywords when used outside example code.

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

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

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

Feedback
Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:
• The product name.
• The product revision or version.
• An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

If you have comments on content then send an e-mail to errata@arm.com. Give:
• The title Arm Streamline Performance Analyzer User Guide.
• The number 100769_0700_00_en.
• If applicable, the page number(s) to which your comments refer.
• A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

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

Other information

• Arm® Developer.
• Arm® Information Center.
• Arm® Technical Support Knowledge Articles.
• Technical Support.
• Arm® Glossary.
Chapter 1
Introduction to Arm Streamline Performance Analyzer

Arm Streamline Performance Analyzer is a tool for visualizing the performance of code executing on an Arm-based target CPU, and rendering workloads running on a Mali GPU. This chapter provides an overview of how Arm Streamline works, and links to the new features and updates in this release.

It contains the following sections:

• 1.1 Features of Arm Streamline editions on page 1-17.
• 1.2 Overview of Arm Streamline Performance Analyzer on page 1-18.
• 1.3 New features and updates on page 1-21.
• 1.4 Importing the capture examples on page 1-25.
• 1.5 Profile an application on a non-root Android device on page 1-26.
• 1.6 Install the default profiling agent from the UI on page 1-28.
• 1.7 Standards compliance in Arm Streamline on page 1-29.
1.1 Features of Arm Streamline editions

Arm Streamline is available in Arm Development Studio and Arm Mobile Studio in a range of editions.

**Arm Development Studio**

The features that Arm Streamline supports in each Arm Development Studio edition are:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Bronze</th>
<th>Silver</th>
<th>Gold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bare-metal for M-class (including Armv8-M)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Bare-metal, Linux, and Android for Armv7-A and Armv7-R</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Bare-metal, Linux, and Android for Cortex®-A53</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Bare-metal, Linux, and Android for Armv8-A and Armv8-R</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Fastline for M-class</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Fastline for A/R-class</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Cycle models support</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Note**

You cannot open a report that was generated using a different toolkit license. The report must first be reanalyzed using the new license.

**Arm Mobile Studio**

The features that Arm Streamline supports in each Arm Mobile Studio edition are:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Starter</th>
<th>Silver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arm-based Android support (non-root)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Rooted system-wide profiling</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Streamline command-line interface</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Exporting captured data</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Bare-metal support</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Fast Model support</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Cycle models support</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Cortex-M profile support</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Cortex-R profile support</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Linux support</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
1.2 Overview of Arm Streamline Performance Analyzer

Arm Streamline Performance Analyzer samples the Program Counter (PC) address at regular intervals to generate a profile of where the processor spends most of the time. It also gathers data from Performance Monitoring Unit (PMU) counters on the target, for example data on memory or cache activity, and generates an analysis report.

Communicating with the target

Arm Streamline collects data by communicating with a software agent on the target. There are two agents that Arm Streamline can use, gator and barman. Use gator for Linux targets, or barman for bare-metal targets. See 2.2 Setup scenarios for Linux targets on page 2-32 and Chapter 13 Bare-Metal Support on page 13-227 for more information.

Performance counters

By default, Arm Streamline uses the best-fit set of hardware performance counters for your target. You can modify these counters in the Counter configuration dialog, which lists the counters available for your target hardware.

Arm Streamline collects samples for the counters in one of the following ways:

- At the fixed intervals of the sample rate. This option is the default sampling method.
- Event-based sampling, which samples on context switches, or when an event happens a set number of times.

See Chapter 5 Counter Configuration on page 5-85.

Annotations

To add context to the information that Arm Streamline gathers, you can add annotations to your code. For example, string annotations display text in the Details panel of the Timeline view. Bookmark annotations allow you to easily return to points of interest in the Timeline and Log views.

While Arm Streamline captures the data, your annotation output is recorded in the capture report. See Chapter 9 Code Annotations on page 9-181 for more information about the different types of annotations and how to use them.

Examples of captures and annotations

To quickly get an idea of the capabilities of Arm Streamline, Arm Development Studio includes some example captures that you can import and view. See 1.4 Importing the capture examples on page 1-25.

Practice using annotations with the examples in DS_install_directory/sw/streamline/examples/annotations and DS_install_directory/examples/Linux_examples/Streamline_annotate. See 9.1 Annotate overview on page 9-182 and 9.7 Importing the Streamline_annotate example on page 9-193 for more information about these examples.

Bare-metal support examples are in DS_install_directory/sw/streamline/examples/barman. See 13.1 Bare-metal support overview on page 13-228 for more information.

Note

Example captures are not included in Arm Mobile Studio.

1.2.1 Getting started

Before you can profile your application or system, your device must be appropriately configured.

By default, your device might already be configured, so you could try the steps that are given in the following table.
### Device or target type

<table>
<thead>
<tr>
<th>Device or target type</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android non-rooted</td>
<td>See 1.5 Profile an application on a non-root Android device on page 1-26.</td>
</tr>
<tr>
<td>Android or Linux rooted</td>
<td>See 1.6 Install the default profiling agent from the UI on page 1-28.</td>
</tr>
</tbody>
</table>

### 1.2.2 User interface

Use the user interface to configure your target, start and stop captures, and view results.

**Target view**

With this view you can start and stop new captures, configure performance counters, set capture and analysis options, and select targets. See 3.1 Target view on page 3-58.

**Streamline Data view**

With this view, you can select captures to analyze, merge captures, import captures from .apc directories, or perf recordings. See 3.3 Arm Streamline Data view overview on page 3-63.

Arm Streamline has the following views for displaying the capture data in different ways:

**Live view**

Opens when you start a capture. This view plots the counters in real time as Arm Streamline captures the data. The Timeline view replaces this view when you stop the capture. See Chapter 6 Live View and Timeline View on page 6-115.

**Timeline view**

Opens when a capture is complete. This view displays the capture information in charts and a details panel. You can customize the charts and create templates to quickly apply these customizations in future.

In the Details panel, you can select the mode in which detailed information is displayed. Some modes indicate the level of activity that threads and processes cause. Other modes list functions and processes that are active in the view. Use the **OpenCL** mode with Mali Midgard targets. If there are visual annotations in the capture, you can use the **Images** mode. See Chapter 6 Live View and Timeline View on page 6-115.

**Call Paths view**

Displays a hierarchy of functions, threads, and processes. You can use disclosure controls on each function, thread, and process to show more of the hierarchy. See Chapter 7 Table Views: Call Paths and Functions on page 7-162.

**Functions view**

Lists all the functions that your application called during a capture, alongside data about their usage. See Chapter 7 Table Views: Call Paths and Functions on page 7-162.

**Code view**

Displays statistics for lines of the source code and, optionally, the disassembled instructions. See Chapter 8 Code View on page 8-173.

**Log view**

Lists the annotations that the code generated during the capture session and information about them, including when and where the annotation was generated. See Chapter 10 Log View on page 10-194.

The following figure shows the Arm Streamline interface, with the **Timeline** view visible:
1 Introduction to Arm Streamline Performance Analyzer

1.2 Overview of Arm Streamline Performance Analyzer

Figure 1-1  Arm Streamline graphical user interface
1.3 New features and updates

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

### New features and updates in version 7.0

<table>
<thead>
<tr>
<th>Feature</th>
<th>Summary</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPE support.</td>
<td>Support for the Statistical Profiling Extension to the Armv8 architecture. This support includes new counter configuration options, and an option for changing the source for the Call Paths, Functions, and Code views.</td>
<td>See 5.11 Statistical Profiling Extension on page 5-112.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Update</th>
<th>Documentation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple encryption of credentials when downloading over SSH.</td>
<td></td>
<td>See 3.2.2 Setup Target dialog box on page 3-60 and 6.5 Image download on page 6-135.</td>
</tr>
<tr>
<td>Updated flags for call-stack unwinding.</td>
<td></td>
<td>2.13 Recommended compiler options on page 2-55.</td>
</tr>
<tr>
<td>mgd daemon is now aga-daemon.</td>
<td></td>
<td>See 6.7 Graphics Analyzer Mode in Live view on page 6-148.</td>
</tr>
<tr>
<td>Updated gatord command-line options.</td>
<td></td>
<td>See 2.11 gatord command-line options on page 2-49.</td>
</tr>
</tbody>
</table>

### New features and updates in version 6.9

<table>
<thead>
<tr>
<th>Update</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Different features of Arm Streamline are available in different Arm Development Studio and Arm Mobile Studio editions.</td>
<td>See 1.1 Features of Arm Streamline editions on page 1-17.</td>
</tr>
<tr>
<td>Restructured information about profiling on a non-root Android device.</td>
<td>See 1.5 Profile an application on a non-root Android device on page 1-26.</td>
</tr>
</tbody>
</table>

### New features and updates in version 6.8

<table>
<thead>
<tr>
<th>Feature</th>
<th>Summary</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import SoC Designer System Analyzer profiling data.</td>
<td>You can import .profile files, generated by SoC Designer System Analyzer, into Arm Streamline.</td>
<td>See 3.9 Importing SoC Designer profiling data on page 3-73.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Update</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updated for Arm Development Studio (Arm DS).</td>
<td>Throughout the document.</td>
</tr>
</tbody>
</table>

1 Introduction to Arm Streamline Performance Analyzer
1.3 New features and updates
## New features and updates in version 6.7

<table>
<thead>
<tr>
<th>Feature</th>
<th>Summary</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactive counters</td>
<td>Support for adding or removing event counters during capture.</td>
<td>See 3.1 Target view on page 3-58, 6.1.1 Interactive counters panel on page 6-117, 6.1.2 Adding and removing events in Live view on page 6-118.</td>
</tr>
<tr>
<td>Local capture mode</td>
<td>Support, in <code>gatord</code>, to save a capture file on the target.</td>
<td>See 2.11 gatord command-line options on page 2-49, 2.9 Capturing data locally on a target on page 2-47, 4.1 Capture &amp; Analysis Options dialog box settings on page 4-77, 2.8 Profiling applications without root access on page 2-46.</td>
</tr>
<tr>
<td>Heat map chart display</td>
<td>Added a heat map view option to charts.</td>
<td>See 6.4.1 Chart configuration panel on page 6-127, 6.4.2 Chart configuration toolbar options on page 6-127.</td>
</tr>
</tbody>
</table>

## Update

<table>
<thead>
<tr>
<th>Feature</th>
<th>Summary</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updated Streamline Data view</td>
<td>to add Target view tab and toolbars.</td>
<td>See Chapter 3 Target and Data Views on page 3-57, 3.2 Starting a capture session on page 3-59.</td>
</tr>
<tr>
<td>Updated description of Cross Section Marker range selection.</td>
<td></td>
<td>See 6.3.7 Cross Section Marker on page 6-124.</td>
</tr>
<tr>
<td>Updated <code>gatord</code> command-line options.</td>
<td></td>
<td>See 2.11 gatord command-line options on page 2-49.</td>
</tr>
<tr>
<td>Updated Python profiling support.</td>
<td></td>
<td>See 16.2 Python profiling support on page 16-284.</td>
</tr>
</tbody>
</table>

## New features and updates in version 6.6

<table>
<thead>
<tr>
<th>Feature</th>
<th>Summary</th>
<th>Documentation</th>
</tr>
</thead>
</table>

## New features and updates in version 6.5

<table>
<thead>
<tr>
<th>Feature</th>
<th>Summary</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Importing example captures</td>
<td>New wizard to simplify importing capture example.</td>
<td>See 1.4 Importing the capture examples on page 1-25.</td>
</tr>
</tbody>
</table>

## Update

<table>
<thead>
<tr>
<th>Feature</th>
<th>Summary</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updated Arm Streamline prerequisites.</td>
<td></td>
<td>See 2.1 Arm Streamline prerequisites on page 2-31.</td>
</tr>
<tr>
<td>Added description of separate image and debug files.</td>
<td></td>
<td>See 6.5 Image download on page 6-135, 6.5.1 Separate debug files on page 6-137.</td>
</tr>
</tbody>
</table>
## New features and updates in version 6.4

<table>
<thead>
<tr>
<th>Feature</th>
<th>Summary</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profiling with Embedded Trace Macrocell.</td>
<td>An extension to the bare-metal support that allows the collection of profiling data using ETM. You can import ETM traces from the Barman Generator Wizard in Arm Streamline, or from the command line.</td>
<td>See 13.5 Profiling with Embedded Trace Macrocell on page 13-241, 15.6 Importing a trace from the command-line on page 15-280.</td>
</tr>
<tr>
<td>Fastline Wizard.</td>
<td>Automates the process of setting up the Fastline plug-in, for profiling applications running on Fast Models.</td>
<td>See Chapter 14 Fastline on page 14-264.</td>
</tr>
<tr>
<td>barman_in_memory_helpers.py.</td>
<td>Arm Streamline outputs this use case script with the barman agent sources when you select a linear or circular RAM buffer. Use it in Arm Development Studio to help dump the contents of the in-memory capture buffer.</td>
<td>See 13.2.3 Barman use case script on page 13-233.</td>
</tr>
<tr>
<td>Profiling with Instruction Trace.</td>
<td>Import an instruction trace with the Import Trace Dump wizard for Arm Streamline to analyze.</td>
<td>See 13.6 Profiling with Instruction Trace on page 13-245.</td>
</tr>
</tbody>
</table>

## Update

<table>
<thead>
<tr>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction updated.</td>
</tr>
<tr>
<td>Information about Mali Midgard and Mali Bifrost counters added.</td>
</tr>
<tr>
<td>Mali counters are available with user space gator, text saying they have not been removed.</td>
</tr>
<tr>
<td>More detail added to barman_sample_counters and barman_sample_counters_with_program_counter.</td>
</tr>
</tbody>
</table>

## New features and updates in version 6.3

<table>
<thead>
<tr>
<th>Feature</th>
<th>Summary</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profiling with Instrumentation Trace Macrocell.</td>
<td>An extension to the bare-metal support that allows the collection of profiling data using ITM. You can import ITM traces from the Barman Generator Wizard in Streamline, or from the command line.</td>
<td>See 13.4 Profiling with Instrumentation Trace Macrocell on page 13-238, 15.6 Importing a trace from the command-line on page 15-280.</td>
</tr>
</tbody>
</table>

## Update

<table>
<thead>
<tr>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>The paths to MGD and ADB are now set in the dialog at Window &gt; Preferences &gt; Streamline &gt; External Tools, not the Capture &amp; Analysis Options dialog.</td>
</tr>
<tr>
<td>The Program Images section of the Capture &amp; Analysis Options dialog has changed.</td>
</tr>
<tr>
<td>The Preferences dialog has been moved and expanded.</td>
</tr>
<tr>
<td>barman_initialize varies depending on the datastore chosen.</td>
</tr>
</tbody>
</table>
## New features and updates in version 6.2

<table>
<thead>
<tr>
<th>Feature</th>
<th>Summary</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Images download.</td>
<td>Download process images from the target device at the end of a live capture session. Use the <strong>Image Download</strong> dialog to select the images to download.</td>
<td>See <strong>6.5 Image download</strong> on page 6-135.</td>
</tr>
<tr>
<td>Profiling with System Trace Macrocell.</td>
<td>An extension to the bare-metal support that allows the collection of profiling data using STM. You can import STM traces from the <strong>Barman Generator Wizard</strong> in Arm Streamline, or from the command line.</td>
<td>See <strong>13.3 Profiling with System Trace Macrocell</strong> on page 13-235, <strong>15.6 Importing a trace from the command-line</strong> on page 15-280.</td>
</tr>
<tr>
<td>Non-root support.</td>
<td>Run <strong>gatord</strong> on a device without root permissions. For Android targets, an Android daemon application is supplied.</td>
<td>See <strong>2.2.1 Comparison of user space gator and kernel space gator</strong> on page 2-32.</td>
</tr>
<tr>
<td>Export Heat Map.</td>
<td>Export the data contained in the <strong>Heat Map</strong> from the GUI or the command line.</td>
<td>See <strong>6.6.2 Heat Map mode</strong> on page 6-138, <strong>15.4 Exporting the Heat Map from the command-line</strong> on page 15-278.</td>
</tr>
</tbody>
</table>

### Update

<table>
<thead>
<tr>
<th>Update</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bare Metal</strong> menu renamed to <strong>Streamline</strong>. <strong>Generate Agent Sources</strong> dialog renamed to <strong>Barman Generator Wizard</strong>.</td>
<td>Screenshots updated to show this.</td>
</tr>
<tr>
<td>The <strong>Counter Configuration</strong> dialog has been improved.</td>
<td>See <strong>Chapter 5 Counter Configuration</strong> on page 5-85.</td>
</tr>
<tr>
<td>Command-line options -pmus and -events added to documentation.</td>
<td>See <strong>15.2 Arm Streamline command-line options</strong> on page 15-273.</td>
</tr>
</tbody>
</table>
1.4 Importing the capture examples

You can import examples of Arm Streamline captures.

On first start-up, Arm Streamline prompts you to import the examples. If you choose not to import them, you can import them later.

**Importing the examples from the file menu**

1. Select **File > Import...**
2. In the **Import** dialog box, select the **Import Streamline Sample Captures** wizard.
3. Select **Next**.
4. In the **Import Sample Captures** dialog box, choose the examples that you require.
5. Select **Finish**.

The examples that you have selected are then imported.
1.5 Profile an application on a non-root Android device

To profile a single Android application, you can connect Arm Streamline to your Android device through a USB, ethernet, or WiFi connection.

Before profiling your Android application for the first time, use adb to install the Arm Streamline gator daemon, `gatord`, on your Android device.

**Prerequisites**

- The *Android Debug Bridge* (adb) must be installed and running on your host.
  
  The adb is a utility that is part of the Android SDK. For information about the Android SDK and adb setup, see the Android website: [http://developer.android.com/tools/help/adb.html](http://developer.android.com/tools/help/adb.html).
- Your Android application must be debuggable.
  
  Before you build your application, specify `android:debuggable="true"` in the `AndroidManifest.xml` file.
  
  This is required because you must use the adb `run-as` command.

**Procedure**

1. Forward the Arm Streamline port to your localhost.
   
   `adb forward tcp:8080 tcp:8080`

2. Configure Arm Streamline to automatically connect to an Android device (optional).
   
   To avoid the need to manually set up port forwarding, you can specify the location of the adb executable. The next time that you connect to your device, you can select it in the **Connection Browser** dialog.
   
   a. Select **Window > Preferences > Streamline > External Tools**.
   b. Enter the path in the **ADB Path** text box.
   c. Click **Ok**.

3. Install the Arm Streamline gator daemon, `gatord`, on your Android device.
   
   a. Copy `gatord`, to the device.
   
   ```
   host$ adb push gatord /data/local/tmp
   ```
   
   - For Armv7-A architecture, copy `DS_install_directory/sw/streamline/bin/arm/gatord`.
   - For Armv8-A architecture, copy `DS_install_directory/sw/streamline/bin/arm64/gatord`.
   b. Start a shell on the device.
   
   ```
   host$ adb shell
   ```
   c. Copy `gatord`, to the application's directory.
   
   ```
   target$ run-as package_name cp /data/local/tmp/gatord gatord
   ```

4. Capture data on the Android device.
   
   a. Enable profiling.
   
   ```
   host$ adb shell setprop security.perf_harden 0
   ```
   b. Run `gatord` and wait for the application to launch (from the Android shell).
   
   ```
   target$ run-as package_name ./gatord --wait-process package_name
   ```
   
   To profile a Mali GPU, add the following command-line options:
   
   ```
   target$ run-as package_name ./gatord --wait-process package_name -M mali_name
   ```
   
   Where `mali_name` is the GPU name without the Mali prefix. For example, for Mali-G72 the name is G72.
   c. Configure and start a capture in the Arm Streamline UI.
   
   The UI waits for the application to launch.
   d. Start the application from the Android UI, or from the command line.
host$ adb shell am start package_name

Arm Streamline is now capturing the performance profile data of the application.

Related references
17.3 Troubleshooting Android issues on page 17-292
3.2.1 Connection Browser dialog box on page 3-59
2.11 gatord command-line options on page 2-49
1.6 Install the default profiling agent from the UI

To capture system-wide profile data in a rooted target, you must install the user space profiling agent, \texttt{gatord}, on your target.

If your device is appropriately configured, you can easily install \texttt{gatord} on your target using this procedure.

**Procedure**

1. Click the **Browse for a target** button \faFolderOpen in the **Target view**, to open the **Connection Browser** dialog.
2. Click **Setup Target...** to launch the **Setup Target** dialog.
3. Select your target type.
4. Fill in the parameters appropriate to your device.
5. Click **Install**.

- If your device is appropriately configured, Arm Streamline has installed the user space profiling agent, \texttt{gatord}, on your target.
- If Arm Streamline showed an error message, see *Chapter 2 Target Setup on page 2-30* for instructions on how to validate and change the configuration of your target.
1.7 Standards compliance in Arm Streamline

Lists the levels of compliance that Arm Streamline conforms to.

**ELF**

Arm Streamline can read executable images in ELF format.

--- Note ---

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

---

**DWARF**

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

--- Note ---

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

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

It contains the following sections:

- 2.1 Arm Streamline prerequisites on page 2-31.
- 2.2 Setup scenarios for Linux targets on page 2-32.
- 2.3 Preparing and building your Linux kernel on page 2-36.
- 2.4 Required kernel configuration menu options on page 2-38.
- 2.5 Building and installing the gator daemon on page 2-40.
- 2.6 Building the gator module on page 2-42.
- 2.7 Running the gator daemon on your target on page 2-44.
- 2.8 Profiling applications without root access on page 2-46.
- 2.9 Capturing data locally on a target on page 2-47.
- 2.10 Stopping the gator daemon on page 2-48.
- 2.11 gatord command-line options on page 2-49.
- 2.12 Support for other hardware on page 2-52.
- 2.13 Recommended compiler options on page 2-55.
2.1 **Arm Streamline prerequisites**

To profile your software using Arm Streamline, you must have a suitably configured Linux kernel running on an Arm-based hardware target.

**Hardware**

A hardware target with an Armv7-A or Armv8-A architecture core. Arm Streamline is supported on all Armv7-A and Armv8-A cores, and is tested on ArmCortex-A cores.

**Software**

Before you can capture data on an Arm Linux target using Arm Streamline, you must check that a target agent (gator) is running. gator can be run in kernel space or user space mode. See [2.2.1 Comparison of user space gator and kernel space gator](#) on page 2-32.

**Kernel configuration**

The kernel configuration must include the options that are described in: [2.4 Required kernel configuration menu options](#) on page 2-38. Your kernel configuration usually can be found in `/proc/config.gz`. If this file is not visible, depending how your system is configured, you can create it by running:

```
sudo modprobe configs
```

Some profiling with Arm Streamline is possible without the kernel source code:

- If `gatord` and `gator.ko` are already included in your system, kernel source code is not required.
- For Linux kernel versions 3.4 and later, user space gator can be used for profiling.

**Compiler**

If kernel source code is available for your system, you can use GCC to build the Linux kernel or `gator.ko`.

You can run the compilation either natively, on the target, or cross-compile it on a Linux host.

You can obtain GCC at the Linaro website: [http://www.linaro.org](http://www.linaro.org), or, depending on the GCC version you require, from the Arm Development Studio installation folder.

Arm Streamline currently supports gator versions 17-25.

- gator versions 17-21 support Linux kernel versions 2.6.32 and later
- gator versions 22 and later only support Linux kernel versions 3.4 and later.

**Related tasks**

- [2.3 Preparing and building your Linux kernel](#) on page 2-36
- [2.6 Building the gator module](#) on page 2-42
- [2.7 Running the gator daemon on your target](#) on page 2-44
- [3.2 Starting a capture session](#) on page 3-59

**Related references**

- [2.4 Required kernel configuration menu options](#) on page 2-38
2.2 Setup scenarios for Linux targets

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

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

2.2.1 Comparison of user space gator and kernel space gator

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

gator consists of the following components:
• A daemon, `gatord`.
• Optionally, a Linux kernel driver module, `gator.ko`.

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

`gatord` must be installed and running on the target to allow Arm Streamline to communicate with the target. However, `gatord` can run with or without `gator.ko`.

gator can operate in the following modes:

**Kernel space gator**

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

**User space gator**

If `gator.ko` is not installed on your device, `gatord` operates in user space mode using one of the following methods:

**Using Linux perf API**

Where available, `gatord` uses the perf API to collect performance information. User space gator is supported on Linux kernel versions 3.4 and later. You can run user space gator on a device with or without root permissions. When you run `gatord` on a device with root permissions, this is called *root user space gator*. When this document uses the term *user space gator*, it is referring to *root user space gator*.

Root user space gator supports system-wide data collection, including profiling all applications and the kernel. When run as non root user, `gatord` is limited to capturing only specific processes that you have permission to profile (usually where the process is from the same user).

**Polling the `/proc` filesystem**

If the perf API is not installed, `gatord` uses the `/proc` filesystem to collect statistics. However, the following limitations apply:
• The available counters are limited to the basic process and system statistics. For example, **CPU Activity** and **Memory**.
• Counters are collected at a lower frequency.
• **CPU Activity** and **Heat Map** display approximate data.
• No profiling information is available, which means the **Call Paths** and **Functions** views are empty.

User space gator is restricted to using user space APIs and does not support the following Linux counters that kernel space gator supports:
• CPU I/O: Wait.
• Idle: State.

User space gator has some other restrictions, for example:
• It polls the following Linux counters every 100ms, instead of every 1ms or when they change because files in the /proc or /sys filesystem are read:
  — Memory.
  — Disk I/O.
  — Network.

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

Related concepts
6.6.7 Processes mode on page 6-143

2.2.2 Validating the target setup

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

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

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

• Check that the gatord process is running on the target. If not, Arm Streamline reports an error when you try to start a capture.
• The version of gator running on the target must be compatible with the version of Arm Streamline you are using. You can use the following command to print the version number of gatord:

```bash
./gatord -v
```

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

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

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

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

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

```bash
lsmod | grep gator
```

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

Related concepts
2.2.3 Building and installing user space gator on page 2-34
2.2.4 Building and installing kernel space gator on page 2-34

Related tasks
2.3 Preparing and building your Linux kernel on page 2-36

Related references
2.4 Required kernel configuration menu options on page 2-38
3.2.1 Connection Browser dialog box on page 3-59
2.2.3 Building and installing user space gator

To use user space gator, the gator daemon, `gatord`, must be installed and running on the target.

Note

If you do not have root permissions for your Android device, see 1.5 Profile an application on a non-root Android device on page 1-26.

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

Two pre-built `gatord` binaries are available:

- For Armv7 targets, and Armv8 targets that support AArch32 execution state.
- For Armv8 AArch64 targets.

You cannot install them on a target that is based on another Arm architecture using the Setup Target... dialog. If you want to do this, or if the supplied `gatord` binary does not work correctly on your system, you must build, install, and run it manually.

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

- `DS_install_directory/sw/streamline/gator/daemon/`
- `https://github.com/ARM-software/gator`. This site is the official distribution channel for all gator releases, and contains the latest source updates between Arm Development Studio releases.

To build `gatord`, follow the instructions in 2.5.2 Building the gator daemon yourself on page 2-40. To install and run it on the target, see 2.7 Running the gator daemon on your target on page 2-44.

If an existing version of `gatord` is running, you must first kill it, see 2.10 Stopping the gator daemon on page 2-48.

Related tasks

2.5.2 Building the gator daemon yourself on page 2-40
2.7 Running the gator daemon on your target on page 2-44
2.10 Stopping the gator daemon on page 2-48

Related references

3.2.1 Connection Browser dialog box on page 3-59

2.2.4 Building and installing kernel space gator

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

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

- `DS_install_directory/sw/streamline/gator/driver/`
- `https://github.com/ARM-software/gator`

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

- Integrate `gator.ko` into the Linux kernel build system. This involves the following steps:
  1. Copy the gator driver source into the kernel source tree, for example 
     `<path_to_kernel_build_dir>/drivers/gator/`.
  2. Add references to gator in the files `Makefile` and `Kconfig`, located in the parent directory, for example `<path_to_kernel_build_dir>/drivers/`.
  3. Launch `menuconfig` and select `GATOR` to configure the required kernel configuration options.
  4. Rebuild the kernel.

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

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

**Related tasks**

- 2.6 Building the gator module on page 2-42
- 2.10 Stopping the gator daemon on page 2-48
2.3 Preparing and building your Linux kernel

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

The options that you need to enable are described in the topic 2.4 Required kernel configuration menu options on page 2-38.

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

**Procedure**

1. Download one of the supported versions of the Linux kernel and configure it. See the topic 2.1 Arm Streamline prerequisites on page 2-31 for the list of supported versions.

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

   __Note__

   You can build and configure the gator driver `gator.ko` by copying the gator driver source directly into the Linux kernel source tree. On Linux, this code is located in the directory `DS_install_directory/sw/streamline/gator/driver/`. For more information, see the topic 2.6 Building the gator module on page 2-42.

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

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

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

   ```bash
   export ARCH=arm
   ```

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

   ```bash
   make platform_defconfig
   ```

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

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

   ```bash
   make menuconfig
   ```

6. Set the required kernel configuration menu options.

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

   ```bash
   make -j5 uImage
   ```

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

   ```bash
   make -j5 dtbs uImage
   ```

   __Note__

   If a device tree is used, it must include the pmu bindings. See `Documentation/devicetree/bindings/arm/pmu.txt` in the Linux kernel sources for details.

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

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

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

**Related tasks**

2.6 Building the gator module on page 2-42
Related references

2.4 Required kernel configuration menu options on page 2-38
2.1 Arm Streamline prerequisites on page 2-31
2.4 Required kernel configuration menu options

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

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

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

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

--- Note ---
Use the hrtimer_module to validate the timer on the target device operates within the expected parameters. The module produces logging information in dmesg output to indicate whether the timer is operating correctly.

From the module directory, enter the following command:

```
make -C <path_to_kernel_source> M="pwd" modules
```

This command produces a .ko file.

Run the module with the command `insmod hrtimer_module.ko`.

Stop the module with the command `rmmod hrtimer_module.ko`.

---

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

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

Device Drivers
Select Graphics support > ARM GPU Configuration > Mali Midgard series support > Streamline Debug support and enable CONFIG_MALI_GATOR_SUPPORT for Mali Midgard support.
Kernel hacking

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

**Related tasks**

2.12.3 Setting up Arm Streamline to support an Arm® Mali™-based device on page 2-52
2.5 Building and installing the gator daemon

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

Arm Streamline includes a pre-built gatord binary, which works in most use cases. However, if the pre-built gator daemon does not work, you can build gatord yourself.

2.5.1 The pre-built gator daemon

Install and run the pre-built gator daemon on a Linux or Android target by clicking Setup target... in the Connection Browser dialog.

To improve portability, gatord is statically compiled against musl libc from http://www.musl-libc.org/download.html instead of glibc. The gator daemon works correctly with either glibc or musl.

**Hardfloat EABI**

Binary applications that are built for the soft or softfp ABI are not compatible on a hardfloat system. All soft or softfp applications must be rebuilt for hardfloat. To check if your Arm compiler supports hardfloat, run gcc -v and look for --with-float=hard.

To compile for non-hardfloat targets, it is necessary to add the following options:

```
-marm -march=armv4t -mfloat-abi=soft
```

It might also be necessary to provide a softfloat filesystem by adding the option --sysroot, for example:

```
--sysroot=../Examples/distribution/filesystem/armv5t_mtx
```

The gatord makefile provides a softfloat filesystem when run as follows:

```
make SOFTFLOAT=1 SYSROOT=<path_to_sysroot>
```

The armv5t_mtx filesystem is provided as part of the Arm Development Studio Linux Example Distribution package, which you can download from the DS-5 Downloads page.

Attempting to run an incompatible binary often results in the error message No such file or directory when the file does exist.

2.5.2 Building the gator daemon yourself

To build the gator daemon, follow the steps in this topic.

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

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

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

**Note**

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

To build gatord, follow these steps:

**Procedure**

1. Either download the gatord source from https://github.com/ARM-software/gator, or copy the source that is supplied in DS_install_directory/sw/streamline/gator/daemon/.

2. Change to the gator daemon directory by using either of the following commands:
• For Linux, enter:
  cd daemon
• For Android, enter:
  mv daemon jni

3. Issue the commands to build gatord.
  • To build gatord for an Armv7 Linux target, enter:
    make CROSS_COMPILE=<cross_compiler_directory>/bin/arm-linux-gnueabihf-
  • To build gatord for an Armv8 Linux target, enter:
    make -f Makefile CROSS_COMPILE=<cross_compiler_directory>/bin/aarch64-linux-
    gnu-
  • To build gatord for Android, enter:
    <NDK_install_directory>/ndk-build
    gatord is now located in libs/armeabi.
    — Note ——
    To build gatord for AArch64 or Arm11™, edit jni/Application.mk and replace arm64-v8a
    with the following:
    — arm64-v8a for AArch64.
    — armeabi for Arm11.

4. If you did not build gatord on the target, transfer it to the target and then move it to the appropriate
   directory. Which directory is appropriate depends on the target. Root should have write permission
   for this directory.
5. Make gatord executable by entering the following command:
   chmod +x gatord

Related tasks
2.7 Running the gator daemon on your target on page 2-44
3.2 Starting a capture session on page 3-59

Related references
3.2.1 Connection Browser dialog box on page 3-59
17.1 Troubleshooting target setup issues on page 17-289
2.6 Building the gator module

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

If you do not build the gator.ko driver, Arm Streamline uses user space gator, which provides a subset of the functionality that kernel space gator provides.

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

Procedure

1. Either download the gator.ko source from https://github.com/ARM-software/gator, or copy the source supplied in DS_install_directory/sw/streamline/gator/driver/.
2. Change to the gator driver directory:
   cd driver
3. Assuming that you have all of the required tools for building kernel modules, you can build gator.ko either by using a build command or by integrating it into the kernel build system. The build command to use depends on whether you are building on a Linux host or a target.

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

- On a Linux host, use the following build command:
  
  ```
  make -C <kernel_build_dir> M=`pwd` ARCH=arm
  CROSS_COMPILE=<cross_compiler_directory>/bin/arm-linux-gnueabihf- modules
  ```

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

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

- On a target, use the following build command:
  ```
  make -C <kernel_build_dir> M=`pwd`
  ```

- To integrate gator.ko into the kernel build system, use the following instructions:
  ```
  cd <kernel_build_dir>
  cd drivers
  mkdir gator
  cp -r <path_to_gator_driver_src>/* gator
  ```

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

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

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

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

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

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

Related tasks

| 2.12.3 Setting up Arm Streamline to support an Arm®Mali™-based device on page 2-52 |
| 2.12.1 Getting L2C-310 memory-mapped peripherals working with Arm Streamline on page 2-52 |
| 12.9 Profiling the Linux kernel on page 12-222 |
| 2.3 Preparing and building your Linux kernel on page 2-36 |
3.2 Starting a capture session on page 3-59

Related references

2.1 Arm Streamline prerequisites on page 2-31
2.7 **Running the gator daemon on your target**

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

--- **Note** ---

- This setup task applies to both Linux and rooted Android targets.
- You can install and run user space gator on the target automatically by clicking the **Setup target**... button in the **Connection Browser** dialog.

To run gatord manually, follow these steps:

**Procedure**

1. Copy gatord and, optionally, gator.ko into the file system on the target. On Android, you can do this on your host using `adb push`.
2. To ensure gatord has execute permission, enter the following command:
   ```bash
   chmod +x gatord
   ```
   On Android, you can execute this command using:
   ```bash
   adb shell
   ```
3. Optional: If gatord is in a different directory to gator.ko on the target, you must do one of the following:
   - Insert the gator.ko module manually using the following command:
     ```bash
     insmod gator.ko
     ```
   --- **Note** ---
   If gatord does not load gator.ko, you must reload gator.ko between runs of gatord using the following commands:
   ```bash
   umount /dev/gator; rmmod gator; insmod gator.ko
   ```
   - Include the path to gator.ko using the `-m` option to gatord. For example:
     ```bash
     ./gatord -m /home/gator/gator.ko &
     ```
     If gatord does not exist in the same directory as gator.ko and you do not either manually insert gator.ko or include a path to it when running gatord, a user space only version of gatord runs when the command is executed. This user-space version of gatord has most, but not all of the functionality of the standard version of gatord running with the gator.ko module. If `/dev/gator/` version does not exist after starting gatord, then you are running user-space gator.
4. Ensure that you have root privileges, if necessary, then enter the following to execute the gator daemon:
   ```bash
   ./gatord &
   ```
   By default, gatord uses port 8080 for communication with the host, but you can specify the port by launching gatord with the port number as a parameter. For example:
   ```bash
   ./gatord -p 5050 &
   ```
   Additionally, specify the port number using the **Capture & Analysis Options** dialog box by appending a colon followed by the port number to the IP address in the address field. For example,
the address is 10.99.28.54 and the port is 5050 you enter 10.99.28.54:5050. If you do not provide a port number, the default port is used.

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

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

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

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

```
setenforce 0
```

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

```
setenforce 1
```

**Related tasks**

- 3.2 Starting a capture session on page 3-59
- 2.10 Stopping the gator daemon on page 2-48

**Related references**

- 3.2.1 Connection Browser dialog box on page 3-59
- 2.11 gatord command-line options on page 2-49
2.8 Profiling applications without root access

You can profile applications either using the UI or the command-line interface. The different procedures are summarized here:

**Capturing a trace locally to filesystem**
See 2.9 Capturing data locally on a target on page 2-47.

**Trace an application specified by the UI.**
See Chapter 4 Capture and Analysis Options on page 4-76.

**Trace a single application.**

```
gatord --system-wide no -a
```

**Trace an application specified on the command line.**

```
gatord [--system-wide no] --app <command> [<arguments...>]
```

When run, *gatord* waits for the UI to launch a capture, and then launches the application, tracing it in the same way as if you had specified the command in the *Capture & Analysis Options* dialog box.

The `-system-wide no` is optional, but if you need to capture system-wide capture and specify the application on the command line, you must specify `-system-wide yes`.

**Trace processes that are already running.**

```
gatord --pid <pids...>
gatord
```

gatord profiles only the given set of processes (and any newly forked processes or threads).

**Wait for an externally launched process to run.**

```
gatord --wait-process <command>
```

<command> must be the path to the executable, the executable name, or on Android, the application id. This mode is primarily useful for Android users where it is not possible to launch the application directly, so you must launch your application from the UI separately. When run like this, *gatord* waits for the process matching <command> to launch before starting capturing.

**Related references**

- 4.1 Capture & Analysis Options dialog box settings on page 4-77
- 17.3 Troubleshooting Android issues on page 17-292
- 2.11 gatord command-line options on page 2-49
- 2.13 Recommended compiler options on page 2-55
2.9 Capturing data locally on a target

You can run gatord from the command line to configure the counters and capture data locally on your target.

To perform a local capture, start gatord with the -o or --output option.

To specify which counters to use, you can either pass the -c <xml> option, or specify one or more -C <specifier> instead. The -C option takes a comma-separated list of counter specifiers, similar to counters specified in the configuration.xml file. The -C option can be specified multiple times.

Most hardware counters are identified by an event code and a slot. For example, on Cortex-A53 there are six slots corresponding to the six PMU event counters. To specify the counter for a particular slot, pass -C <device>_cnt<s>:<e> where <device> is the prefix that identifies the device type, <s> is the slot number, and <e> is the event code. For example, the Instructions Executed counter would be configured in slot 0 as:

```
-C ARMv8_Cortex_A53_cnt0:0x08
```

To configure the cycle counter, specify -C <device>_ccnt. For example:

```
-C ARMv8_Cortex_A53_ccnt
```

Other counters do not have event codes and are identified only by name. For example:

```
-C PERF_COUNT_SW_PAGE_FAULTS
```

A range of other options are available to configure system-wide or application only tracing. See 2.11 gatord command-line options on page 2-49 for more details.

Related tasks

12.1 Creating a configuration.xml file on page 12-212
2.10 Stopping the gator daemon

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

To stop the gator daemon, follow these steps:

Procedure
1. Determine the process id of gatord using the following command:
   
   ps ax | grep gatord

2. Kill the identified process using the following command:
   
   kill process_id

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

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

   lsmod | grep gator

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

   umount /dev/gator; rmmod gator

You must be logged in as root to do this.
### 2.11 gatord command-line options

gatord must be running before you can capture trace data. The command-line options configure how gatord captures events and how it communicates with Arm Streamline running on your host.

Gatord has two modes of operation:

**Daemon mode (the default mode)**

Sends captured events to a host running Arm Streamline.

**Local capture mode**

Writes the capture to a file then exits.

To enable this mode, specify an output directory with the `--output` flag.

Arguments available to all modes:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Lists all of the available gatord command-line options.</td>
</tr>
<tr>
<td>-c, --config-xml &lt;config_xml&gt;</td>
<td>Specify the path and filename of the configuration.xml file that defines the capture options. In daemon mode, the list of counters is written to this file. In local capture mode, the list of counters is read from this file.</td>
</tr>
<tr>
<td>-e, --events-xml &lt;events_xml&gt;</td>
<td>Specify the path and filename of the events.xml file. events.xml defines all of the counters that Arm Streamline collects during the capture session.</td>
</tr>
<tr>
<td>-E, --append-events-xml &lt;events_xml&gt;</td>
<td>Specify the path and filename of events.xml to append.</td>
</tr>
<tr>
<td>-P, --pmus-xml &lt;pmu_xml&gt;</td>
<td>Specify path and filename of pmu.xml to append.</td>
</tr>
<tr>
<td>-m, --module-path &lt;module&gt;</td>
<td>Specify path and filename of gator.ko.</td>
</tr>
<tr>
<td>-v, --version</td>
<td>Print version information.</td>
</tr>
<tr>
<td>-d, --debug</td>
<td>Enable debug messages.</td>
</tr>
<tr>
<td>-A, --app &lt;cmd&gt; &lt;args...&gt;</td>
<td>Specify the command to execute when the capture starts. This argument must be the last argument that is passed to gatord. All subsequent arguments are passed to the launched application.</td>
</tr>
<tr>
<td>-S, --system-wide &lt;yes</td>
<td>no&gt;</td>
</tr>
<tr>
<td>-u, --call-stack-unwinding &lt;yes</td>
<td>no&gt;</td>
</tr>
<tr>
<td>-r, --sample-rate &lt;low</td>
<td>normal&gt;</td>
</tr>
<tr>
<td>-t, --max-duration &lt;s&gt;</td>
<td>Specify the maximum duration that the capture can run for in seconds. Defaults to 0, meaning unlimited.</td>
</tr>
<tr>
<td>-f, --use-efficient-ftrace &lt;yes</td>
<td>no&gt;</td>
</tr>
<tr>
<td>-w, --app-cwd &lt;path&gt;</td>
<td>Specify the working directory for the application that gatord launches. Defaults to the current directory.</td>
</tr>
<tr>
<td>-x, --stop-on-exit &lt;yes</td>
<td>no&gt;</td>
</tr>
<tr>
<td>-Q, --wait-process &lt;command&gt;</td>
<td>Wait for a process matching the specified command to launch before starting capture. Attach to the specified process and profile it.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| -M, --mali-type <types>      | Specify a comma-separated list that defines the Arm Mali GPUs that are present on the system. Use this option when it is not possible to auto-detect the GPUs. For example, because access to /sys is denied. The specified value must be one of the following:  
  - The product name. For example, Mali-T880 or G72.  
  - The 16-bit hex GPU ID value. If all devices are the same type, you can specify one type. Otherwise, you must specify a list that matches the order of --mali-device. |
| -D, --mali-device <paths>    | Specify the paths to the Arm Mali GPU device nodes in the /dev filesystem. When this parameter is not supplied, or not auto-detected, it defaults to /dev/mali0,/dev/mali1,… for the number of GPUs that are present. This option is only valid if --mali-type is also specified. |
| -Z, --mmmap-pages <n>        | The maximum number of pages to map per mmaped perf buffer is equal to <n+1>. n must be a power of two. |

Arguments available in daemon mode only:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| -p, --port <port_number>  | Set the port number that gatord uses to communicate with the host. The default is 8080.  
  If you use the argument uds, the TCP socket is disabled and an abstract Unix domain socket is created. This socket is named streamline-data. If you use Android, creating a Unix domain socket is useful because gatord is prevented from creating a TCP server socket.  
  Alternatively, you can connect to localhost:<local_port> in Arm Streamline using:  
  `adb forward tcp:<local_port> localabstract:streamline-data` |
| -a, --allow-command       | Allows you to run a command on the target during profiling. The command is specified in the Capture & Analysis Options dialog.  
  **Caution**  
  If you use this option, an unauthenticated user could run arbitrary commands on the target using Arm Streamline. |

Arguments available to local capture mode only:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-s, --session-xml &lt;session_xml&gt;</td>
<td>Specify the session.xml file that the configuration is taken from. Any additional arguments override values that are specified in this file.</td>
</tr>
<tr>
<td>-o, --output &lt;apc_dir&gt;</td>
<td>Specifies the path and filename of the output file (.apc) for a local capture.</td>
</tr>
<tr>
<td>-i, --pid &lt;pids...&gt;</td>
<td>A comma-separated list of process IDs to profile</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>-C, --counters &lt;counters&gt;</code></td>
<td>A comma-separated list of counters to enable. This option can be specified multiple times.</td>
</tr>
</tbody>
</table>
| `-X, --spe <id>[[:events=<indexes>][:ops=<types>][:min_latency=<lat>]]` | Enable Statistical Profiling Extension (SPE). Where:  
  - `<id>` is the name of the SPE properties that are specified in the `events.xml` or `pmus.xml` file. It uniquely identifies the available events and counters for the SPE hardware.  
  - `<indexes>` is a comma-separated list of event indexes to filter the sampling by. A sample will only be recorded if all events are present.  
  - `<types>` is a comma-separated list of operation types to filter the sampling by. A sample will be recorded if it is any of the types in `<types>`. Valid types are `LD` for load, `ST` for store and `B` for branch.  
  - `<lat>` is the minimum latency. A sample will only be recorded if its latency is greater than or equal to this value. The valid range is [0,4096). |

### Argument usage examples

Using `--pmus-xml` and `--append-events-xml` to add support for a new PMU without having to rebuild `gatord`.

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

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

The `events.xml` file must include the XML header and elements that are shown in the following example:

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

### Related references

4.1 Capture & Analysis Options dialog box settings on page 4-77
2.12 Support for other hardware

Arm Streamline provides support for the CoreLink™ Level 2 Cache Controller (L2C-310), the CoreLink Cache Coherent Network (CCN), and Mali-based devices.

2.12.1 Getting L2C-310 memory-mapped peripherals working with Arm Streamline

The gator driver source provides a file, `gator_events_12c-310.c`, which contains hard coded offsets for the locations of the L2 cache counter registers. You can configure the offset for your board by specifying a module parameter when loading `gator.ko`.

For example:

```
insmod gator.ko l2c310_addr=<offset>
```

You can disable the l2c-310 counter by providing an offset of zero, for example:

```
insmod gator.ko l2c310_addr=0
```

Related tasks

2.6 Building the gator module on page 2-42

2.12.2 CCN

CCN requires a perf driver to work.

The necessary perf driver is merged into Linux 3.17 but can be backported to previous versions. See [https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/diff/?id=a33b0daab73a0e08cc04459dd44b0121a8e8f81b](https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/diff/?id=a33b0daab73a0e08cc04459dd44b0121a8e8f81b) and later bug fixes.

2.12.3 Setting up Arm Streamline to support an Arm® Mali™-based device

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

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

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

- A supported Mali-based device.
- A sufficiently recent version of the Mali driver. Consult your supplier to see if this version of the driver is available for your device. For more information, see the Mali pages on Arm Developer, [https://developer.arm.com/graphics/](https://developer.arm.com/graphics/).

--- Note ---

- For recent Mali GPUs, the kernel module, `gator.ko` is not required, therefore the modifications described below are not needed. The GPU type is detected out of the box.
- If the specific type is not detected, you can specify the GPU type and, optionally, the GPU device on the command line using the `--mali-type` and `--mali-device` arguments.

---

Follow the normal installation and setup instructions for Arm Streamline and `gatord`. If you are using kernel space gator, see the topic 2.6 Building the gator module on page 2-42 for information on how to build `gator.ko`.

The following instructions describe the additional configuration options that `gator.ko` requires to support Mali-based devices. They assume that you are building `gator.ko` out-of-tree.
Procedure

1. Arm Streamline only supports one type of GPU (and driver version) at a time, and you choose this GPU type at build time.
   • For a Mali-4xx GPU, specify the following options to your gator.ko make command:
     ```
     GATOR_WITH_MALI_SUPPORT=MALI_4xx # Set by CONFIG_GATOR_MALI_4XXMP
     # gator source needs to #include "linux/mali_linux_trace.h"
     CONFIG_GATOR_MALI_PATH=".../<path_to_Mali_DDK_kernel_files>/src/devicedrv/mali"
     GATOR_MALI_INTERFACE_STYLE=<3|4> # 3=Mali-400 DDK >= r3p0-04rel0 and < r3p2-01rel3
     # 4=Mali-400 DDK >= r3p2-01rel3
     # A default of 4 is set in
     # streamline/gator/driver/gator_events_mali_4xx.c
     ```
     To add the corresponding support to the Mali drivers, user space needs the following options:
     — MALI_TIMELINE_PROFILING_ENABLED=1
     — MALI_FRAMEBUFFER_DUMP_ENABLED=1
     — MALI_SW_COUNTERS_ENABLED=1
     Kernel space needs USING_PROFILING=1 # Sets CONFIG_MALI400_PROFILING=y
     These settings are the defaults in later driver versions. See the documentation that is supplied with the Arm Driver Development Kit (DDK) for Mali-4xx GPUs for more details.
   • For a Mali Midgard GPU, specify the following options to your gator.ko make command:
     ```
     GATOR_WITH_MALI_SUPPORT=MALI_MIDGARD # Set by CONFIG_GATOR_MALI_MIDGARD
     # gator source needs access to headers under .../kernel/drivers/gpu/arm/...
     # a default of . is suitable for in-tree builds
     DDK_DIR=".../<path_to_Mali_DDK_kernel_files>"
     ```
     To add the corresponding support to the Mali drivers, user space needs gator=1.
     Kernel space needs CONFIG_MALI_GATOR_SUPPORT=y
     See the documentation that is supplied with the Arm DDK for Mali Midgard GPUs for more details.

2. Install the gator driver as a kernel module as normal:
   ```
   insmod gator.ko
   ```
   **Note**
   If you chose to build the Mali driver out of tree, install it before installing the gator driver.

3. Verify that you built the module successfully:
   ```
   ls -l /dev/gator/events/ARM_Mali*
   ```
   This command produces a list of counters.
   If you have successfully built the gator driver with support for Mali technology, you can run a capture session on a Mali-based target.
   Follow the normal instructions for setting capture options and triggering a capture session.

**Related tasks**
3.2 Starting a capture session on page 3-59
2.6 Building the gator module on page 2-42

2.12.4 Setting up Arm Streamline to support Arm® Mali™-V500

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

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

1. Start `gatord`. You can use either user space or kernel space `gatord`. It checks whether the file `/dev/mv500` exists. If it exists, this means that the kernel is configured for Mali-V500.

2. Select the Mali-V500 specific counters you are interested in using the **Counter Configuration** dialog box.

3. Start video decoding and start a capture, in either order.

Streamline displays the captured information.
2.13 Recommended compiler options

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

When using GCC or Clang, use the following options:

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

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

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

- **-marm**
  
  When building for AArch32, this option is required if GCC was compiled with the **--with-mode=thumb** option enabled. Using the **--with-mode=thumb** option without **-marm** breaks call stack unwinding in Arm Streamline.

### Call-stack unwinding

When using user mode gator either as root or non-root user, you must provide additional compiler flags for call stack unwinding to work:

For AArch64 applications, the flag **-fno-omit-frame-pointer** is required. **-mno-omit-leaf-frame-pointer** must also be set on GCC. **-mno-omit-leaf-frame-pointer** is not supported on Clang, therefore the caller for samples in leaf functions will be missing from the stack trace, unless you use gator.ko. If you are using gator.ko, **-fno-omit-frame-pointer** is not required, as it is an unnecessary overhead in this case.

For AArch32 applications, the flags **-fno-omit-frame-pointer** and **-marm** are required. If you are not using gator.ko, you must also set **-mapcs-frame**.

**Note**

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

### Android

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

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

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

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

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

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

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

```
readelf -S .../images/*.dex
```
Related information
readelf
Chapter 3
Target and Data Views

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

It contains the following sections:

• 3.1 Target view on page 3-58.
• 3.2 Starting a capture session on page 3-59.
• 3.3 Arm Streamline Data view overview on page 3-63.
• 3.4 Re-analyzing stored Arm Streamline capture data on page 3-65.
• 3.5 Duplicating a capture on page 3-67.
• 3.6 Merging captures on page 3-68.
• 3.7 Setting preferences on page 3-70.
• 3.8 Importing perf data on page 3-71.
• 3.9 Importing SoC Designer profiling data on page 3-73.
3.1 Target view

This view allows you to set up the target connection, configure the settings for capture and analysis, and select counters before starting a capture.

The following controls are available in the toolbar of the Target view:

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

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

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

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

- **Address field**
  
  Network name or IP address of the target. To search for an available target, enter the network name or IP address of your target, or click the Browse for target button.

  **Browse for target**
  
  Opens the Connection Browser dialog box, which lists the available targets on your network. Select one, then click the Select button. The Connection field updates to show the newly selected target.
3.2 Starting a capture session

To start a capture session, select a target then click the start capture button.

**Prerequisites**

The gator daemon, and optionally the gator driver, must be running on your target.

**Procedure**

1. To select a target, either:
   - Click the **Browse for a target** button to open the **Connection Browser** dialog.
   - Enter an IP address in the **Address** field in the **Target** view toolbar.
   - On an Android target, where the ADB forwards the port, enter **localhost** in the **Address** field.

   **Note**

   If you selected an Android device, in the **Connection Browser** dialog, which is connected using ADB, the **Address** field is automatically populated.

2. To start a capture, click the **Start capture** button.

![Figure 3-2  Starting a capture session](image)

If an up-to-date version of gator is running on your target, **Live** view opens and begins plotting capture data on the bar graphs in real time. Otherwise, a message box is opened, showing the version of gator that Arm Streamline requires.

3.2.1 Connection Browser dialog box

From the **Connection Browser** dialog box, you can select a target without having to look up its name or IP address.

When the **Connection Browser** dialog is launched, Arm Streamline searches your network and provides a selectable list of possible targets.
The **Connection Browser** sorts the targets by type:

**Streamline Agent**

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

**Streamline Agent via ADB**

Lists all Android targets that are connected to the host using ADB or Tizen targets that are connected using SDB, and have gator installed and running. You must have installed ADB or SDB, and set the path to it in the dialog at **Window > Preferences > Streamline > External Tools**, in order for this list to be populated.

- To connect to a target, select a target from the list and click **Select**.
- To add a target, or to change the connection settings of a target, click **Setup Target**.

**Related references**

4.1 Capture & Analysis Options dialog box settings on page 4-77  
Chapter 3 Target and Data Views on page 3-57  
3.2.2 Setup Target dialog box on page 3-60

### 3.2.2 Setup Target dialog box

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

----------  
**Note**
----------

When configuring a Linux target over SSH, the credentials supplied for the target device will be stored in a preferences file in your home directory. The file is encrypted using a simple algorithm. If you are concerned about the security of these credentials, do not use this feature.
Figure 3-4  Automatically setting up Arm Streamline on the target

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

Click Install to install and run a pre-built user space gatord binary which runs on Android or Linux and on Armv7 or Armv8 targets. It runs using the -a flag, so that you can enter a command in the Capture & Analysis Options dialog box without having to restart it.

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

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

Note

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

Related concepts

2.2.1 Comparison of user space gator and kernel space gator on page 2-32
12.8 Enabling atrace and ttrace annotations on page 12-221

Related references

2.11 gatord command-line options on page 2-49

Related references

3.2.1 Connection Browser dialog box on page 3-59
17.2 Troubleshooting target connection issues on page 17-290
3.3 Arm Streamline Data view overview

Use this view to manage your capture files and select captures for viewing or analysis.

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

![Image of Arm Streamline Data view]

**Figure 3-5  Arm Streamline Data view**

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

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

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

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

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

- **Delete**
  Deletes the selected captures from the file system.

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

- **Export Capture File(s)…**
  Opens a dialog box that enables you to save the currently selected .apc directories as zip archives that you can later import into Arm Streamline using the Import Capture File(s)… option. Arm recommends using this option rather than copying .apc directories, because it significantly reduces the file sizes.
Import Capture File(s)...  
Opens a File System dialog box. To import a capture, navigate to a .zip archive that contains a valid .apc directory, or to a perf recording. Perf files are expected to match the naming pattern *perf.data*, *.perf*, or *.data*. If you have used a different name, select the *.* option. Select the file, then click Open. The newly imported capture then appears in the list of captures in the Streamline Data view.

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

3.3.1 Capture color-coding  
Each of the captures in the Arm 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 Arm Streamline. Double-click on a blue capture to open its report.

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

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

![Figure 3-6 Color-coded captures](image-url)
3.4 Re-analyzing stored Arm Streamline capture data

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

There are a number of reasons why you might want to do this, for example:
- To change your analysis options.
- Your sources might not have been available when the data was captured.
- A new version of Streamline might be incompatible with the existing report data.

To re-analyze a Streamline capture:

**Procedure**

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

**Results:**

![Analyze dialog box](image)

**Note**

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

**Related references**

*Chapter 3 Target and Data Views on page 3-57*

*2.9 Capturing data locally on a target on page 2-47*

*4.1 Capture & Analysis Options dialog box settings on page 4-77*
3.5 Duplicating a capture

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

To duplicate a capture:

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

Results:

![Figure 3-8 Duplicating a capture](image)
3.6 Merging captures

You can select and merge multiple captures in the Streamline Data view.

Typically, you can merge captures which were taken separately so that you can easily view the global behavior of your system.

Procedure

1. In the Streamline Data tab, select the captures that you require to be merged.
   Only Perf and gator protocol captures are supported. Previously merged, and bare metal captures are not supported.
2. Right-click the selected captures then select Merge Capture Files... in the context menu.
3. In the Merge captures dialog box, edit the default settings as needed.
4. Click Ok. A merge job is started in the background. When it has finished, the merged capture is shown in the data browser and is ready for analysis.

3.6.1 Merge captures dialog box

In the Merge captures dialog box, the start time, relative time delays of the captures, and the merged file name and location can be changed.

![Merge captures dialog box](image)

Figure 3-9 Merge captures dialog box

The Merge captures dialog box contains the following settings:

**Capture**

Names of captures to be merged.
Start Delay

Relative start delay in nanoseconds.

You can adjust the relative timing offsets of the captures if, for example, the clocks were not synchronized or set correctly when the capture was taken.

Typically, you should set the start delay of capture that started first as 0ns, and enter the times at which the other captures were started.

Synchronization

The Timestamp button sets the start delays back to the original values taken from the captures.

Start date and time

The start date and time is shown for information only. It is not used in the analysis.

Save location

Full file name of the merged capture file. By default, Arm Streamline creates a file name that is based on the names of the captures to be merged.

Related concepts

6.14 Merged captures on page 6-161
3.7 Setting preferences

Open the Arm Streamline Preferences dialog by selecting Window > Preferences > Streamline. Here you can select the theme for the Timeline view. The following subsections contain further options:

Bare Metal

Select Import ITM traces as synchronized streams to make Arm Streamline assume that ITM traces are synchronized from the start, and not discard data before the first synchronization packet in the trace.

Confirmations

Select the Prompts to enable confirmation prompts when deleting captures or templates, or removing charts or chart series. All prompt options are selected by default.

Select Always run operations in background to hide the progress dialog. Progress information is then displayed in the Progress view. Enable the Progress view by selecting Window > Progress.

Select Enable Customer Experience Improvement Program to allow Arm to collect anonymous information about how you are using Streamline. Arm uses this information to improve the Streamline features that you use most often. This option is selected by default.

Data Locations

Lists the locations from which you can load Arm Streamline captures. The first location in the list is the default save location. Reordering the list changes the order in which the captures appear in the documents view.

Note

There is an example Arm Streamline capture in the xaos directory. xaos is one of the Arm Development Studio Linux examples in DS_install_directory/examples/Linux_examples.zip. After unzipping the file, click New then select the xaos project directory to display the example capture in the Arm Streamline Data view.

Note

The default location depends on your OS:

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

External Tools

Enter the path to the ADB executable in the ADB Path field.

Enter the path to the Graphics Analyzer executable in the Graphics Analyzer Installation field.

Select Continue live capture after Graphics Analyzer is activated to instruct Arm Streamline to continue displaying the live capture of data after Graphics Analyzer launches and begins tracing. Otherwise the Live view in Arm Streamline is stopped.
3.8 Importing perf data

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

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

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

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

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

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

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

The parameters for this command are as follows:

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

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

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

Sample Rate: Low
  cpu_clock/period=10000000/

Sample Rate: None
  Do not include this parameter.
```

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

An example of this command in use could be:

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

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

An example for multiple different groups:

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

An example with more, generally low frequency, events:

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

Other supported combinations of events and flags are as follows:
Frequency-based sampling
   perf record -a -F 1000 -e branch-misses

Single process sampling
   perf record -e instructions -- dmesg

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

Related references
17.5 Troubleshooting perf import issues on page 17-296
3.9 Importing SoC Designer profiling data

You can import SoC Designer System Analyzer profiling data into Arm Streamline.

SoC Designer System Analyzer outputs profiling data to a `.profile` file.

- You can import the profiling data into Arm Streamline by clicking **Import Capture File(s)...** and selecting a `.profile` file.
- When the data has been imported, you can:
  - View the counters that are recorded in the file.
  - View a function trace in the heatmap.
  - View a detailed function trace in CAM view.

Arm Streamline `.profile` importing is compatible with SoC Designer version 9.2 and later.

**Note**

- In Arm Streamline, execution time is shown in seconds. In profile data from SoC Designer System Analyzer, time is shown in cycles.
- To view function traces, you must enable it in SoC Designer System Analyzer before you record the profile.
Figure 3-10 An imported SoC Designer System Analyzer profile in Heat Map view
Figure 3-11  An imported SoC Designer System Analyzer profile in CAM view
Chapter 4
Capture and Analysis Options

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

It contains the following section:
• 4.1 Capture & Analysis Options dialog box settings on page 4-77.
4.1 Capture & Analysis Options dialog box settings

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

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

The Connection section contains the following settings:

- **Address:** 10.2.195.169
- **Warning:** Before establishing connections using ADB, you may need to set up ADB path.
Address

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

Note

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

Browse for a target button ( )

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

Capture

The Capture section contains the following options:

Sample Rate

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

Buffer Mode

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

<table>
<thead>
<tr>
<th>Mode</th>
<th>Buffer size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large</td>
<td>16MB</td>
</tr>
<tr>
<td>Medium</td>
<td>4MB</td>
</tr>
<tr>
<td>Small</td>
<td>1MB</td>
</tr>
</tbody>
</table>

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

Note

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

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

--- Note ---

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

Discard Data

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

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

Duration

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

Use more efficient ftrace collection

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

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

Working Directory

The absolute path of the directory on the target in which to run the command that is specified in the Command field. If a path is not given, the path to the current working directory of the gator process is used.

User Name

The user account to run the command as. By default, the command runs as the same user as gator. For example, if gator is run as root, the command runs as root.

Command

A command to run on the target. The command is run a few seconds after the capture begins.

Stop Capture

When this checkbox is set, the capture stops when the command specified in the Command field exits. If the checkbox is clear, the capture continues to run after the command finishes.

Energy Capture

The following settings define your energy capture device:

Energy Capture drop-down menu

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

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

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

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

Port

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

Tool Path

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

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

Energy

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

Power

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

Voltage

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

Current

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

Resistance

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

Analysis

The Analysis section contains the following controls:

Process Extra Debug Information (when available) checkbox

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

Note

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

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

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

Program Images

Use this area to explore your file system and define the images and libraries that you want to profile. The images that you define here do not affect the data that is collected during the capture session. They only affect the report data that is automatically generated after the termination of the capture session. These images can be changed when you re-analyze the stored capture.

--- Note ---

- When compiling images on your host, ensure that you use the `-g` compiler option to enable debug symbols.
- Disabling inlining with the `-fno-inline` compiler option substantially improves the call path quality.
- As an alternative to manually selecting images and libraries on the host in the Program Images section, you can use the image download functionality described in [6.5 Image download on page 6-135](#).

The following buttons are included in the Program Images section:

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

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

**Add ELF Image…**

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

**Select Separate Debug Image…**

Click to select a separate file containing the extracted debug information and CFI. This button is available when the **Debug Info** or **CFI** columns display the icon.

**Remove**

Removes the selected entries.

The Program Images table has the following column headings:

**Use**

Select the checkboxes to include the corresponding images in the analysis. Deselected images are ignored.

**Name**

The name of the 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. This information is used to determine line numbers and inlining information.

عمار 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. This icon is only displayed if the image is an APK.

CFI
Displays the following icons to indicate whether 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.

عمار 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. This icon is only displayed if the image is an APK.

Remarks
Further remarks about the image.

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

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

Save
Saves the settings and exits.

Cancel
Discards all changes and exits.

Related concepts
3.3 Arm Streamline Data view overview on page 3-63
6.1 Live view overview on page 6-116
12.8 Enabling atrace and ttrace annotations on page 12-221

Related tasks
3.2 Starting a capture session on page 3-59
3.4 Re-analyzing stored Arm Streamline capture data on page 3-65
11.8 Setting up National Instrument Multifunction Data Acquisition devices (NI DAQ) to capture energy data on page 11-209
Related references

3.1 Target view on page 3-58
3.2.1 Connection Browser dialog box on page 3-59
2.11 gatord command-line options on page 2-49
17.2 Troubleshooting target connection issues on page 17-290
17.8 Troubleshooting report issues on page 17-301
Chapter 5
Counter Configuration

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

It contains the following sections:

• 5.1 Opening the Counter Configuration dialog box on page 5-86.
• 5.2 Counter Configuration dialog box structure on page 5-87.
• 5.3 Adding new events to the Events to Collect list on page 5-89.
• 5.4 Removing events from the Events to Collect list on page 5-90.
• 5.5 Counter Configuration dialog box settings on page 5-91.
• 5.6 Events specific to Arm® Mali™ technology on page 5-92.
• 5.7 Mali Midgard counters on page 5-93.
• 5.8 Mali Bifrost counters on page 5-101.
• 5.9 Event-based sampling on page 5-110.
• 5.10 Setting up event-based sampling on page 5-111.
• 5.11 Statistical Profiling Extension on page 5-112.
5.1 **Opening the Counter Configuration dialog box**

Arm Streamline uses a default best-fit set of hardware performance counters to aid in the analysis of your applications. You can edit the set of counters with the **Counter Configuration** dialog box, which is accessed through the **Target** view.

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

**Procedure**

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

![Figure 5-1 Target view - Counter Configuration](image)

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

**Related references**

5.5 **Counter Configuration dialog box settings** on page 5-91
5.2 Counter Configuration dialog box structure

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

Figure 5-2 Counter Configuration dialog box

The dialog box contains the following areas:
Available Events

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

Note

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

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

Note

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

Events to Collect

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

Event Based Sampling

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

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

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

**Procedure**

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

**Results:**
The added events appear in the Events to Collect list under their category name.

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

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

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

![Figure 5-3 Interface drop-down menu](image)

**Related concepts**
6.4.4 Chart configuration templates on page 6-133

**Related references**
5.5 Counter Configuration dialog box settings on page 5-91
5.4 Removing events from the Events to Collect list

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

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

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

    **Note**

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

**Related references**

5.5 **Counter Configuration dialog box settings** on page 5-91
5.5 Counter Configuration dialog box settings

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

It contains the following options:

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

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

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

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

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

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

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

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

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

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

**Help**
Opens **Counter Configuration** help.

**Related concepts**
- 5.9 Event-based sampling on page 5-110
- 6.4.4 Chart configuration templates on page 6-133
5.6 Events specific to Arm® Mali™ technology

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

For information about the meaning of the Mali-specific events and how to interpret the profiling data, see the Mali GPU Application Optimization Guide. The guide is available on Arm Developer, https://developer.arm.com/graphics/developer-guides.

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

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

Related references
5.7 Mali Midgard counters on page 5-93
5.8 Mali Bifrost counters on page 5-101
5.7 Mali Midgard counters

Arm Streamline can capture performance counters for each functional block in the design of a Mali Midgard GPU. These blocks are Job Manager, Shader cores, Tiler, and L2 caches. This topic describes the counters in each block.

Midgard GPUs implement many performance counters natively in the hardware. You can also generate derived counters by combining raw hardware counters. This topic describes all the Mali Midgard counters that are available in Arm Streamline, and some of the useful derived counters. To minimize the impact on performance and power from extra hardware logic, many of the counters are close approximations of the described behavior. Therefore there may be slight deviations from the expected behavior.

Note

- The naming convention for Mali Midgard counters in Arm Streamline is ARM_Mali-T<GPUID>_<COUNTER_NAME>.
- If a counter is only available for specific GPUs, the description states which GPUs it is available for.

Job Manager counters

This section describes the counters that the Mali Job Manager component implements.

The following counters provide information about the number of cycles that the GPU spends processing workloads:

**GPU_ACTIVE**
Increments for every cycle that the GPU has any workload queued in a Job slot, or the GPU cycle counter is running for OpenCL profiling. If the GPU is waiting for external memory to return data, it is still counted as active if there is a workload in the queue.

**GPU_UTILIZATION (Derived)**
The overall GPU utilization.

\[
\text{GPU(Utilization)} = \frac{\text{GPU_ACTIVE}}{\text{GPU_MHZ}}
\]

Note
If your device supports Dynamic Frequency and Voltage Scaling (DVFS), the GPU frequency is often not constant while running content. If possible, disable platform DVFS to lock the processor, GPU, and memory bus at a fixed frequency.

**JS0_ACTIVE**
Increments every cycle that the GPU is running a Job chain in Job slot 0. Corresponds directly to fragment shading workloads because this Job slot is only used for processing fragment Jobs.

**JS1_ACTIVE**
Increments every cycle that the GPU is running a Job chain in Job slot 1. This Job slot can be used for compute shaders, vertex shaders, and tiling workloads, but the counter does not differentiate between them.

**JS2_ACTIVE**
Increments every cycle that the GPU is running a Job chain in Job slot 2. This Job slot can be used for compute shaders and vertex shaders.

**IRQ_ACTIVE**
Increments every cycle that the GPU has an interrupt waiting to be handled by the driver running on the processor.

The following counters relate to how the Job Manager issues work to the shader cores:
JS0_TASKS (1)
Available for Mali-T600, Mali-T620, and Mali-T720.
Increments for every task the Job Manager issues to a shader core. For JS0, these tasks correspond to a 16x16 pixel screen region.

JS0_TASKS (2)
Available for Mali-T760 and Mali-T800 series.
Increments for every task the Job Manager issues to a shader core. For JS0, these tasks correspond to a 32x32 pixel screen region.

JS1_TASKS
Increments for every task the Job Manager issues to a shader core or the tiler. For JS1, these tasks correspond to a range of vertices or compute work items for shader cores, or a range of indices for the tiler.

JS2_TASKS
Increments for every task the Job Manager issues to a shader core. For JS2, these tasks correspond to a range of vertices or compute work items.

Shader Core counters
This section describes the counters that the Mali Shader Core implements. The GPU hardware records each counter separately for each shader core. Arm Streamline displays the average value of the counter across all the shader cores.

Note
This section refers to fragment workloads or compute workloads. Vertex workloads are treated as a one dimensional compute problem by the shader core, so they are counted as a compute workload.

The following counters show the total activity level of the shader core:

FRAG_ACTIVE
Increments every cycle that at least one fragment task is active inside the shader core.

COMPUTE_ACTIVE
Increments every cycle that at least one compute task is active inside the shader core.

TRIPIPE_ACTIVE
Increments every cycle that at least one thread is active inside the programmable tri-pipe.

TRIPIPE_UTILIZATION (Derived)
An approximation of the overall utilization of the tri-pipe.

\[
\text{TRIPIPE\_UTILIZATION} = \frac{\text{TRIPIPE\_ACTIVE}}{\text{GPU\_ACTIVE}}
\]

The following counters show the task and thread issue behavior of the fixed function compute frontend of the shader core, which issues work into the programmable tri-pipe:

COMPUTE_TASKS
Increments for every compute task that the shader core handles.

COMPUTE_THREADS
Increments for every compute thread that the shader core spawns.

The following counters show the task and thread issue behavior of the fixed-function fragment frontend of the shader core:

FRAG_PRIMITIVES
Increments for every primitive that is read from the tile list.
FRAG_PRIMITIVES_DROPPED
Increments for every primitive that is read from the tile list and then discarded because it is not relevant for the tile being rendered.

THREADS_PER_PRIMITIVE_LOAD (Derived)
The number of fragment threads that are issued per primitive.

\[
\text{THREADS\_PER\_PRIMITIVE\_LOAD} = \frac{\text{FRAG\_THREADS}}{\text{FRAG\_PRIMITIVES} - \text{FRAG\_PRIMITIVES\_DROPPED}}
\]

FRAG_QUADS_RAST
Increments for every 2x2 pixel quad that the rasterization unit rasterizes.

FRAG_QUADS_EZS_TEST
Increments for every 2x2 pixel quad that undergoes early depth and stencil (ZS) test and update operations.

FRAG_QUADS_EZS_KILLED
Increments for every 2x2 pixel quad that early ZS testing kills.

FRAG_CYCLES_NO_TILE
Increments every cycle that a lack of available physical tile memory blocks the shader core early ZS unit from progressing.

FRAG_CYCLES_QUADS_BUFFERED
Not available for Mali-T600.
Increments every cycle that the quad buffer contains at least one 2x2 pixel quad waiting to be executed in the tri-pipe.

FRAG_THREADS
Increments for every real or dummy fragment thread that the GPU creates.

FRAG_DUMMY_THREADS
Increments for every dummy fragment thread that the GPU creates.

The following counters record the fragment backend behavior:

FRAG_THREADS_LZS_TEST
Increments for every thread that triggers late ZS testing.

FRAG_THREADS_LZS_KILLED
Increments for every thread that late ZS testing kills.

FRAG_NUM_TILES (1)
Available for Mali-T600, Mali-T620, and Mali-T720.
Increments for every 16x16 pixel tile that the shader core renders.

FRAG_NUM_TILES (2)
Available for Mali-T760 and Mali-T800 series
Increments for every 32x32 pixel tile that the shader core renders.

FRAG_TRANS_ELIM
Increments for every physical rendered tile that has its writeback canceled due to a matching transaction elimination CRC hash.

The following counters show the behavior of the arithmetic pipe:

ARITH_WORDS (1)
Not available for Mali-T720, Mali-T820, and Mali-T830
Increments for every arithmetic instruction that is architecturally executed. This counter is normalized to give the per pipe performance.
ARITH_WORDS (2)
Available for Mali-T720, Mali-T820, and Mali-T830.
Increments for every batched arithmetic instruction that is executed.

ARITH_ARCH_UTILIZATION (Derived)
The utilization of the arithmetic hardware.
\[ \text{ARITH_ARCH_UTILIZATION} = \frac{\text{ARITH_WORDS}}{\text{TRIPIPE_ACTIVE}} \]

The following counters show the behavior of the load/store pipe:

LS_WORDS
Increments for every load or store instruction that is architecturally executed.

LS_ARCH_UTILIZATION (Derived)
The architectural utilization of the load store pipe.
\[ \text{LS_ARCH_UTILIZATION} = \frac{\text{LS_WORDS}}{\text{TRIPIPE_ACTIVE}} \]

LS_ISSUES (1)
Available for Mali-T600, Mali-T620, and Mali-T720.
Increments for every load or store instruction that is issued, or reissued due to varying or data cache misses.

LS_ISSUES (2)
Available for Mali-T760 and Mali-T800 series.
Increments for every load or store instruction that is issued, or reissued due to varying cache misses.

LS_UARCH_UTILIZATION (Derived)
The microarchitectural utilization.
\[ \text{LS_UARCH_UTILIZATION} = \frac{\text{LS_ISSUES}}{\text{TRIPIPE_ACTIVE}} \]

LS_CPI (Derived)
Cycles Per Instruction.
\[ \text{LS_CPI} = \frac{\text{LS_ISSUES}}{\text{LS_WORDS}} \]

The following counters monitor the performance of the load/store cache:

LSC_READ_HITS
Increments for every load/store L1 cache read access that is a hit.

LSC_READ_MISSES
Available for Mali-T600, Mali-T620, and Mali-T720.
Increments for every load/store L1 cache read access that is a miss.

LSC_READ_HITRATE (Derived)
The percentage of the total number of read accesses that are hits.
\[ \text{LSC_READ_HITRATE} = \frac{\text{LSC_READ_HITS}}{(\text{LSC_READ_HITS} + \text{LSC_READ_MISSES})} \]

LSC_READ_OPS
Available for Mali-T760 and Mali-T800 series.
Increments for every load/store L1 cache read access.

LSC_READ_HITRATE (Derived)
The percentage of the total number of read accesses that are hits.
\[ \text{LSC_READ_HITRATE} = \frac{\text{LSC_READ_HITS}}{\text{LSC_READ_OPS}} \]
LSC_WRITE_HITS
Increments for every load/store L1 cache write access that is a hit.

LSC_WRITE_MISSES
Available for Mali-T600, Mali-T620, and Mali-T720.
Increments for every load/store L1 cache write access that is a miss.

LSC_WRITE_HITRATE (Derived)
The percentage of the total number of write accesses that are hits.

\[
LSC_{WRITE\_HITRATE} = \frac{LSC\_WRITE\_HITS}{LSC\_WRITE\_HITS + LSC\_WRITE\_MISSES}
\]

LSC_WRITE_OPS
Available for Mali-T760 and Mali-T800 series.
Increments for every load/store L1 cache write access.

LSC_WRITE_HITRATE (Derived)
The percentage of the total number of write accesses that are hits.

\[
LSC_{WRITE\_HITRATE} = \frac{LSC\_WRITE\_HITS}{LSC\_WRITE\_OPS}
\]

LSC_ATOMIC_HITS
Increments for every atomic memory access that hits in the L1 atomic cache.

LSC_ATOMIC_MISSES
Available for Mali-T600, Mali-T620, and Mali-T720.
Increments for every atomic memory access that misses in the L1 atomic cache.

LSC_ATOMIC_HITRATE (Derived)
The percentage of the total number of atomic memory accesses that are hits.

\[
LSC_{ATOMIC\_HITRATE} = \frac{LSC\_ATOMIC\_HITS}{LSC\_ATOMIC\_HITS + LSC\_ATOMIC\_MISSES}
\]

LSC_ATOMIC_OPS
Available for Mali-T760 and Mali-T800 series.
Increments for every atomic memory access that misses in the L1 atomic cache.

LSC_ATOMIC_HITRATE
The percentage of the total number of atomic memory accesses that are hits.

\[
LSC_{ATOMIC\_HITRATE} = \frac{LSC\_ATOMIC\_HITS}{LSC\_ATOMIC\_OPS}
\]

LSC_LINE_FETCHES
Increments for every line that the L1 cache fetches from the L2 memory system.

LSC_DIRTY_LINE
Increments for every dirty line that is evicted from the L1 cache into the L2 memory system.

LSC_SNOOPS
Increments for every snoop into the L1 cache from the L2 memory system.

The following counters show the texture pipe behavior:

TEX_WORDS
Increments for every architecturally executed texture instruction.

TEX/issues (1)
Available for Mali-T600, Mali-T620, and Mali-T720.
Increments for every texture issue cycle used. Some instructions take more than one cycle due to data cache misses or multi-cycle filtering operations.
TEX_ISSUES (2)

Available for Mali-T760 and Mali-T800 series.
Increments for every texture issue cycle used. Some instructions take more than one cycle due to multi-cycle filtering operations.

TEX_CPI (Derived)
Cycles Per Instruction.

\[
TEX\_CPI = \frac{TEX\_ISSUES}{TEX\_WORDS}
\]

Tiler counters

The tiler counters provide details of the workload of the fixed function tiling unit. This unit places primitives into the tile lists that the fragment frontend reads during fragment shading.

The following counters show the overall activity of the tiling unit:

TI_ACTIVE
Available for Mali-T600, Mali-T620, Mali-T760, Mali-T860, and Mali-T880
Increments every cycle that the tiler processes a task.

The following counters give a functional breakdown of the tiling workload that is given to the GPU by the application:

TI_POINTS
Increments for every point primitive that the tiler processes. This counter increments before any clipping or culling, so it reflects the raw workload from the application.

TI_LINES
Increments for every line segment primitive that the tiler processes. This counter increments before any clipping or culling, so it reflects the raw workload from the application.

TI_TRIANGLES
Increments for every triangle primitive that the tiler processes. This counter increments before clipping or culling, so it reflects the raw workload from the application.

The following counters show how clipping and culling affect the workload:

TI_PRIM_VISIBLE
Increments for every primitive that is visible according to its type, clip-space coordinates, and front-face or back-face orientation.

TI_PRIM_CULLED
Increments for every primitive that is culled due to the application of front-face or back-face culling rules.

TI_PRIM_CLIPPED
Increments for every primitive that is culled because it is outside of the clip-space volume.

TI_FRONT_FACING
Incremented for every triangle that is front-facing. This counter increments after culling, so it only counts visible primitives that are emitted into the tile list.

TI_BACK_FACING
Incremented for every triangle that is back-facing. This counter increments after culling, so it only counts visible primitives that are emitted into the tile list.

L2 Cache Counters

This section describes the behavior of the L2 memory system counters. In systems that implement multiple L2 caches or bus interfaces, the counters that Arm Streamline displays are the sum of the counters from all the L2 counter blocks.
The following counters profile the internal read traffic into the L2 cache from the various internal masters:

**L2_READ_LOOKUP**
Increments for every read transaction that the L2 cache receives.

**L2_READ_HITS**
Increments for every read transaction that the L2 cache receives that also hits in the cache.

**L2_READ_HITRATE** *(Derived)*
The percentage of read transactions that the L2 cache receives that also hit in the cache.

\[
L2\text{\_READ\_HITRATE} = \frac{L2\text{\_READ\_HITS}}{L2\text{\_READ\_LOOKUP}}
\]

**L2_READ_SNOOP**
Increments for every inner coherent read snoop transaction that the L2 cache receives.

The following counters profile the internal write traffic into the L2 cache from the various internal masters:

**L2_WRITE_LOOKUP**
Increments for every write transaction that the L2 cache receives.

**L2_WRITE_HITS**
Increments for every write transaction that the L2 cache receives that also hits in the cache.

**L2_WRITE_HITRATE** *(Derived)*
The percentage of read transactions that the L2 cache receives that also hit in the cache.

\[
L2\text{\_WRITE\_HITRATE} = \frac{L2\text{\_WRITE\_HITS}}{L2\text{\_WRITE\_LOOKUP}}
\]

**L2_WRITE_SNOOP**
Increments for every inner coherent write snoop transaction that the L2 cache receives.

The following counters profile the external read memory interface behavior:

--- **Note** ---
This behavior includes traffic from the entire GPU L2 memory subsystem as some types of access bypass the L2 cache.

**L2_EXT_READ_BEATS**
Increments on every clock cycle that a read beat is read off the AXI bus.

**L2_EXT_READ_BYTES** *(Derived)*
Converts the beat counter into a raw bandwidth counter.

\[
L2\text{\_EXT\_READ\_BYTES} = L2\text{\_EXT\_READ\_BEATS} \times L2\text{\_AXI\_WIDTH\_BYTES}
\]

**L2_EXT_READ_UTILIZATION** *(Derived)*
The total percentage of available AXI port bandwidth that is used.

\[
L2\text{\_EXT\_READ\_UTILIZATION} = \frac{L2\text{\_EXT\_READ\_BEATS}}{(L2\text{\_AXI\_PORT\_COUNT} \times \text{GPU\_ACTIVE})}
\]

--- **Note** ---
Normalize the number of read beats accumulated by Arm Streamline into a per-port count. If a design uses one to four shader cores, a single AXI port is present. Otherwise two AXI ports are present.
L2_EXT_R_BUF_FULL
Not available for Mali-T720.
Increments every cycle that the GPU is unable to create a read transaction because there are no free entries in the internal response buffer.

L2_EXT_RD_BUF_FULL
Not available for Mali-T720.
Increments if a read response is received and the internal read data buffer is full.

L2_EXT_AR_STALL
Increments every cycle that the GPU is unable to issue a new read transaction to AXI because AXI is unable to accept the request.
The following counters profile the external write memory interface behavior:

Note
This behavior includes traffic from the entire GPU L2 memory subsystem.

L2_EXT_WRITE_BEATS
Increments on every clock cycle that a write data beat is sent on the AXI bus.

L2_EXT_WRITE_BYTES (Derived)
Converts the beat counter into a raw bandwidth counter.

\[
L2\_EXT\_WRITE\_BYTES = L2\_EXT\_WRITE\_BEATS \times L2\_AXI\_WIDTH\_BYTES
\]

L2_EXT_WRITE_UTILIZATION (Derived)
The percentage of available AXI port bandwidth used.

\[
L2\_EXT\_WRITE\_UTILIZATION = \frac{L2\_EXT\_WRITE\_BEATS}{(L2\_AXI\_PORT\_COUNT \times GPU\_ACTIVE)}
\]

Note
Normalize the number of read beats accumulated by Arm Streamline into a per-port count.

L2_EXT_W_BUF_FULL
Increments every cycle that the GPU is unable to create a new write transaction because there are no free entries in the internal write buffer.

L2_EXT_W_STALL
Increments every cycle that the GPU is unable to issue a new write transaction to AXI because AXI is unable to accept the request.

For more information about these counters, see Mali Midgard Family Performance Counters on Arm Community.
### 5.8 Mali Bifrost counters

Arm Streamline can capture performance counters for each functional block in the design of a Bifrost GPU. These blocks are Job Manager, Shader cores, Tiler, and L2 caches. This topic describes the counters in each block.

Bifrost GPUs implement many performance counters natively in the hardware. You can also generate derived counters by combining raw hardware counters. This topic describes all the Mali Bifrost counters that are available in Arm Streamline, and some of the useful derived counters. To minimize the impact on performance and power from extra hardware logic, many of the counters are close approximations of the described behavior. Therefore there may be slight deviations from the expected behavior.

#### Job Manager counters

This section describes the counters that the Mali Job Manager component implements.

The following counters provide top-level activity information. For example, information about the number of cycles that the GPU spends processing a workload, or waiting for the handling of completion interrupts:

- **JM.GPU_ACTIVE**
  Increments every cycle that the GPU has any workload queued in a Job slot. If the GPU is waiting for external memory to return data, it is still counted as active if there is a workload in the queue.

- **JM.GPU_UTILIZATION** *(Derived)*
  The overall GPU utilization.

\[
JM.GPU_UTILIZATION = \frac{JM.GPU_ACTIVE}{GPU_MHZ}
\]

- **JM.JS0_ACTIVE**
  Increments every cycle that the GPU is running a Job chain in Job slot 0. Corresponds directly to fragment shading workloads because this Job slot is only used for processing fragment Jobs.

- **JM.JS0_UTILIZATION** *(Derived)*
  The percentage JS0 utilization.

\[
JM.JS0_UTILIZATION = \frac{JM.JS0_ACTIVE}{JM.GPU_ACTIVE}
\]

- **JM.JS1_ACTIVE**
  Increments every cycle that the GPU is running a Job chain in Job slot 1. This Job slot can be used for compute shaders, vertex shaders, and tiling workloads, but the counter does not differentiate between them.

- **JM.JS1_UTILIZATION** *(Derived)*
  The percentage JS1 utilization.

\[
JM.JS1_UTILIZATION = \frac{JM.JS1_ACTIVE}{JM.GPU_ACTIVE}
\]

- **JM.IRQ_ACTIVE**
  Increments every cycle that the GPU has an interrupt waiting for the driver running on the processor to handle it.

The following counters provide task dispatch information:

- **JM.JS0_TASKS**
  Increments for every task the Job Manager issues to a shader core. For JS0, these tasks correspond to a 32x32 pixel screen region.

- **JM.PIXEL_COUNT** *(Derived)*
  An approximation of the total scene pixel count.

\[
JM.PIXEL_COUNT = JM.JS0_TASKS \times 32 \times 32
\]
**Shader Core counters**

This section describes the counters that the Mali Shader Core implements. Arm Streamline displays the average of all the shader core counters that the GPU hardware records.

**Note**

This section refers to fragment workloads or compute workloads. Vertex, Geometry, and Tessellation workloads are treated as one dimensional compute problems by the shader core, so they are counted as compute workloads.

The following counters show the total activity level of the shader core:

- **SC.COMPUTE_ACTIVE**
  Increments every cycle that at least one compute task is active inside the shader core.

- **SC.FRAG_ACTIVE**
  Increments every cycle that at least one fragment task is active inside the shader core.

- **SC.EXEC_CORE_ACTIVE**
  Increments every cycle that at least one quad is active inside the programmable execution core.

- **SC.EXEC_CORE_UTILIZATION (Derived)**
  An approximation of the overall utilization of the execution core.
  
  \[
  SC.EXEC_CORE_UTILIZATION = \frac{SC.EXEC_CORE_ACTIVE}{JM.GPU_ACTIVE}
  \]

The following counters show the task and thread issue behavior of the fixed function compute frontend of the shader core:

- **SC.COMPUTE_QUADS**
  Increments for every compute quad that the shader core spawns.

- **SC.COMPUTE_QUAD_CYCLES (Derived)**
  The average number of compute cycles per compute quad.
  
  \[
  SC.COMPUTE_QUAD_CYCLES = \frac{SC.COMPUTE_ACTIVE}{SC.COMPUTE_QUADS}
  \]

The following counters show the task and thread issue behavior of the fixed-function fragment frontend of the shader core:

- **SC.FRAG_PRIMITIVES_RAST**
  Increments for every primitive entering the frontend fixed-function rasterization stage.

- **SC.FRAG_QUADS_RAST**
  Increments for every 2x2 pixel quad that the rasterization unit rasterizes.

- **SC.FRAG_QUADS_EZS_TEST**
  Increments for every 2x2 pixel quad that undergoes ZS testing.

- **SC.FRAG_QUADS_EZS_UPDATE**
  Increments for every 2x2 pixel quad that completes an early ZS update operation.

- **SC.FRAG_QUADS_EZS_KILLED**
  Increments for every 2x2 pixel quad that early ZS testing kills.

- **SC.FRAG_QUADS_KILLED_BY_OVERDRAW (Derived)**
  Increments for every 2x2 pixel quad that survives early ZS testing, but that is overdrawn by an opaque quad before spawning as fragment shading threads in the programmable core.
  
  \[
  SC.FRAG_QUADS_KILLED_BY_OVERDRAW = SC.FRAG_QUADS_RAST - SC.FRAG_QUADS_EZS_KILL - SC.FRAG_QUADS
  \]

- **SC.FRAG_QUADS_OPAQUE**
  Increments for every 2x2 pixel quad that is architecturally opaque that survives early ZS testing.
  Architecturally opaque pixel quads do not use blending, shader discard, or alpha-to-coverage.
**SC.FRAG_QUADS_TRANSPARENT** (Derived)
Increments for every 2x2 pixel quad that is architecturally transparent that survives early ZS testing. Architecturally transparent pixel quads do not use blending, shader discard, or alpha-to-coverage.

\[
\text{SC.FRAG_QUADS_TRANSPARENT} = \text{SC.FRAG_QUADS_RAST} - \text{SC.FRAG_QUADS_EZS_KILL} - \text{SC.FRAG_QUADS_OPAQUE}
\]

**Note**
Transparent in this context means alpha transparency or a shader-dependent coverage mask.

**SC.FRAG_QUAD_BUFFER_NOT_EMPTY**
Increments every cycle that the fragment unit is active and the pre-pipe buffer contains at least one 2x2 pixel quad waiting to be executed in the execution core.

**SC.FRAG_QUADS**
Increments for every fragment quad that the GPU creates.

**SC.FRAG_PARTIAL_QUADS**
Increments for every fragment quad that contains at least one thread slot that has no sample coverage.

**SC.FRAG_PARTIAL_QUAD_PERCENTAGE** (Derived)
Calculates the percentage of spawned quads that have partial coverage.

\[
\text{SC.FRAG_PARTIAL_QUAD_PERCENTAGE} = \frac{\text{SC.FRAG_PARTIAL_QUADS}}{\text{SC.FRAG_QUADS}}
\]

**SC.FRAG_QUAD_CYCLES** (Derived)
Calculates the average number of fragment cycles for each fragment quad.

\[
\text{SC.FRAG_QUAD_CYCLES} = \frac{\text{SC.FRAG_ACTIVE}}{\text{SC.FRAG_QUADS}}
\]

The following counters show the fragment backend behavior:

**SC.FRAG_THREADS_LZS_TEST**
Increments for every thread that triggers late depth and stencil (ZS) testing.

**SC.FRAG_THREADS_LZS_KILLED**
Increments for every thread that late ZS testing kills.

**SC.FRAG_NUM_TILES**
Increments for every tile that is rendered.

**SC.FRAG_TILES_CRC_CULLED**
Increments for every physical rendered tile that has its writeback canceled due to a matching transaction elimination CRC hash.

The following counters show the behavior of the arithmetic execution engine:

**SC.EE_INSTRS**
Increments for every arithmetic instruction that is architecturally executed for a quad in an execution engine. This counter is normalized based on the number of execution engines that the design implements, so it gives the performance per engine.

**SC.EE_UTILIZATION** (Derived)
The utilization of the arithmetic hardware.

\[
\text{SC.EE_UTILIZATION} = \frac{\text{SC.EE_INSTRS}}{\text{SC.EXEC_CORE_ACTIVE}}
\]

**SC.EE_INSTRS_DIVERGED**
Increments for every arithmetic instruction architecturally executed where there is control flow divergence in the quad resulting in at least one lane of computation being masked out.

The following counters show the behavior of the load/store pipe:
**SC.LSC_READS_FULL**
Increments for every LS cache access executed that returns 128 bits of data.

**SC.LSC_READS_SHORT**
Increments for every LS cache access executed that returns less than 128 bits of data.

**SC.LSC_WRITES_FULL**
Increments for every LS cache executed that writes 128 bits of data.

**SC.LSC_WRITES_SHORT**
Increments for every LS cache access executed that writes less than 128 bits of data.

**SC.LSC_ATOMICS**
Increments for every atomic operation that is issued to the LS cache.

**SC.LSC_ISSUES (Derived)**
The total number of load/store cache access operations issued.

\[
SC.LSC\_ISSUES = SC.LSC\_READS\_FULL + SC.LSC\_READS\_SHORT + \\
SC.LSC\_ISSUES = SC.LSC\_WRITES\_FULL + SC.LSC\_WRITES\_SHORT + \\
SC.LSC\_ATOMICS
\]

**SC.LSC_UTILIZATION (Derived)**
Utilization of the load/store cache.

\[
SC.LSC\_UTILIZATION = \frac{SC.LSC\_ISSUES}{SC.EXEC\_CORE\_ACTIVE}
\]

**SC.LSC_READ_BEATS**
Increments for every 16 bytes of data that is fetched from the L2 memory system.

**SC.LSC_L2_BYTES_PER_ISSUE (Derived)**
The average number of bytes read from the L2 cache for each load/store L1 cache access.

\[
SC.LSC\_L2\_BYTES\_PER\_ISSUE = \frac{(SC.LSC\_READ\_BEATS \times 16)}{SC.LSC\_ISSUES}
\]

**SC.LSC_READ_BEATS_EXTERNAL**
Increments for every 16 bytes of data that are fetched from the L2 memory system that missed in the L2 cache and required a fetch from external memory.

**SC.LSC_EXTERNAL_BYTES_PER_ISSUE (Derived)**
The average number of bytes that are read from the external memory interface for each load/store L1 cache access.

\[
SC.LSC\_EXTERNAL\_BYTES\_PER\_ISSUE = \frac{(SC.LSC\_READ\_BEATS\_EXTERNAL \times 16)}{SC.LSC\_ISSUES}
\]

**SC.LSC_WRITE_BEATS**
Increments for every 16 bytes of data that are written to the L2 memory system.

The following counters show the texture pipe behavior:

---
**Note**
---

The texture pipe event counters increment for each thread (fragment), not for each quad.

**SC.TEX_INSTRS**
Increments for every architecturally executed texture instruction.

**SC.TEX_ISSUES**
Increments for every texture issue cycle used.

**SC.TEX_UTILIZATION (Derived)**
The texture unit utilization.

\[
SC.TEX\_UTILIZATION = \frac{SC.TEX\_ISSUES}{SC.EXEC\_CORE\_ACTIVE}
\]
**SC.TEX_CPI (Derived)**
The average cycle usage of the texture unit per instruction.

\[ SC.TEX.CPI = \frac{SC.TEX.ISSUES}{SC.TEX.INSTRS} \]

**SC.TEX_INSTR_3D**
Increments for every architecturally executed texture instruction that accesses a 3D texture.

**SC.TEX_INSTR_TRILINEAR**
Increments for every architecturally executed texture instruction that uses a trilinear minification filter.

**SC.TEX_INSTR_MIPMAP**
Increments for every architecturally executed texture instruction that accesses a texture that has mipmaps enabled.

**SC.TEX_INSTR_COMPRESSED**
Increments for every architecturally executed texture instruction that accesses a texture that is compressed.

**SC.TEX_READ_BEATS**
Increments for every 16 bytes of texture data that is fetched from the L2 memory system.

**SC.TEX_L2_BYTES_PER_ISSUE (Derived)**
The average number of bytes read from the L2 cache per texture L1 cache access.

\[ SC.TEX.L2.BYTES.PER.ISSUE = \frac{(SC.TEX.READ.BEATS \times 16)}{SC.TEX.ISSUES} \]

**SC.TEX_READ_BEATS_EXTERNAL**
Increments for every 16 bytes of texture data fetched from the L2 memory system that missed in the L2 cache and required a fetch from external memory.

**SC.TEX_EXTERNAL_BYTES_PER_ISSUE (Derived)**
The average number of bytes read from the external memory interface per texture operation.

\[ SC.TEX.EXTERNAL.BYTES.PER.ISSUE = \frac{(SC.TEX.READ.BEATS.EXTERNAL \times 16)}{SC.TEX.ISSUES} \]

The following counters show the varying unit behavior:

**SC.VARY_INSTR**
Increments for every architecturally executed varying unit instruction for a fragment quad.

**SC.VARY_ISSUES_16**
Increments for every architecturally executed cycle of mediump 16-bit varying interpolation.

**SC.VARY_ISSUES_32**
Increments for every architecturally executed cycle of highp 32-bit varying interpolation.

**SC.VARY_UTILIZATION (Derived)**
The utilization of the varying unit.

\[ SC.VARY_UTILIZATION = \frac{(SC.VARY.ISSUES._16 + SC.VARY.ISSUES._32)}{SC.EXEC_CORE_ACTIVE} \]

**Tiler counters**
The tiler counters provide details of the workload of the fixed function tiling unit. This unit places primitives into the tile lists that the fragment frontend reads during fragment shading.

The following counters show the overall activity of the tiling unit:

**TI.ACTIVE**
Increments every cycle that the tiler processes a task.

The following counters give a functional breakdown of the tiling workload that is given to the GPU by the application:
TI.PRIMITIVE_POINTS
Increments for every point primitive that the tiler processes. This counter increments before any clipping or culling, so it reflects the raw workload from the application.

TI.PRIMITIVE_LINES
Increments for every line segment primitive that the tiler processes. This counter increments before any clipping or culling, so it reflects the raw workload from the application.

TI.PRIMITIVE_TRIANGLES
Increments for every triangle primitive that the tiler processes. This counter increments before any clipping or culling, so it reflects the raw workload from the application.

TI.INPUT_PRIMITIVES (Derived)
The total number of primitives entering primitive assembly.

\[
\text{TI.INPUT_PRIMITIVES} = \text{TI.PRIMITIVE_POINTS} + \text{TI.PRIMITIVE_LINES} + \text{TI.PRIMITIVE_TRIANGLES}
\]

The following counters give a breakdown of how clipping and culling affect the workload. The culling schemes are applied in the following order:

1. Primitive assembly
2. Facing culling
3. Frustum culling
4. Coverage culling

This order impacts the interpretation of the counters in terms of comparing the culling rates against the total number of primitives entering and leaving each stage.

TI.CULLED_FACING
Increments for every primitive that is culled due to the application of front-face or back-face culling rules.

TI.CULLED_FRUSTUM
Increments for every primitive that is culled due to being outside of the clip-space volume.

TI.CULLED_COVERAGE
Incremented for every microtriangle primitive that is culled due to no coverage of active sample points.

TI.PRIMITIVE_VISIBLE
Incremented for every primitive that is visible and survives all types of culling that are applied.

Note

Visible in this context means that a primitive is inside the viewing frustum, facing in the correct direction, and has at least some sample coverage. Primitives that are visible at this stage may not generate any rendered fragments. For example, ZS testing during fragment processing may determine that a primitive is entirely occluded by other primitives.

TI.CULLED_FACING_PERCENT (Derived)
The percentage of primitive inputs that the facing test culls.

\[
\text{TI.CULLED_FACING_PERCENT} = \frac{\text{TI.CULLED_FACING}}{\text{TI.INPUT_PRIMITIVES}}
\]

TI.CULLED_FRUSTUM_PERCENT (Derived)
The percentage of primitive inputs that the frustum test culls.

\[
\text{TI.CULLED_FRUSTUM_PERCENT} = \frac{\text{TI.CULLED_FRUSTUM}}{(\text{TI.INPUT_PRIMITIVES} - \text{TI.CULLED_FACING})}
\]

TI.CULLED_COVERAGE_PERCENT (Derived)
The percentage of primitive inputs that the coverage test culls.

\[
\text{TI.CULLED_COVERAGE_PERCENT} = \frac{\text{TI.CULLED_FRUSTUM}}{(\text{TI.INPUT_PRIMITIVES} - \text{TI.CULLED_FACING} - \text{TI.CULLED_FRUSTUM})}
\]
**TI.FRONT_FACING**
Incremented for every triangle that is front-facing. This counter increments after culling, so it only counts visible primitives that are emitted into the tile list.

**TI.BACK_FACING**
Incremented for every triangle that is back-facing. This counter increments after culling, so it only counts visible primitives that are emitted into the tile list.

The following counters track the workload requests for the Index-Driver Vertex Shading pipeline:

**TI.IDVS_POSITION_SHADING_REQUEST**
Increments for every batch of triangles that are position shaded. Each batch consists of four vertices from sequential index ranges.

**TI.IDVS_VARYING_SHADING_REQUEST**
Increments for every batch of triangles that are varying shaded. Each batch consists of four vertices from sequential index ranges.

### L2 Cache counters

This section describes the behavior of the L2 memory system counters. In systems that implement multiple L2 caches or bus interfaces, the counters that are presented in Arm Streamline are the sum of the counters from all the L2 counter blocks.

**Note**
All derivations in this topic are computations per slice. As Arm Streamline reports the sum of the slices, it may be necessary to divide these derivations by the number of cache slices present in your design.

The following counters profile the internal use of the L2 cache versus the available cycle capacity:

**L2.ANY_LOOKUP**
Increments for any L2 read or write request from an internal master, or snoop request from an internal or external master.

**L2.INTERNAL_UTILIZATION (Derived)**
The internal utilization of the L2 cache by the processing masters in the system.

\[
L2.\text{INTERNAL\_UTILIZATION} = \frac{L2.\text{ANY\_LOOKUP}}{JM.\text{GPU\_ACTIVE}}
\]

The following counters profile the internal read traffic into the L2 cache from the various internal masters:

**L2.READ_REQUEST**
Increments for every read transaction that the L2 cache receives.

**L2.EXTERNAL_READ_REQUEST**
Increments for every read transaction that the L2 cache sends to external memory.

**L2.READ_MISS_RATE (Derived)**
Indicates the number of reads that are missing and are being sent on the L2 external interface to main memory.

\[
L2.\text{READ\_MISS\_RATE} = \frac{L2.\text{READ\_REQUEST}}{L2.\text{EXTERNAL\_READ\_REQUEST}}
\]

**L2.WRITE_REQUEST**
Increments for every write transaction that the L2 cache receives.

**L2.EXTERNAL_WRITE_REQUEST**
Increments for every write transaction that the L2 cache sends to external memory.
L2.WRITE_MISS_RATE (Derived)
Indicates the number of writes that are missing and are being sent on the L2 external interface to main memory.

\[
L2\text{.WRITE\_MISS\_RATE} = \frac{L2\text{.WRITE\_REQUEST}}{L2\text{.EXTERNAL\_WRITE\_REQUEST}}
\]

The following counters profile the external read memory interface behavior:

--- Note ---
This behavior includes traffic from the entire GPU L2 memory subsystem as some types of access bypass the L2 cache.

---

L2.EXTERNAL_READ_BEATS
Increments on every clock cycle that a read beat is read off the external AXI bus.

L2.EXTERNAL_READ_BYTES (Derived)
Converts the beat counter into a raw bandwidth counter.

\[
L2\text{.EXTERNAL\_READ\_BYTES} = \sum (L2\text{.EXTERNAL\_READ\_BEATS} \times L2\text{.AXI\_WIDTH\_BYTES})
\]

L2.EXTERNAL_READ_UTILIZATION (Derived)
The total utilization of the AXI read interface per cache slice.

\[
L2\text{.EXTERNAL\_READ\_UTILIZATION} = \frac{L2\text{.EXTERNAL\_READ\_BEATS}}{SC\text{.GPU\_ACTIVE}}
\]

L2.EXTERNAL_READSTALL
Increments every cycle that the GPU is unable to issue a new read transaction to AXI because AXI is unable to accept the request.

L2 Read Latency Histogram
The L2 interface implements a six entry histogram that tracks the response latency for the external reads. The counter for the sixth level is synthesized from multiple raw counter values.

<table>
<thead>
<tr>
<th>Histogram range</th>
<th>Counter equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-127 Cycles</td>
<td>L2.EXT_RRESP_0_127</td>
</tr>
<tr>
<td>128-191 Cycles</td>
<td>L2.EXT_RRESP_128_191</td>
</tr>
<tr>
<td>192-255 Cycles</td>
<td>L2.EXT_RRESP_192_255</td>
</tr>
<tr>
<td>256-319 Cycles</td>
<td>L2.EXT_RRESP_256_319</td>
</tr>
<tr>
<td>320-383 Cycles</td>
<td>L2.EXT_RRESP_320_383</td>
</tr>
</tbody>
</table>

L2 Read Outstanding Transaction Histogram
The L2 interface implements a four entry histogram that tracks the outstanding transaction levels for the external reads. The counter for the fourth level is synthesized from multiple raw counter values.

<table>
<thead>
<tr>
<th>Histogram range</th>
<th>Counter equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-25%</td>
<td>L2.EXT_READ_CNT_Q1</td>
</tr>
<tr>
<td>25-50%</td>
<td>L2.EXT_READ_CNT_Q2</td>
</tr>
<tr>
<td>50-75%</td>
<td>L2.EXT_READ_CNT_Q3</td>
</tr>
<tr>
<td>75%-100%</td>
<td>L2.EXTERNAL_READ - L2.EXT_READ_CNT_Q1 - L2.EXT_READ_CNT_Q2 - L2.EXT_READ_CNT_Q3</td>
</tr>
</tbody>
</table>
The following counters profile the external write memory interface behavior:

--- Note ---

This behavior includes traffic from the entire GPU L2 memory subsystem as some types of access bypass the L2 cache.

---

**L2.EXTERNAL_WRITE_BEATS**
Increments on every clock cycle a write beat is read off the external AXI bus.

**L2.EXTERNAL_WRITE_BYTES (Derived)**
Converts the beat counter into a raw bandwidth counter.

\[
L2.EXTERNAL_WRITE_BYTES = \text{SUM}(L2.EXTERNAL_WRITE_BEATS \times L2.AXI_WIDTH_BYTES)
\]

**L2.EXTERNAL_WRITE_UTILIZATION (Derived)**
The total utilization of the AXI write interface per cache slice.

\[
L2.EXTERNAL_WRITE_UTILIZATION = \frac{L2.EXTERNAL_WRITE_BEATS}{SC.GPU_ACTIVE}
\]

**L2.EXTERNAL_WRITESTALL**
Increments every cycle that the GPU is unable to issue a new write transaction to AXI because AXI is unable to accept the request.

**L2 Write Outstanding Transaction Histogram**
The L2 interface implements a four entry histogram that tracks the outstanding transaction levels for the external writes. The counter for the fourth level is synthesized from multiple raw counter values.

<table>
<thead>
<tr>
<th>Histogram range</th>
<th>Counter equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-25%</td>
<td>L2.EXT_WRITE_CNT_Q1</td>
</tr>
<tr>
<td>25-50%</td>
<td>L2.EXT_WRITE_CNT_Q2</td>
</tr>
<tr>
<td>50-75%</td>
<td>L2.EXT_WRITE_CNT_Q3</td>
</tr>
<tr>
<td>75%-100%</td>
<td>(L2.EXTERNAL_WRITE - L2.EXT_WRITE_CNT_Q1 - L2.EXT_WRITE_CNT_Q2 - L2.EXT_WRITE_CNT_Q3)</td>
</tr>
</tbody>
</table>

For more information about these counters, see *Mali Bifrost Family Performance Counters* on Arm Community.
5.9 Event-based sampling

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

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

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

Note

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

Related tasks

5.10 Setting up event-based sampling on page 5-111

Related references

5.5 Counter Configuration dialog box settings on page 5-91
5.10 Setting up event-based sampling

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

To enable event-based sampling, follow these steps:

Procedure
1. Open the Counter Configuration dialog box using the button in the Streamline Data view.
2. Select an event from the Events to Collect list.
3. Click the Toggle event-based sampling button.
   If the Toggle event-based sampling button is not selectable after you have selected an event, then that event does not support event-based sampling.
4. Enter a value in the newly activated Threshold field.
   Avoid setting a very low threshold for high frequency events. If you enter a threshold value that generates too many samples, the capture could fail, and you might have to restart your target. To find an appropriate value to enter in the Threshold field, turn off event-based sampling to run a standard, time-based profile with the event counter that you want to use enabled. Look at the resulting Timeline view and note the peak per-second value in the chart for your counter. Your target for the Threshold field is 1000 samples per second, so if the peak for that event is 2000000, a good value to insert in the Threshold field is 2000.

Results:

Figure 5-4 Setting up event-based sampling
5.11 Statistical Profiling Extension

The Statistical Profiling Extension (SPE) is an optional extension to the Armv8 architecture for profiling software and hardware using randomized sampling.

SPE introduces a set of registers that are specific to the SPE architecture and adds some fields to some Armv8-A System registers in AArch64 state.

Note

SPE is disabled in AArch32 state.

Statistical profiling is a non-invasive debug operation that works as follows:

1. An operation is chosen from a sample population at a programmable interval. Operations are architecture instructions or microarchitectural operations. They appear in the sample population the number of times that they are executed.
2. A trace of the sampled operation is taken.
3. You can filter out potential sample records based on the type of operation, event, or latency, see 5.11.2 SPE counter configuration on page 5-113.
4. A sample record is created that contains the traced information. It is written to and stored in a memory buffer. When the memory buffer is full, software can process the sample records.

All sample records contain:

• A timestamp.
• The context.
• Whether the sampled operation generated an exception.
• Whether the sampled operation completed execution.

If the sampled operation completes execution and does not generate an exception, the sample record also contains:

• The PC virtual address for the sampled operation.
• Whether the sampled operation is a branch, a load, a store, or other.
• Whether the sampled operation is conditional, conditional select, or not.
• The total latency.
• The issue latency.

The architecture defines an extra set of data that is collected in the sample record for each sampled operation:

• Events
• Cycle counters
• Addresses

Further information is recorded for specific types of operations.

You can disable profiling at individual Exception levels, and it is always disabled at EL3.

In a multithreaded implementation, Statistical Profiling is implemented per-thread. The sample interval counter counts only operations for the thread that is being profiled. Latency and other cycle counters count each cycle for the processing element for which the thread was active and could issue an operation.

5.11.1 System requirements for SPE support

When using the SPE support in Arm Streamline, there are some more system requirements.
The SPE system requirements are as follows:

- Use Linux kernel 4.16 or later. To confirm that support is available, check for the path /sys/bus/event_source/devices/arm_spe_XX.
- Disable kernel page table isolation for the target. To ensure that kernel page table isolation is disabled, boot the device with the command-line argument kpti=off.
- For kernel versions 4.20 and later, apply the patch at http://lkml.iu.edu/hypermail/linux/kernel/1903.3/06760.html, or upgrade to a kernel version greater than or equal to 5.1-RC5.
- Use user space gator, not kernel space gator.

### 5.11.2 SPE counter configuration

When using the SPE, the **Counter Configuration** dialog box contains options for filtering the sample records.

![Figure 5-5 SPE counter configuration](image)
Collected sample records can be filtered by type, event, and latency:

- **Type** is any combination of:
  - Loads
  - Stores
  - Branches

- **Event** is any combination of:
  - Architecturally retired instructions.
  - Level 1 cache refilling accesses.
  - Mispredicted branches.
  - TLB missing accesses.

- **Latency** is whether the total latency of the operation exceeds the chosen threshold.

When filtering is enabled, sample records that do not meet the filtering criteria are discarded and are not written to the Profiling Buffer.

**Related references**

- 7.6 *Call Paths* view column headers on page 7-169
- 7.7 *Functions* view column headers on page 7-171

**Related references**

- 5.11.1 System requirements for SPE support on page 5-112
- 5.11.2 SPE counter configuration on page 5-113
- 2.11 *gatord* command-line options on page 2-49
- 7.1 Toolbar options in the table views on page 7-163
- 7.6 *Call Paths* view column headers on page 7-169
- 7.7 *Functions* view column headers on page 7-171
- 8.5 *Code* view toolbar options on page 8-179
Describes the **Live** and **Timeline** views, which display charts showing the data collected during the capture session. **Live** view is displayed while the capture takes place, and charts the data in real time. **Timeline** view is displayed after the capture session ends and the data has been analyzed. It provides additional information in a details panel.

It contains the following sections:

- 6.2 *Timeline view overview* on page 6-120.
- 6.3 *Charts* on page 6-121.
- 6.4 *Chart configuration* on page 6-127.
- 6.5 *Image download* on page 6-135.
- 6.6 *Details panel in the Timeline view* on page 6-138.
- 6.8 *Toolbar options in the Live and Timeline views* on page 6-150.
- 6.9 *Contextual menu options in the Live and Timeline views* on page 6-154.
- 6.10 *Keyboard shortcuts in the Live and Timeline views* on page 6-156.
- 6.11 *Warnings tag* on page 6-157.
- 6.12 *Counter classes* on page 6-158.
- 6.14 *Merged captures* on page 6-161.
6.1 Live view overview

The Live view shows data capture in real time and shows a list of processes alongside usage data. When you trigger a capture session in Arm Streamline, the Live view opens automatically.

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

![Live view screenshot](image-url)

**Figure 6-1  Live view**

The Live view toolbar displays the real-time 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, and the Live view halts scrolling until the data can catch up.

**Note**

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

**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 dies, its name and ID remain in the list but are shown in gray, for example, xaos in the following list:
Downloading process images

To download process images from the target, select the **Download process images from target** checkbox.

When the capture stops, a dialog box is shown where you can select the images to download.

### 6.1.1 Interactive counters panel

During a capture, you can change which events are captured. Using the interactive counters panel, you can add or remove events from the live view.

When capturing starts, the **Target view** shows the **Interactive counters** panel, and the charts panel shows the **Live view** of the selected counters.

- **Add selected event**
  - The selected counter is added to the live view.

- **Remove selected event**
  - The selected counter is removed from the live view.
**Toggle event-based sampling**

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

**Save and restart**

Save the current capture, based on the original counter configuration, and restart the capture with the new configuration.

---

**Note**

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

---

**Restart with new counter configuration**

Discard the current capture and restart with the new counter configuration.

---

**Stop capture and analyze**

Stop capture from the target and analyze the collected data.

---

**Note**

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

---

**Stop capture and discard**

Stop capture from the target and discard the collected data.

Other user indications are:

- 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. This usually happens when charts are added either by applying templates, or using a default chart, and there are not enough counters to allocate to charts.
- 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, 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. Linked categories are indicated by a double arrow in the category heading.

### 6.1.2 Adding and removing events in Live view

Events can be added or removed interactively in **Live view**.

Use the **Interactive counters panel** to add or remove events in **Live view**:

**Adding events to Live view**

- Select an event, then click .
- Double-click the event.
- Select an event, then press Return.
- Drag an event onto the chart panel.
Removing events from Live view

- Select an event, then click ✗.
- Select an event, then press Del.
- Removing 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 by either:

- Click to save the current counter configuration and restart the capture with the new configuration.
- Click to discard the current capture and restart with the new configuration.

Changes to the counter configuration also occur:

- When you edit a series expression, counters are automatically added or removed.
- By applying a template or adding a default chart.

Setting event-based sampling

- To enable or disable event-based sampling, select an event-based sampling capable event and click .
- To toggle event-based sampling, double-click an event.

You can set the reporting threshold in the Threshold field at the bottom of the Interactive counters panel.

Related concepts

5.9 Event-based sampling on page 5-110
6.2 Timeline view overview

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

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

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

**Related concepts**
- 6.3 Charts on page 6-121
- 6.6 Details panel in the Timeline view on page 6-138
6.3 Charts

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

6.3.1 Default charts

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

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

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

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

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

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

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

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

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

*Related concepts*
5.2 Counter Configuration dialog box structure on page 5-87
6.4.1 Chart configuration panel on page 6-127

*Related references*
6.8 Toolbar options in the Live and Timeline views on page 6-150

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

If you have a Mali-based target that is configured to support Mali GPU-specific profiling, and is running a version of gator that supports the target GPU, the Live and Timeline views provide GPU charts that are specific to Mali-based targets.
If you have run the capture session on either a Mali-400 or Mali-450 based target, the following charts are added to the default set of charts:

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

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

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

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

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

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

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

### 6.3.3 Moving and re-sizing charts

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

When you click within a box, it becomes a handle which you can use to drag and drop charts into a preferred order.
To re-order the charts, click and drag the handle control, then release it where you want it placed. To hide a chart, drag it to the bottom of the charts and drag the divider bar up until it is hidden.

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

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

### 6.3.4 Chart disclosure control

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

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

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

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

### 6.3.5 Per-cluster charts

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

For example, CPU Activity and Clock Frequency are per-core counters. In a big.LITTLE system, they are shown in per-cluster charts, with the cluster name shown in the chart title:
Clicking the chart disclosure control for a per-cluster chart displays charts for each core in that cluster only.

**Note**

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

- Version 4.2 or later for Armv7 targets.
- Version 4.4 or later for Armv8 targets.

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

**Related concepts**

6.3.4 Chart disclosure control on page 6-123

### 6.3.6 Quick access tooltips

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

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

### 6.3.7 Cross Section Marker

The Cross Section Marker is a versatile tool for looking at specific ranges of data in the **Live** and **Timeline** views. It starts one bin wide but can be resized using the handles on both sides or moved left and right using the middle of the marker. An overlay shows you data pertinent to the current range covered by the Cross Section Marker.
By default, the Cross Section Marker is inactive. To activate or move it, click anywhere in the charts or in the graphic portion of the details panel in the **Timeline** view. The Cross Section Marker appears where you clicked and provides data specific to the bin where you placed it. The position of the Cross Section Marker in **Live** view is preserved in **Timeline** view.

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

--- **Note** ---

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

---

The Cross Section Marker shows the selected range with a vertical shaded area. When single bin is selected, a single solid vertical line is shown, centered on the selected bin with a shaded range of one bin width. When a time range is selected, vertical lines show the start and end of the range and the selected range is shown shaded.

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

![Cross Section Marker blurred border](image)

**Figure 6-10 Cross Section Marker blurred border**

### 6.3.8 Filtering using the caliper controls

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

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

**Procedure**

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

2. Repeat the process for the right caliper.

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

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

Related references
6.8 Toolbar options in the Live and Timeline views on page 6-150
6.9 Contextual menu options in the Live and Timeline views on page 6-154
6.10 Keyboard shortcuts in the Live and Timeline views on page 6-156
6.4 Chart configuration

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

6.4.1 Chart configuration panel

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

![Chart configuration button](image)

**Figure 6-12  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, in the following figure, defines the chart title as **Bus**, and the chart type as Stacked. The series section defines the **Access (A15)** series.

![Chart configuration panel](image)

**Figure 6-13  Chart configuration panel**

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

6.4.2 Chart configuration toolbar options

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

It has the following options:

- **Create a new series**
  
  Adds an empty series to the chart.

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

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

Figure 6-14 Filled chart

Line

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

Figure 6-15 Line chart

Bar

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

Figure 6-16 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 6-17 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 of the series contained in the chart. For example, if the first value of series A is three and the first value of series B is five, the first data point in the stacked chart that contains these series is eight.

![Figure 6-18 Stacked chart](image)

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

Overlay

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

![Figure 6-19 Data from series A obstructed by data from series B](image)

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

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

**Note**

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

![Logarithmic scale](image)

**Average Selection**

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

**Average Cores**

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

**Percentage**

Select this option to plot values as a percentage of the largest value in the chart.
The following energy offset buttons allow you to align energy data with the other data in the view.

--- Note ---

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

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

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

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

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

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

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

**Related concepts**

- 11.1 Energy Probe overview on page 11-200

### 6.4.3 Chart configuration series options

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

![Series options](Figure 6-21 Series options)

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

Each series in a chart contains the following options:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**Hertz**
Converts the counter to a rate. It takes the value for each time bin and divides it by the unit of time represented by the time bin, then converts it up to seconds. You can use it to convert a cycles count into cycles per second.
Maximum
Works the same way as Minimum, except that it displays a maximum value for each time bin.

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

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

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

Remove Series
Removes the current series from the chart.

Related concepts
6.4.1 Chart configuration panel on page 6-127
6.12 Counter classes on page 6-158

Related references
6.4.2 Chart configuration toolbar options on page 6-127

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

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

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

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

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

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

Figure 6-22  Warning message for a missing counter
Templates are stored by default in the following locations:
• On Windows, in C:\Users<username>\Documents\Streamline\Templates.
• On Linux and Macintosh, in ~/Documents/Streamline/Templates.

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

**Related references**

5.5 Counter Configuration dialog box settings on page 5-91
6.8 Toolbar options in the Live and Timeline views on page 6-150
6.5 Image download

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

——— Note ————

When downloading images from a Linux target over SSH, the credentials supplied for the target device will be stored in a preferences file in your home directory. The file is encrypted using a simple algorithm. If you are concerned about the security of these credentials, do not use this feature.

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

![Figure 6-23  Select Download process images from target.](image)

When you stop the capture, the Image Download dialog opens.
The upper part of this dialog lists the processes that are running on the target. The lower part lists the process images that you have selected for download.

Select a process then click Add Process to add it to the list of images to download. To remove an image from this list, select the image then click Remove.
Add Process and Libs also adds any libraries on which the selected process depends.

To download the selected images, click Start Downloading.

6.5.1 Separate debug files

To limit the size of the executable image file, debug information can be held in a separate file.

Arm Streamline supports the separate debug file linking process that is used in many ELF images.

There are two ways to locate the separate debug file:

- 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.

In both cases, Arm Streamline follows the same process as GDB to locate the debug file.

For more information, on debugging information in separate files, see https://sourceware.org/gdb/onlinedocs/gdb/Separate-Debug-Files.html.
Details panel in the Timeline view

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

Details panel modes

The details panel has the following modes:

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

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

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

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

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

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

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

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

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

Related concepts

6.2 Timeline view overview on page 6-120
6.13 Visual Annotation in the Timeline view on page 6-160

Related references

6.10 Keyboard shortcuts in the Live and Timeline views on page 6-156

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

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

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

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

--- **Note** ---

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

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

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

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

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

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

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

I  Process is idle.

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

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

Note Annotations are not exported.

You can export the Heat Map for any activity source.

Related concepts
6.6.5 Selecting the activity source on page 6-142

Related references
15.4 Exporting the Heat Map from the command-line on page 15-278

6.6.3 Core Map and Cluster Map modes

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

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

6.6.4 Filtering by threads or processes

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

When there is an active selection, the affected chart series show activity that is caused by the selected processes or threads only. Filtered series are identified by the letter F beside the series name. Arm Streamline still displays the total activity in dark gray so that you can visually compare the activity values of the selected processes or threads to the total.
Like the CPU Activity chart, the GPU Vertex and Fragment charts display only activity that is initiated by the selected processes or threads. This allows you to differentiate between GPU activity that is caused by your application and activity resulting from other applications or system services.

### 6.6.5 Selecting the activity source

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

By default, CPU Activity is selected.

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

### 6.6.6 Samples mode

Samples mode in the Timeline view lists all functions in which one or more samples occurred in the time window that is covered by the Cross Section Marker.
The details panel has the following columns when it is in **Samples** mode:

**Function**
- The name of the function.

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

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

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

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

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

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

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

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

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

### 6.6.7 Processes mode

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

<table>
<thead>
<tr>
<th>Index</th>
<th>Process ID</th>
<th>Process Name</th>
<th>% CPU</th>
<th>Memory Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>27711</td>
<td>sshd</td>
<td>0.00%</td>
<td>4.07 MB</td>
</tr>
<tr>
<td>1</td>
<td>init.svinit</td>
<td>0.00%</td>
<td>932.00 KB</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>28139</td>
<td>gobserver</td>
<td>0.00%</td>
<td>1.41 MB</td>
</tr>
<tr>
<td>3</td>
<td>kernel</td>
<td>0.41%</td>
<td>0 B</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1483</td>
<td>rpcbind</td>
<td>0.00%</td>
<td>1.48 MB</td>
</tr>
<tr>
<td>5</td>
<td>28147</td>
<td>gatord</td>
<td>3.01%</td>
<td>3.60 MB</td>
</tr>
<tr>
<td>6</td>
<td>28140</td>
<td>xaos</td>
<td>51.30%</td>
<td>17.55 MB</td>
</tr>
</tbody>
</table>

**Figure 6-33  Processes mode**

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

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

**Process ID**
The PID that Linux assigned to the process.

**Process Name**
The name of the process.

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

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

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

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

### 6.6.8 Images mode

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

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

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

Note

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

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

To enable OpenCL mode, you must create an instrumentation configuration file. For details of the options it must contain, see the documentation that is supplied with the Arm Driver Development Kit (DDK) for Mali Midgard devices.
In OpenCL, commands are added to queues, then execute in parallel on one of the available hardware devices, usually a CPU, GPGPU, or DSP. Commands in a queue execute in series, but commands can also depend on the completion of commands in other queues.

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

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

Arm Streamline allows you to click a command to give it focus, hide any non-relevant information, and show its dependency connections. The following figure shows a selected command that is highlighted with a yellow border. A yellow line shows where it entered the queue.
Zoom out to see more commands and the points at which they were enqueued. Click the **Show all connections** option to display connections for all commands.

![Figure 6-38 Zooming out in OpenCL mode](image)

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

![Figure 6-39 OpenCL mode tooltip](image)

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

![Figure 6-40 Filtering in OpenCL mode](image)

**Related tasks**

2.12.3 Setting up Arm Streamline to support an Arm®Mali™-based device on page 2-52
6.7 Graphics Analyzer Mode in Live view

Switch to Graphics Analyzer mode while profiling an application in Arm Streamline to analyze spikes in graphics activity.

To be able to use Graphics Analyzer mode, install Graphics Analyzer and set up the target with `aga-daemon`. Instructions for setting up the target for Graphics Analyzer are found in the Graphics Analyzer User Guide, which comes with your installation of Graphics Analyzer.

In Live view, click the Analyze in Graphics Analyzer icon in the toolbar to launch Graphics Analyzer from the directory address that is provided in the dialog at Window > Preferences > Streamline > External Tools.

--- Note ---
The functionality that is described here is provided for Graphics Analyzer version 4.0.0.

The following steps then take place:

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

Arm Streamline continues to profile and collect data while the Live view is locked. To continue displaying this data in Live view in Arm Streamline alongside Graphics Analyzer, select the Continue live capture after Graphics Analyzer is activated option in the External Tools dialog. If this option is selected, a bookmark annotation appears in Live view to indicate when the first function is traced by Graphics Analyzer.

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

To continue using Arm Streamline as normal, close the Graphics Analyzer application.

Graphics Analyzer mode has the following limitations:

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

For further information about the Graphics Analyzer, see the Graphics Analyzer pages on Arm Developer,

Related references
6.8 Toolbar options in the Live and Timeline views on page 6-150
17.6 Troubleshooting Graphics Analyzer Mode issues on page 17-298
6.8 Toolbar options in the Live and Timeline views

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

The toolbar controls in the Live and Timeline views are:

**Tags**

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

- ![tag](image) The capture uses event-based sampling.
- ![tag](image) The capture uses user space gator.
- ![tag](image) 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.

**Stop capture and analyze - Live view only**

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

**Stop capture and discard - Live view only**

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

**Toggle bookmark markers**

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

**Center the display on the Cross Section Marker**

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

**Reset Calipers**

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

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

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

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

![Figure 6-41 Zoom level drop-down list](image)

**Time index marker**

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

**Capture duration**

The capture duration in seconds.

**Go to beginning** - *Live view only*

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

**Halt scrolling** - *Live view only*

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

**Go to live position** - *Live view only*

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

**Analyze in Graphics Analyzer** - *Live view only*

Launches the Graphics Analyzer from the address that is specified in the dialog at **Window > Preferences > Streamline > External Tools**, and locks the *Live* view in Arm Streamline. If Graphics Analyzer is already running in another window, Arm Streamline connects to Graphics Analyzer and a new trace begins. If the option to **Continue live capture after Graphics Analyzer is activated** has been selected in the *External Tools* dialog, Arm Streamline continues to display the live capture alongside tracing in Graphics Analyzer. If live capture is continued, a bookmark annotation appears on the *Live* view to indicate when the first function is traced by Graphics Analyzer. Graphics Analyzer retrieves trace details from all graphics API applications running on the target device from the time Graphics Analyzer mode started.
Switch and manage templates

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

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

Add charts with default settings...

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

Cycle through themes

Switches between the dark and light color schemes.

---

**Figure 6-42** Dark color scheme

**Figure 6-43** Light color scheme
Export - *Timeline view only*

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

Another export button below the details panel allows you to Export Heat Map Data. See 6.6.2 Heat Map mode on page 6-138 for further information.

Help

Displays contextual help.

*Related concepts*

6.2 Timeline view overview on page 6-120
6.4.4 Chart configuration templates on page 6-133
6.7 Graphics Analyzer Mode in *Live* view on page 6-148
9.1 Annotate overview on page 9-182

*Related tasks*

6.3.8 Filtering using the caliper controls on page 6-125

*Related references*

6.9 Contextual menu options in the Live and Timeline views on page 6-154
6.10 Keyboard shortcuts in the Live and Timeline views on page 6-156
6.9 Contextual menu options in the Live and Timeline views

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

The contextual menu options are:

Create Bookmark at...

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

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

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

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

Set Left Caliper

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

Set Right Caliper

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

Reset Calipers

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

Set Cross Section Marker To...

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

Set Cross Section Marker Start

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

Set Cross Section Marker End

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

Park the Cross Section Marker

Resets the Cross Section Marker to its default, inactive position.
Related concepts
6.3.7 Cross Section Marker on page 6-124
6.6 Details panel in the Timeline view on page 6-138

Related tasks
6.3.8 Filtering using the caliper controls on page 6-125

Related references
6.8 Toolbar options in the Live and Timeline views on page 6-150
6.10 Keyboard shortcuts in the Live and Timeline views on page 6-156
6.10 Keyboard shortcuts in the Live and Timeline views

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

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

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

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

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

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

**B**
- Toggles the vertical markers for bookmarks on and off.

The following shortcuts apply to the Timeline view only:

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

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

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

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

**Related concepts**
- 6.6.1 Details panel modes on page 6-138
- 6.3.7 Cross Section Marker on page 6-124

**Related references**
- 6.9 Contextual menu options in the Live and Timeline views on page 6-154
- Chapter 10 Log View on page 10-194
6.11 Warnings tag

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

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

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, Arm Streamline was unable to successfully correlate power data from the Energy Probe.

**White**

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

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

Arm Streamline supports the following basic classes of counters:

**Absolute**

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

**Delta**

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

**Incident**

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

**Activity**

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

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

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

--- Note ---

In the delta counter chart, the value of 10 at 4.999ms is amortized from 5ms back to 0ms, because there is no other value, so its value is 2 for that period. The value of 6 at 6.999ms is amortized from 7ms back to 5ms, which is when the last value was received, so its value is 3 for that period.
Figure 6-46 Counter classes

Related references
6.4.3 Chart configuration series options on page 6-131
6.13 Visual Annotation in the Timeline view

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

![Timeline view with Visual Annotation](image)

**Figure 6-47** Visual annotation in the Timeline view

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

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

**Related concepts**
9.3 Visual Annotate overview on page 9-186

**Related references**
9.6 Annotate macros on page 9-190
6.14 Merged captures

You can view merge captures in the Timeline view to allow comparison of multiple program runs.

Merged capture information is shown in the following modes and charts in the Timeline view:

Details panel

In the details panel, the Heat Map, Core Map, and Cluster Map modes show each merged capture and the relative start and end times. Each capture is shown in a row with the name of the original capture, and a green line indicating the time range of the capture. To view the processes and threads, click the disclosure control.

Charts

- Charts that are captured for each core can be viewed by clicking the disclosure control. The activity of each core in each capture is shown as a separate chart.
- System-wide counters can be disclosed to show individual source capture charts.

Related tasks

3.6 Merging captures on page 3-68
Chapter 7
Table Views: Call Paths and Functions

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

It contains the following sections:

• 7.1 Toolbar options in the table views on page 7-163.
• 7.2 Call Paths view contextual menu options on page 7-165.
• 7.3 Functions view contextual menu options on page 7-166.
• 7.4 Keyboard shortcuts in the table views on page 7-167.
• 7.5 Sorting table reports on page 7-168.
• 7.6 Call Paths view column headers on page 7-169.
• 7.7 Functions view column headers on page 7-171.
7.1 Toolbar options in the table views

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

The following toolbar options are available:

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

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

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

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

**Edit Source**

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

**Source**

Opens a drop-down list of the available sources for the counters. Change the source to make the view display the different sets of counters. The list can contain the following sources:

- **Periodic Sampling.** This option is the default.
- **EBS.** This option is similar to **Periodic Sampling** but only shows samples for EBS counters.
- **SPE.** This option shows samples for SPE counters.
- **Python.** This option is available if you use `gator.py` to collect samples from Python scripts.

**Filter**

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

![Filter field](image)

**Export table to a text file**

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

**Show Help**

Opens contextual help.

*Related references*

7.6 Call Paths view column headers on page 7-169

7.7 Functions view column headers on page 7-171
8.5 Code view toolbar options on page 8-179
16.2 Python profiling support on page 16-284
7.2 **Call Paths view contextual menu options**

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

The following menu options are available:

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

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

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

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

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

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

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

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

- **Edit Source**
  Opens the source file for the selected function in your default code editor.
7.3 Functions view contextual menu options

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

The following menu options are available:

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

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

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

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

**Edit Source**
Opens the source file for the selected function in your default code editor.
7.4 Keyboard shortcuts in the table views

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

The following keyboard shortcuts are available for the table views:

**Up arrow**
Moves the current selection up one row.

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

**Down arrow**
Moves the current selection down one row.

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

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

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

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

**Page Down**
Moves down one page.

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

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

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

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

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

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

**Procedure**

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

   ———— Note ————

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

2. Hold down the shift key and click on a different column header.
   **Results:**

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

---

**Figure 7-1 Multi-level sort**

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

——— Note ————
7.6 Call Paths view column headers

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

The Call Paths view has the following column headers:

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

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

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

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

--- Note ---
All data in the Call Paths view depends on the text that is entered in the filter field in the toolbar, and the filtering selection in the Timeline view. If you have used the caliper controls to filter data in the Timeline view, the data in the Call Paths view reflects this selection.

When the source is set to Periodic Sampling, EBS, or Python, the Call Paths view also has the following column headers:

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

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

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

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

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

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, which opens a menu, 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.
### Table Views: Call Paths and Functions

#### 7.6 Call Paths view column headers

<table>
<thead>
<tr>
<th>Call Path Area</th>
<th>Conditioned Instruction</th>
<th>Branch Pn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process/Thread/Code</td>
<td>755</td>
<td>639</td>
</tr>
<tr>
<td>Architecturally retired</td>
<td>469</td>
<td>347</td>
</tr>
<tr>
<td>Level 1 Data Cache</td>
<td>97</td>
<td>25</td>
</tr>
<tr>
<td>Exception generated</td>
<td>76</td>
<td>84</td>
</tr>
<tr>
<td>Total latency</td>
<td>26</td>
<td>34</td>
</tr>
<tr>
<td>Translation latency</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>Level 1 Data Cache Access</td>
<td>25</td>
<td>14</td>
</tr>
<tr>
<td>TLB</td>
<td>38</td>
<td>12</td>
</tr>
<tr>
<td>TLB refill Translation Table walk</td>
<td>14</td>
<td>3</td>
</tr>
</tbody>
</table>

- **Related tasks**
  - 6.3.8 Filtering using the caliper controls on page 6-125

- **Related references**
  - 17.8 Troubleshooting report issues on page 17-301
  - 4.1 Capture & Analysis Options dialog box settings on page 4-77
  - 7.1 Toolbar options in the table views on page 7-163
  - 5.11.2 SPE counter configuration on page 5-113
7.7 Functions view column headers

The Functions view provides a list of all functions that were called during the capture session alongside usage data. It has the following column headers:

Function Name
The name of the function.

Instances
The number of times the function appears in the Call Paths view.

Stack
The number of bytes used by the stack in this function. If the stack usage cannot be precisely determined, the value is colored dark red.

Size
The size of the function in bytes.

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

Image
The image file that contains the function.

Note
All data in the Functions view depends on the text that is entered in the filter field in the toolbar, and the filtering selection in the Timeline view. If you have used the caliper controls to filter data in the Timeline view, the data in the Functions view reflects this selection.

When the source is set to Periodic Sampling, EBS, or Python, the Functions view also has the following column headers:

Self
The number of samples collected in all instances of the function. This figure excludes samples collected in any functions called by this function.

% Self
The Self figure as a percentage of the total number of samples collected in all functions.

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, which opens a menu, 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.
<table>
<thead>
<tr>
<th>Column Headers</th>
<th>Conditional Instruction</th>
<th>Branch Pn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process/Thread/Code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecturally retired</td>
<td>639</td>
<td>191</td>
</tr>
<tr>
<td>Level 1 Data Cache</td>
<td>547</td>
<td>113</td>
</tr>
<tr>
<td>Exception generated</td>
<td>25</td>
<td>19</td>
</tr>
<tr>
<td>Total latency</td>
<td>7</td>
<td>17</td>
</tr>
<tr>
<td>Translation latency</td>
<td>60</td>
<td>63</td>
</tr>
<tr>
<td>Level 1 Data Cache Access</td>
<td>14</td>
<td>73</td>
</tr>
<tr>
<td>TLB</td>
<td>12</td>
<td>26</td>
</tr>
<tr>
<td>TLB refill Translation Table walk</td>
<td>3</td>
<td>37</td>
</tr>
</tbody>
</table>

### Conditional Instruction
- Taken (#/%) | 15 |
- Not Taken (#/%) | 10 |

### Related tasks
- 6.3.8 Filtering using the caliper controls on page 6-125

### Related references
- 7.1 Toolbar options in the table views on page 7-163
- 5.11.2 SPE counter configuration on page 5-113
Chapter 8
Code View

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

It contains the following sections:
• 8.1 Code view basics on page 8-174.
• 8.2 Selecting rows in the Code view on page 8-175.
• 8.3 Path prefix substitution in the Code view on page 8-176.
• 8.4 Using the Find field in the Code view on page 8-178.
• 8.5 Code view toolbar options on page 8-179.
• 8.6 Code view keyboard shortcuts on page 8-180.
8.1 Code view basics

The Code view helps with the discovery of function-level hot spots. It flattens statistics and displays them at the source and disassembly levels.

By default, the Code view shows the source code next to color-coded statistics. To toggle between viewing the source code only and both source and disassembly, click the disassembly view button.

The totals panel displays the number of samples collected in the selected code, and this value as a percentage of the total number of samples collected.

Note
All of the sampling data in the Code view is dependent on the filtering selection in the Timeline view. If you have used the caliper controls to filter data in the Timeline view, the sampling data in the Code view reflects this selection.

Related tasks
6.3.8 Filtering using the caliper controls on page 6-125

Related references
8.5 Code view toolbar options on page 8-179
8.6 Code view keyboard shortcuts on page 8-180
8.2 Selecting rows in the Code view

To select a single row of code in the Code view, simply click on it. If you want the selection to span a range of code lines, you have a few options.

To select multiple rows in the Code view, follow these steps:

Procedure
1. Click on the code row that is the start of your range.
2. There are multiple ways to define the range:
   • Hold the mouse after clicking on the first row and drag it across a range of rows.
   • Hold down the shift key and select the last row of the series to select the entire sequence of rows.
   • Hold down the control key if you want to select additional rows without selecting all of the rows in between.

Results: Selecting code in the Code view highlights related instructions in the disassembly panel.

Note
• The selection behavior in the disassembly section is slightly different than that of the source code section. Selecting a single line of disassembly selects all of the instructions that relate to a single line of source code, and you can use function tags to highlight all instruction lines associated with a single function.
• If selected source code lines or disassembly instructions contain too many rows to fit in the bounds of the current window, small selection indicators appears on the right hand side of the Code view. If there are more selected rows than can fit in the view, the indicators show you how many more are present off screen. Click on the More indicator to see additional selected rows.
8.3 Path prefix substitution in the Code view

Arm Streamline automatically locates and displays the source code in the Code view. If, however, the source files are not located in the same directory they were in during compilation, the Code view is not populated and you must set up path substitutions so that Arm Streamline can find the code.

In cases where Arm Streamline cannot locate the source, it displays a missing source file message in the source section of the Code view.

Follow these steps to set up path prefix substitutions:

Procedure
1. There are two ways to start:
   - Click the link under the missing file message. This displays a standard file dialog which allows you to locate the file. Doing this creates an entry in the Path Prefix Substitutions dialog to appropriately map the file.
   - Click the Set Path Prefix Substitutions button in the toolbar. This opens the Path Prefix Substitutions dialog box.

Results:
2. Click the plus symbol to add a new path prefix substitution.
3. Enter valid paths in the Prefix and Replacement fields.
4. Click the **Apply** button.

**Results:** If the path given in the Replacement field contains code that lines up with the code used in the capture, Arm Streamline populates the view with your source code and the statistical overlay.
8.4 Using the Find field in the Code view

To search your code and instructions for a function name or an instruction address, use the Find field, located just below the toolbar in the Code view.

![Figure 8-4 Find field]

To find a specific function or hexadecimal instruction address, follow these steps:

Procedure
1. Enter a string in the Find field.
   Results: The box to the right of the Find field updates to show any current matches.
2. Press the Enter key to go to the first match in the code.
3. Press the Enter key again to cycle through all available matches.

Related concepts
8.1 Code view basics on page 8-174
8.5 Code view toolbar options

The toolbar of the Code view contains the following buttons:

**Disassembly View**
- Opens the disassembly panel. The disassembly panel takes up the bottom section of the Code view and shows the assembly language instructions that are associated with the source code.

**Recently Selected Functions**
- Opens a drop-down list that enables you to choose a recently selected function from a list. This selects the function in the Code view.

**Edit Source**
- Opens the source file in your preferred editor.

**Set Path Prefix Substitutions**
- Opens the Path Prefix Substitutions dialog box. This enables you to set up path substitutions so that Arm Streamline can find code that is not located in the same directory that it was in during compilation.

**Source**
- Select the source from the first drop-down list, then select the counter to view from the second drop-down list. The first list can contain the following sources:
  - **Periodic Sampling.** This option is the default.
  - **EBS.** This option is similar to Periodic Sampling but only shows samples for EBS counters.
  - **SPE.** This option shows samples for SPE counters.
  - **Python.** This option is available if you use gator.py to collect samples from Python scripts. When using this option, the disassembly view is empty.

**Help**
- Opens contextual help.

Related concepts
- 8.1 Code view basics on page 8-174

Related tasks
- 8.3 Path prefix substitution in the Code view on page 8-176

Related references
- 5.11.2 SPE counter configuration on page 5-113
- 7.1 Toolbar options in the table views on page 7-163
- 16.2 Python profiling support on page 16-284
8.6 Code view keyboard shortcuts

The keyboard shortcuts in the Code view help you to quickly navigate through your source code and disassembly instructions.

The keyboard shortcuts available for the table views are:

**Up arrow**
Moves the current selection up one row.

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

**Down arrow**
Moves the current selection down one row.

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

**Home**
Takes you to the top of the function that contains the currently selected row. If a line of code is selected in the source view that does not have any instructions associated with it, the home key takes you top of the source file.

**End**
Takes you to the bottom of the function that contains the currently selected row. Like the home key, if the selected line of source does not have any instructions associated with it, the end key takes you to the bottom of the file.

**Page Up**
Moves up one page. A page is defined by the range of rows currently displayed in either the source or disassembly view.

**Page Down**
Moves down one page.

Related concepts

8.1 Code view basics on page 8-174
Chapter 9
Code Annotations

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

It contains the following sections:

• 9.1 Annotate overview on page 9-182.
• 9.2 Adding string annotations to your code on page 9-185.
• 9.3 Visual Annotate overview on page 9-186.
• 9.4 Adding Visual Annotate to your code on page 9-188.
• 9.5 Kernel annotations on page 9-189.
• 9.6 Annotate macros on page 9-190.
• 9.7 Importing the Streamline_annotate example on page 9-193.
9.1 Annotate overview

While Arm Streamline provides a large variety of target information, sometimes you might require extra context. Arm Streamline lets you instrument your source code by adding annotations to it.

When the user space application writes to the gator annotate socket, the gator driver integrates the recorded annotate-driven output into the Arm Streamline sample and trace capture report. The annotated text is marked with a thread identifier, which keeps the data uncluttered and eliminates the need for user mutexes.

You can add the following types of annotations:

--- Note ---

Build the examples in `DS_install_directory/sw/streamline/examples/annotations/` as follows:
- Build natively on an Arm target by running `make`.
- Cross compile by running `make CROSS_COMPILE=filename`. The compiled binaries appear in a folder called `bin`.
- Build for Android by running `ndk-build`.

---

String

String annotations work in a similar way to `printf()` statements. However, instead of console output, they populate the Log view and place framing overlays directly in the Timeline view.

See `DS_install_directory/sw/streamline/examples/annotations/text.c` for examples of how to use some of the string annotation macros.

Visual

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

See `DS_install_directory/sw/streamline/examples/annotations/visual.c` for an example of how to use the `ANNOTATE_VISUAL()` macro.

Marker

Marker annotations add bookmarks to the Timeline and Log views, optionally with a text string, to identify time points of interest.

See `DS_install_directory/sw/streamline/examples/annotations/text.c` for examples of how to use the various marker annotation macros.

Custom counters

These annotations are counters that you dynamically define in user space code. After you have run a capture session, the Timeline view displays a chart for each custom counter you have created, showing the numeric values plotted over time.

See `DS_install_directory/sw/streamline/examples/annotations/absolute.c` for examples of how to use custom absolute counters and `DS_install_directory/sw/streamline/examples/annotations/delta.c` for examples of how to use custom delta counters.
Groups and channels

This set of macros enables you to break down threads into multiple channels of activity and assign each channel to a group. A number identifies each group and channel, and each one has a name for display in the Timeline and Log views. Groups and channels are defined per thread. Therefore although each channel number must be unique within the thread, channels in different threads can have the same number.

See DS_install_directory/sw/streamline/examples/annotations/text.c for examples of how to use the group and channel annotation macros.

Custom activity maps

These annotations allow you to define and visualize a complex dependency chain of jobs. Each custom activity map (CAM) view contains one or more tracks and each track contains one or more jobs.

Jobs might have dependencies on other jobs. Dependencies between jobs are shown using connecting lines with circles at each end to indicate the direction of the dependency. A job with a closed circle depends on a job with an open circle.

See DS_install_directory/sw/streamline/examples/annotations/cam.c for examples of how to use the CAM annotation macros.

![Custom Activity Maps](image)

Annotation macros are defined in streamlineannotate.h. StreamlineAnnotate.java provides equivalent methods for string, visual, and marker annotations, and for groups and channels.

In addition to the basic reference examples for each annotation type in DS_install_directory/sw/streamline/examples/annotations/, some of the included Linux examples use annotations. Streamline.annotate uses string, visual, and marker annotations, and groups and channels, and xaos uses string, visual, and marker annotations. Both are located in DS_install_directory/sw/install_directory/examples/Linux_examples.zip. To use these examples, import the Linux application example projects into Arm Development Studio IDE. Refer to the readme.html for each example for more details.

**Note**

In DS-5 version 5.20 and later, applications that use user space gator, in addition to ones that use kernel space gator, can emit annotations. Applications that are built using the annotation implementation in earlier versions of DS-5 continue to work in version 5.20 and later. However they only work with kernel space gator. If they use user space gator, you must rebuild them using the new annotation implementation.

**Related tasks**

- 9.2 Adding string annotations to your code on page 9-185
- 9.7 Importing the Streamline_annotate example on page 9-193
Related references

9.6 Annotate macros on page 9-190
9.2 Adding string annotations to your code

String annotations work in a similar way to `printf()` statements but instead of console output, they populate the Log view and place framing overlays directly in the Timeline view.

To add string annotations to your code, follow these steps:

**Procedure**

1. Include `streamline_annotate.h` and `streamline_annotate.c` in your project. Alternatively, build a `libstreamline_annotate` library with the Makefile in `DS_install_directory/sw/streamline/gator/annotate`, and include this library in your project.

   If you are working in Java, `StreamlineAnnotate.java` provides the same functionality as the macros in `streamline_annotate.h` for string, visual, and marker annotations, and for groups and channels.

2. Add the `ANNOTATE_SETUP` macro to your code. You must call this before any other annotate macros.

3. Add annotations to your code using the macros defined in `streamline_annotate.h`. For example, write a null-terminated string from any thread and set its color using the `ANNOTATE_COLOR(color, string)` macro. Either choose a color constant from those defined in `streamline_annotate.h` or send the ASCII escape code followed by a 3-byte RGB value.

4. Optional: Use `ANNOTATE_END()` to clear the annotation message for the thread.

5. Ensure `gatord` is running.

6. Run your application and exercise the area of the code that emits the annotations.

String annotations are displayed as text overlays inside the relevant channels in the details panel of the Timeline view, for example inside Channel 0 in the following screenshot. The letter A is displayed in the process list to indicate the presence of annotations. String annotations are also displayed in the Message column in the Log view.

![String annotation overlays](image)

**Figure 9-2** String annotation overlays

**Related concepts**
9.1 Annotate overview on page 9-182

**Related tasks**
9.7 Importing the Streamline_annotate example on page 9-193

**Related references**
9.6 Annotate macros on page 9-190
9.3 Visual Annotate overview

In addition to simple text annotations, Arm Streamline supports annotations that contain images, providing further application-level context to the **Timeline** view.

In the same way as string annotations, the application writes to the gator annotate socket and Arm Streamline integrates the image data and its timestamp into the sample and trace capture report.

Visual annotations are displayed in the visual annotation chart, shown in the following screenshot:

![Visual Annotate in the Timeline view](image)

To include images in the data sent to the host during a capture session, use the `ANNOTATE_VISUAL()` macro in your source code. `ANNOTATE_VISUAL()` provides a parameter for image data.

Visual Annotate supports images in the following formats:

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

There is no limit to the image size but the larger the image, the greater the impact on system performance. Increasing the amount of data sent to the host in this way increases the probe effect for the applications you are profiling.

The following example function is from the `Streamline_annotate` example project, included in the `Linux_examples` archive in the examples directory of your Arm Development Studio installation:

```c
void displayImage() {
  char filename[32];
  char* image;
  unsigned int size;

  // Supported formats include gif, png, jpeg, tiff, ico, bmp (+rle)
  strcpy(filename, "splash.bmp");
  image = readFromDisk(filename, &size);
  if (image == NULL) {
    printf("error loading image %s\n", filename);
    exit(1);
  }

  // Add text along with the image annotation
  ANNOTATE_VISUAL(image, size, filename);
  free(image);
}
```
You can see the effects of Visual Annotate in the **Timeline** and **Log** views of your Arm Streamline Analysis Reports. The **Timeline** view includes a chart that displays the images. In the **Log** view, any annotation event that includes an image has a camera icon in the message field. Select a row containing a camera icon to see the image.

**Note**

A Mali GPU automatically emits visual annotations when you select a Filmstrip counter.

*Related concepts*

6.13 **Visual Annotation in the Timeline view** on page 6-160

*Related tasks*

2.12.3 **Setting up Arm Streamline to support an Arm®-Mali™-based device** on page 2-52

*Related references*

9.6 **Annotate macros** on page 9-190
9.4 Adding Visual Annotate to your code

Arm Streamline provides a set of macros that enable you to add images to the Timeline and Log views in your generated Analysis Reports.

To add visual annotations to your code, follow these steps:

**Procedure**

1. Include the files `streamline_annotate.h` and `streamline_annotate.c` in your project. Alternatively, build a `libstreamline_annotate` library with the Makefile in `DS_install_directory/sw/streamline/gator/annotate`, and include this library in your project.
2. Add the `ANNOTATE_SETUP` macro to your code. It must be called before any other annotate macros are called.
3. Insert the `ANNOTATE_VISUAL(data, length, str)` macro into your code. Replace `data` with your image, `length` with the size of the data being written to the annotate file, and, optionally, `str` with a descriptive string to be included with the image.
4. Build your code.
5. Ensure that `gatord` is running then run your application and exercise the area of the code that emits the annotations.
9.5 Kernel annotations

You can insert annotation macros in either user space code or kernel space code. There are a few important considerations when using them in kernel space code or in a module.

——— Note ————

As the annotation macros might block, Arm recommends that you do not add them to kernel space code in an interrupt context. For more information, see gator_annotate.c in the gator driver source code.

———

For string, visual, and marker annotations, and channels and groups, the same macros are defined for both user space and kernel space using the _KERNEL_ preprocessor conditional. Insert these macros in kernel space code or in a module in the same way you would in user space code, with the following exceptions:

• Do not call ANNOTATE_SETUP in kernel space code. This macro is a prerequisite for other annotation macros in user space code, but not in kernel space code.
• You do not need to include streamline_annotate.c in the project.

You can only use the custom counter macros and custom activity map macros in user space code. See the gator_events_mmapped.c example in the gator driver source code for an example of adding custom counters to kernel space code.

Related tasks
12.3 Using the gator_events_mmapped.c custom counters example on page 12-215

Related references
9.6 Annotate macros on page 9-190
9.6 Annotate macros

Arm Streamline provides a variety of macros to mark up your code, whether you want color markup, macros that include channel information, or macros that insert bookmarks into the Timeline and Log views.

These macros are defined in `DS_install_directory/sw/streamline/gator/annotate/streamline_annotate.h`.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANNOTATE_DEFINE</td>
<td>You do not need to call <code>ANNOTATE_DEFINE</code>. It is defined in <code>streamline_annotate.h</code> to avoid compilation errors in legacy code.</td>
</tr>
<tr>
<td>ANNOTATE_SETUP</td>
<td>Call this macro to set up annotation, before calling any other annotate macros.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>When you annotate within the kernel or a module, do not use <code>ANNOTATE_SETUP</code>.</td>
</tr>
<tr>
<td>ANNOTATE(string)</td>
<td>Adds a string annotation. This macro does not define a specific channel, so the annotation is added to Channel 0 by default.</td>
</tr>
<tr>
<td>ANNOTATE_CHANNEL(channel, string)</td>
<td>Adds a string annotation to a channel defined by the numeric identifier passed in the channel parameter.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>Annotation channels and groups are used to organize annotations within the threads and processes section of the Timeline view. Each annotation channel appears in its own row under the thread. Channels can also be grouped and displayed under a group name, using the <code>ANNOTATE_NAME_GROUP</code> macro.</td>
</tr>
<tr>
<td>ANNOTATE_COLOR(color, string)</td>
<td>Works in the same way as the basic <code>ANNOTATE</code> macro, except the color parameter defines an interface display color for the annotation string. See <code>streamline_annotate.h</code> for the defined colors.</td>
</tr>
<tr>
<td>ANNOTATE_CHANNEL_COLOR(channel, color, string)</td>
<td>Defines an annotation string with a display color, and assigns it to a channel.</td>
</tr>
<tr>
<td>ANNOTATE_END()</td>
<td>Terminates the annotation. Because the <code>ANNOTATE_END</code> macro does not define a specific channel, it defaults to channel 0.</td>
</tr>
<tr>
<td>ANNOTATE_CHANNEL_END(channel)</td>
<td>Terminates the annotation for the given channel.</td>
</tr>
<tr>
<td>ANNOTATE_NAME_CHANNEL(channel, group, string)</td>
<td>Defines a channel and attaches it to an existing group. The channel number must be unique within the thread.</td>
</tr>
<tr>
<td>ANNOTATE_NAME_GROUP(group, string)</td>
<td>Defines an annotation group. The group identifier, group, must be unique within the thread.</td>
</tr>
<tr>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ANNOTATE_VISUAL(data, length, str)</td>
<td>Records an annotation that includes an image in one of the following formats: GIF, PNG, JPEG, TIFF, ICO, BMP +RLE. Specify the image in <code>data</code>, the amount of data being written to the annotate file in <code>length</code>, and optionally a descriptive string to be included with the image in <code>str</code>.</td>
</tr>
<tr>
<td>ANNOTATE_MARKER()</td>
<td>Adds a default bookmark to the Timeline and Log views.</td>
</tr>
<tr>
<td>ANNOTATE_MARKER_STR(string)</td>
<td>Adds a bookmark with a string to the Timeline and Log views. The string is displayed in the Timeline view when you hover over the bookmark and in the Message column in the Log view.</td>
</tr>
<tr>
<td>ANNOTATE_MARKER_COLOR(color)</td>
<td>Adds a bookmark with a color to the Timeline and Log views.</td>
</tr>
<tr>
<td>ANNOTATE_MARKER_COLOR_STR(color, string)</td>
<td>Adds a bookmark with a string and a color. The bookmark appears in the Timeline and Log views.</td>
</tr>
<tr>
<td>ANNOTATE_DELTA_COUNTER(id, title, name)</td>
<td>Defines a custom delta counter. Specify an integer in <code>id</code> to uniquely identify the counter, and a string for the chart title in <code>title</code> and for the series name in <code>name</code>. Use this macro in user space code only.</td>
</tr>
<tr>
<td>ANNOTATE_ABSOLUTE_COUNTER(id, title, name)</td>
<td>Defines a custom absolute counter. The parameters have the same meanings as for ANNOTATE_DELTA_COUNTER. Use this macro in user space code only.</td>
</tr>
<tr>
<td>ANNOTATE_COUNTER_VALUE(id, value)</td>
<td>Emits a value for a custom delta or absolute counter. Identify the counter using <code>id</code> and specify the integer value using <code>value</code>. Use this macro in user space code only.</td>
</tr>
<tr>
<td>CAM_TRACK(view_uid, track_uid, parent_track, name)</td>
<td>Adds a track to a CAM view. To make the track a child of another track, specify the parent track ID in <code>parent_track</code>, or specify -1 to make it a root track. <code>track_uid</code> must be unique within the view.</td>
</tr>
<tr>
<td>CAM_JOB(view_uid, job_uid, name, track, start_time, duration, color)</td>
<td>Adds a job with a start time and duration to a CAM track. <code>job_uid</code> must be unique within the view. The job is displayed in the Timeline view as a colored bar representing its duration. The job name is displayed inside the bar. The start time and duration of the job are specified in nanoseconds. Use <code>gator_get_time()</code>, declared in streamline_annotate.h to get the current timestamp in nanoseconds.</td>
</tr>
<tr>
<td>CAM_JOB_DEP(view_uid, job_uid, time, dependency)</td>
<td>The single-dependency version of CAM_JOB_DEPS.</td>
</tr>
<tr>
<td>CAM_JOB_DEPS(view_uid, job_uid, name, track, start_time, duration, color, dependency_count, dependencies)</td>
<td>Adds a job with dependencies on other jobs to a CAM track. <code>track</code> is the track to add this job to, <code>dependencies</code> are the jobs that this job depends on, and <code>dependency_count</code> is the number of dependencies.</td>
</tr>
<tr>
<td>CAM_JOB_START(view_uid, job_uid, name, track, time, color)</td>
<td>Adds a job with a start time to a CAM track. End the job using CAM_JOB_STOP.</td>
</tr>
<tr>
<td>CAM_JOB_SET_DEP(view_uid, job_uid, time, dependency)</td>
<td>The single-dependency version of CAM_JOB_SET_DEPS.</td>
</tr>
<tr>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CAM_JOB_SET_DEPS(view_uid, job_uid,</td>
<td>Sets the dependencies of a job that was previously started using CAM_JOB_START. If you call this macro multiple times, Arm Streamline uses</td>
</tr>
<tr>
<td>time, dependency_count, dependencies)</td>
<td>the dependencies with the latest timestamp.</td>
</tr>
<tr>
<td>CAM_JOB_STOP(view_uid, job_uid, time)</td>
<td>Ends a CAM job that was previously started using CAM_JOB_START.</td>
</tr>
<tr>
<td>CAM_VIEW_NAME(view_uid, name)</td>
<td>Creates a named Custom Activity Map (CAM) view. view_uid must be unique.</td>
</tr>
</tbody>
</table>

**Related concepts**
- 9.1 Annotate overview on page 9-182
- 9.5 Kernel annotations on page 9-189

**Related tasks**
- 9.2 Adding string annotations to your code on page 9-185
- 9.7 Importing the Streamline_annotate example on page 9-193
9.7 Importing the Streamline_annotate example

The best way to get started with Streamline Annotate is to look at the Streamline_annotate example. It contains examples of text, visual, and bookmark annotations, as well as groups and channels.

To import the Streamline_annotate example into Eclipse for Arm Development Studio, follow these steps.

**Procedure**
1. Select **File > Import...** in Eclipse for Arm Development Studio.
2. Use the **Disclosure control** to open the **General Tab** in the **Import dialog** box.
3. Select **Existing Projects into Workspace**.
4. Click **Next**.
5. Use the radio button to activate the **Select archive file** field.
6. Click the **Browse** button next to the **Select archive file** field.
7. Navigate to `../examples` in your Arm Development Studio installation directory.
8. Select the **linux_examples** archive.
9. Click **Open**.
10. Make sure the Streamline_annotate example is checked. Select any other projects that you want to import.
11. Click **Finish**.

**Results:** When completed, the Streamline_annotate example appears in your **Project Explorer** view along with any other examples you imported. See the readme.html for more information about the example.

**Related concepts**
- **9.1 Annotate overview on page 9-182**

**Related tasks**
- **9.2 Adding string annotations to your code on page 9-185**

**Related references**
- **9.6 Annotate macros on page 9-190**
Chapter 10
Log View

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

It contains the following sections:
• 10.1 Log view column headers on page 10-195.
• 10.2 Log view filter fields and toolbar options on page 10-197.
• 10.3 Log view contextual menu options on page 10-198.
10.1 Log view column headers

The Log view lists all annotations, except CAM annotations, that were generated during your capture session.

Note

To populate the Log view, insert ANNOTATE statements in your code or add bookmarks to the capture in Live view.

The Log view contains the following column headers:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>When</td>
<td>The number of seconds since the start of the capture session when the message was generated. All messages appear in the Log view in chronological order.</td>
</tr>
<tr>
<td>Duration</td>
<td>The duration in seconds of the annotation. For bookmark annotations, this value is zero.</td>
</tr>
<tr>
<td>Message</td>
<td>The contents of the annotation message. For bookmark annotations, whether they are generated on the target by an annotation or added manually in Live view, a bookmark icon precedes the message text. For visual annotations, a camera icon is displayed. To see the image, select a row with a camera icon.</td>
</tr>
<tr>
<td>Group</td>
<td>The name of the group to which the annotation channel belongs. Group 0 is displayed if the channel does not belong to a group.</td>
</tr>
<tr>
<td>Channel</td>
<td>The name of the channel to which the annotation belongs. Channel 0 is displayed if the annotation does not belong to a channel.</td>
</tr>
<tr>
<td>Where</td>
<td>The process and thread that generated the message.</td>
</tr>
</tbody>
</table>

![Figure 10-1 Annotations in the Log view](image)
Note

Unlike the other table reports in Arm Streamline, you cannot sort the data in the Log view.

Related concepts
9.1 Annotate overview on page 9-182

Related references
9.6 Annotate macros on page 9-190
10.2 Log view filter fields and toolbar options

The Log view provides several filter fields above the table data which enable you to filter the messages displayed.

You can enter a regular expression in any of the filter fields listed below, except the When and Duration fields. Only messages that match the pattern appear in the list, sorted in chronological order. The Log view contains the following filter fields:

When
Filters the Log view based on when Arm Streamline captured the annotation. To define a range, use a dash to separate the lower and upper limits. For example, 4-5 filters the Log view to only show annotations captured between seconds 4 and 5 inclusive. To define a range up to a certain number, enter a dash and then a single number. To define a range that includes a number and everything above it, enter a number followed by a dash. For example, to filter for annotations that occurred at the thirty second mark or later in the capture, enter 30-.

Duration
Filters the Log view based on the duration of the annotation. As with the When field, you can enter a number or a range of numbers.

Message
Shows only messages that match the regex.

Group
Shows only messages that belong to a particular group.

Channel
Shows only messages assigned to a particular channel.

Where
Filters the view based on the name of the process or thread that logged the message.

In addition, a totals panel gives additional information when you select multiple messages in the view. It contains the following fields:

Log Entries
The total number of entries you have selected in the Log view.

Delta
The time difference in seconds between the selected entries that were logged first and last. To select multiple entries, click on one, then hold down the Shift or Ctrl key and click on another.

Figure 10-2 Log view totals panel

The Log view has the following toolbar options:

Export table to a text file
Opens the Export dialog box, which enables you to export the data from the Log view to a text file.

Show Help
Opens contextual help.
10.3 Log view contextual menu options

Right-click anywhere in the table to open a contextual menu that provides you with options to navigate to selected annotations in either the Timeline view or the Call Paths view.

The menu contains the following options:

Select Time Range in Timeline
  Opens the Timeline view with the Cross Section Marker set to include all of the selected entries.

Select Process/Thread in Timeline
  Opens the Timeline view with the Cross Section Marker moved to the location of the selected annotation message.

Select in Call Paths
  Opens the Call Paths view. All functions related to the selection in the Log view are selected in the Call Paths view.

Related concepts
6.3.7 Cross Section Marker on page 6-124

Related references
7.6 Call Paths view column headers on page 7-169
Chapter 11
Capturing Energy Data

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

It contains the following sections:

• 11.1 Energy Probe overview on page 11-200.
• 11.2 Energy Probe requirements on page 11-202.
• 11.3 Shunt resistor selection for Energy Probe on page 11-203.
• 11.4 Setting up Energy Probe on page 11-204.
• 11.5 Adding the catman application to Arm Streamline on page 11-206.
• 11.6 Updating your firmware on page 11-207.
• 11.7 Energy Probe data in Arm Streamline on page 11-208.
• 11.8 Setting up National Instrument Multifunction Data Acquisition devices (NI DAQ) to capture energy data on page 11-209.
11.1 **Energy Probe overview**

To capture energy data for Arm Streamline you must have either an Energy Probe or a supported NI DAQ device.

The Energy Probe is designed to be a low impact, inexpensive solution to give application and system software developers quick feedback on the impact of their code on the system energy footprint. It is intended to provide a better understanding of the static and dynamic behavior of the target system for the purposes of debugging, profiling, and analysis. It is not intended to be a high-precision data acquisition instrument for power benchmarking.

The Energy Probe has three power connectors, each of which is designed to connect to a 2-pin header for measuring the power. This allows the energy probe to provide three independent power, current, or voltage measurements.

![Energy Probe schematic](image1)

*Figure 11-1  Energy Probe schematic*

In addition to the three power connectors, the Energy Probe has a single pin GND connector that must be connected to ground on your target board. It provides a ground connection for the Energy Probe.

Each of the three power connectors measures:
- The current flow through a shunt resistor of a known value on your target system
- The voltage at the positive terminals of the Energy Probe.

![Energy Probe electrical connection example](image2)

*Figure 11-2  Energy Probe electrical connection example*
Caution

The Energy Probe has flying leads and must be carefully connected to your target. Do not plug the green ground wire into anything but the target ground. This includes I/O pins and power pins. Doing so could cause damage to your Energy Probe or the power supply of your target.

Related concepts
11.7 Energy Probe data in Arm Streamline on page 11-208

Related tasks
11.4 Setting up Energy Probe on page 11-204
11.5 Adding the caiman application to Arm Streamline on page 11-206

Related references
11.2 Energy Probe requirements on page 11-202
11.2 Energy Probe requirements

Using the Energy Probe with Arm Streamline requires the Energy Probe hardware, Arm Development Studio installed on your host, and a target with a shunt resistor that meets the specifications required by the Energy Probe.

The Energy Probe has the following requirements:
• An installation of Arm Development Studio.
• A suitable Arm Development Studio license.
• A target running Linux kernel version 3.4 or later, configured for Arm Streamline profiling.
• An Energy Probe unit.
• A USB extension cable.
• USB drivers for the Energy Probe.
• A target that has 2-pin IDC 0.1" power measurement headers. The target also requires a shunt resistor with a supply voltage less than 15V and rated at least 0.5W. The shunt resistor needs a 1-pin IDC ground terminal and must not drop more than 165mV.

——— Note ———
You can tell if your target has the right power management headers by visual inspection. For more information about whether or not your target meets the other requirements for Energy Probe, see the documentation for your target.

Related references
2.1 Arm Streamline prerequisites on page 2-31
11.3 **Shunt resistor selection for Energy Probe**

With 20x amplification, Energy Probe requires the correct selection of a shunt resistor to provide the best possible dynamic range in power measurement, while avoiding saturation of the input of Energy Probe.

A shunt resistor with a value that is too low reduces measurement dynamic range, resulting in less resolution in the power data. A shunt resistor with a value that is too high can cause the input stage of the Energy Probe to saturate, which causes a flat line in the charts that are related to Energy Probe in the Timeline view.

To avoid input saturation, the drop across the shunt resistor must never be more than 165mV. You can also use the following equation to determine whether your shunt resistor is appropriate:

\[
R_{\text{shunt(max)}} = \frac{165 \times V_{\text{supply}}}{1000 \times \text{Power}}
\]

\(V_{\text{supply}}\) is the input/core voltage. \(\text{Power(max)}\) is the maximum power that the Energy Probe measures. \(R_{\text{shunt(max)}}\) is the maximum value of the shunt resistor. A shunt resistor value that is greater than \(R_{\text{shunt(max)}}\) might cause input saturation.

This equation provides the absolute maximum value for the shunt resistor. Use a value that is more than five percent lower than this to allow for component tolerances.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
</table>

When connecting the Energy Probe, consider the following:
- The black and white probe closest to the green wire is Channel 0.
- For best results, attach Channel 0 to the power source which best represents the CPU load. Streamline aligns the power data with the software activity by maximizing the correlation of Channel 0 with the CPU load.
- The probe white wire is V+. The black wire is V-.

<table>
<thead>
<tr>
<th>Examples</th>
</tr>
</thead>
</table>
- 5V power supply, 8W(max), \(R_{\text{shunt(max)}} = 100\) milliohms.
- 1V core voltage, 2.5W(max), \(R_{\text{shunt(max)}} = 50\) milliohms.
- 1.5V core voltage, 0.4W(max), \(R_{\text{shunt(max)}} = 500\) milliohms.
11.4 Setting up Energy Probe

It is critical that you set up your Energy Probe correctly. Failure to do so can result in missing power data in Streamline, and, in some cases, damage to your target.

![Connection to target](image)

To set up Energy Probe, follow these steps:

**Procedure**

1. Connect your target to your host through a USB port.
2. If necessary, install drivers using the standard Windows driver installation.
   When first plugging in the Energy Probe on Windows, the LED flashes RED at approximately one second intervals. This indicates the USB enumeration has failed and that you must install the driver.
3. If prompted to locate the driver file, select the `ARM_EnergyProbe.inf` configuration file, which is located in `DS_install_directory/sw/streamline/energy_meter/energy_probe/`.
   **Results:** When you have installed the drivers, and Energy Probe is operating correctly, the LED remains green.
4. To set it up on Linux Ubuntu, enter the following command:
   ```bash
   sudo apt-get install libudev-dev
   ```
5. On Linux Ubuntu, you must do either of the following for Energy Probe to successfully use the device:
   - Change the permissions of the tty device. To do this, enter the following command:
chmod 777 tty_location

• Add yourself to the same group as the tty device. To do this, enter the following command:

    usermod -a -G Group_Name User_Name

dialout is the usual name of the group for the tty device.

Results: If you attach a probe to the target with the channel wire connector the wrong way around, this causes the LED to turn red and blink. This does not damage your target or the Energy Probe but it does not provide useful data. If this happens, disconnect all Energy Probes and wait until the LED changes to green, then reattach them one by one and make sure you reverse any probe that caused the LED to change to red. When you have attached the Energy Probe correctly, the LED remains green.

    Note

In case of low consumption, it is possible that the LED remains red even if the connection is correct. This might happen if the target device is idle.

6. If your target has headers available for power or energy metering, attach the ground wire, then attach the probes.

7. Attach the probes.

    Make sure that the probe polarity is correct by observing whether the LED changes from green to red. The probe white wire is V+. The black wire is V-.
11.5 Adding the caiman application to Arm Streamline

For Arm Streamline to communicate with the Energy Probe, you must give the location of the caiman application using the Tool Path field in the Capture & Analysis Options dialog box.

To configure Arm Streamline for the Energy Probe, follow these steps:

Procedure
1. Select the ARM Energy Probe option from the Energy Capture drop-down menu.
2. Click the Select the energy capture tool button in the Tool Path field.
3. Locate caiman.exe in your Arm Streamline installation directory, select it, and click Open.

Results:

![Figure 11-4 Tool Path field](image)

Note

- It might be necessary to specify the tty device when using the Arm Energy Probe on Linux. To do so, enter the following in the Device field: /dev/ttyACM0. When using NI DAQ, the device name is usually Dev1.
- On Windows, you can leave the Device field empty. Arm Streamline auto-detects your energy probe.
11.6 Updating your firmware

Necessary updates occasionally become available for the Energy Probe firmware.

Follow these instructions to update your firmware:

Procedure

1. Detach the Energy Probe probes and ground from the target.
2. Plug the Energy Probe into a USB port on your Windows host machine.
   
   Because of a limitation with the USB bootloader from the MCU vendor, updating firmware on a Linux host is not supported.
3. To enable firmware programming mode, insert a paperclip into the small hole on the top near the LED.
4. Press and hold down the button for three seconds.
5. Wait for the LED to turn red, and then off. Firmware programming mode is now active.
6. When the drive folder opens, delete the firmware.bin file. This file is a placeholder.
7. Drag and drop the new emeter_firmware.bin file onto the drive folder. This file is located in DS_install_directory/sw/streamline/energy_meter/energy_probe/.
8. Safely remove the Energy Probe drive from the host machine using the Windows Safely Remove Hardware icon.
9. Unplug the Energy Probe from the host machine.
10. Wait for one second and plug it back in.

Related tasks

11.4 Setting up Energy Probe on page 11-204
11.7 Energy Probe data in Arm Streamline

After you have successfully set up the Energy Probe and have run a capture session, you can see the Energy Probe data aligned to the software activity in the Timeline view.

If your target meets the requirements, and you have attached a power probe point that closely resembles the activity of the CPU cores using Channel 0, the Arm Streamline correlation algorithm aligns the data for you. Notes that appear when you click the warnings tag on the left side of the toolbar communicate any limitations or failures in this area.

When you have set up everything correctly, run a standard report capture and analysis. A large variance between idle and activity ensures that the correlation algorithm has something to lock onto, so set your workloads accordingly. If you have connected and configured your Energy Probe correctly, a power chart appears in the Timeline view.

![Figure 11-5 Energy Probe data in the Timeline view](image-url)
11.8 Setting up National Instrument Multifunction Data Acquisition devices (NI DAQ) to capture energy data

As an alternative to the Energy Probe, you can use a NI DAQ device to gather energy data from your target and view the results in Arm Streamline.

To collect power statistics using a NI DAQ device, you must have the following:

- NI DAQ hardware.
- NI-DAQmx software installed on your host machine, so `caiman` can communicate with the NI DAQ device. This software package includes drivers for the NI DAQ device. You must use the NI-DAQmx Base software on Linux.

In addition, you must set the location of the appropriate `caiman` application using the Tool Path field in the **Capture & Analysis Options** dialog box.

Note

National Instruments only distribute 32-bit versions of their libraries. Therefore only the 32-bit version of `caiman` works with the NI-DAQ device, even on 64-bit platforms. For example, a Windows 64-bit installation of Arm Development Studio contains a 32-bit version of `caiman`.

If the pre-built `caiman` executable that is distributed with Arm Streamline is insufficient, or you want to change some options, you can re-build `caiman` from source as follows:

1. Extract the source.
2. Open `CMakeLists.txt`.
3. Modify the settings at the beginning of the file as required. If necessary, modify `include_directories` and `target_link_libraries` to add other dependencies, such as NI-DAQ.
4. Use CMake to generate either a Makefile or a Visual Studio project.
5. Build the project as normal.

Note

`caiman` uses CMake so that one configuration can generate both Visual Studio and Makefile projects. See [http://www.cmake.org](http://www.cmake.org) for more information about CMake.

For example, to build a NI DAQ enabled version of `caiman` for Red Hat Enterprise Linux 6, set the following values in `CMakeLists.txt`:

- `SUPPORT_UDEV 0`.
- `SUPPORT_DAQ 1`.
- `NI_RUNTIME_LINK 0`.

Also, verify the NI DAQ install paths within `CMakeLists.txt`.

See the National Instruments website, [http://www.ni.com](http://www.ni.com) for the list of operating systems that NI DAQ supports.

To set up your NI DAQ device, follow these steps:

**Procedure**

1. Connect the Ai1 connections on the NI DAQ device to go across the shunt resistor on your target.
2. Connect Ai0 negative to ground.
3. Connect Ai GND to ground.
4. Loop Ai0 positive to Ai1 negative.
   - The Ai0 connectors are measured across the load, which is used to derive the voltage.
5. Repeat steps 1-3 using the other connectors on the NI DAQ device to measure additional channels.
6. In the **Streamline Data** view, click **Capture & Analysis Options**.
7. Select the **NI DAQ** option from the **Energy Capture** drop-down menu.
8. Enter a valid system name in the **Device** field.
   To get the system name of your National Instruments device, you must run the **NI-DAQmx Base list Devices** application, which is installed as part of the NI-DAQmx software package.
9. Click the **Select the energy capture tool** button in the **Tool Path** field.
10. Locate either the pre-built caiman in your Arm Development Studio installation directory, or the modified version that you built from source, and select it.
11. Click **Open**.

**Note**

The NI DAQ device takes a while to initialize with the NI-DAQmx Base drivers, therefore the first 3-8 seconds of power data is not captured.

**Related references**

4.1 **Capture & Analysis Options dialog box settings** on page 4-77
Chapter 12
Advanced gator Customizations

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

It contains the following sections:
• 12.1 Creating a configuration.xml file on page 12-212.
• 12.2 Capturing energy data locally and importing it to a capture on page 12-214.
• 12.3 Using the gator_events_mmapped.c custom counters example on page 12-215.
• 12.4 Creating custom performance counters with kernel space gator on page 12-216.
• 12.5 gator_events functions on page 12-218.
• 12.6 Creating filesystem and ftrace counters on page 12-219.
• 12.7 Attributes for custom, filesystem, and ftrace counters in the events XML file on page 12-220.
• 12.8 Enabling atrace and ttrace annotations on page 12-221.
• 12.9 Profiling the Linux kernel on page 12-222.
• 12.10 Adding support to gatord for a new CPU or perf PMU on page 12-223.
• 12.11 Increasing the memory that is available for Arm Streamline on page 12-225.
• 12.12 Automatically start gator on boot on page 12-226.
12.1 Creating a configuration.xml file

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

You normally configure counters using the Counter Configuration dialog, but if you have no network connection to your target, then you can manually create a configuration.xml file before running a capture session.

--- Note ---

Arm Streamline defines a default set of counters. If this is sufficient, you do not need to create a configuration.xml file.

---

Procedure

1. Start gatord, then run `ls /dev/gator/events` to list all the counters that your target supports.
   **Example:**
   
   The list might look like this:

   ```
   ARMv7_Cortex_A9_ccnt  Linux_block_rq_wr      Linux_power_cpu_freq
   ARMv7_Cortex_A9_cnt0  Linux_cpu_wait_contention Linux_proc_statm_freq
   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_rx
   Linux_block_rq_wr     Linux_power_cpu_freq
   ```

2. Counters whose name does not end in `_cnt<n>`, for example `Linux_block_rq_rd`, support a single event only. To find out which event these counters support, search for the counter name in events-*.xml in the gator daemon source code, for example:
   **Example:**
   
   ```bash
   $ grep Linux_block_rq_rd events-*.xml
   events-Linux.xml:    <event counter="Linux_block_rq_rd" title="Disk I/O" name="Read" units="B"
   description="Disk I/O Bytes Read"/>
   ```

   This shows that the `Linux_block_rq_rd` counter is associated with the Disk 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 events-*.xml file they are defined in.
   **Example:**
   
   For example, to search for the file that defines the `ARMv7_Cortex_A9` counters, use the following command:
   ```bash
   $ grep -l ARMv7_Cortex_A9 events-*.xml
   events-Cortex-A9.xml
   ```

   Next, view the contents of events-Cortex-A9.xml to find out which events the `ARMv7_Cortex_A9` counters correspond to, for example:
   ```xml
   $ cat events-Cortex-A9.xml
   <counter_set name="ARMv7_Cortex_A9_cnt" count="6"/>
   <category name="Cortex-A9" counter_set="ARMv7_Cortex_A9_cnt" per_cpu="yes" supports_event_based_sampling="yes">
   <event counter="ARMv7_Cortex_A9_ccnt" event="0xff" title="Clock" name="Cycles" display="hertz" units="Hz"
   average_selection="yes" average_cores="yes" description="The number of core clock cycles"/>
   <event event="0x00" title="Software" name="Increment" description="Incremented only on writes to the
   Software Increment Register"/>
   <event event="0x01" title="Cache" name="Instruction refill" description="Instruction fetch that causes a
   refill of at least the level of instruction or unified cache closest to the processor"/>
   </category>
   ```
This shows that the \texttt{ARMv7\_Cortex\_A9\_ccnt} counter is the Clock: Cycles event. It also shows that the \texttt{ARMv7\_Cortex\_A9\_cnt\langle n\rangle} counters can be associated with a large number of events, including Software: Increment, Cache: Instruction refill, and Cache: Inst TLB refill.

**Results:**

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

4. Create an empty \texttt{configuration.xml} file. Copy and paste into it the counter, event (if any), title, and name attributes from the \texttt{<event .../>} nodes for the events you are interested in.

**Example:**

For example, to add the Disk IO: Read event, copy the required attributes from \texttt{events-Linux.xml}. Your \texttt{configuration.xml} file should look like this:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configurations revision="3">
  <configuration counter="Linux\_block\_rq\_rd" title="Disk I/O" name="Read"/>
</configurations>
```

---

**Note**

The \texttt{Linux\_block\_rq\_rd} counter does not have an event attribute.

---

To add another event, for example Clock: Cycles, copy its attributes, including the event attribute, into a new \texttt{<configuration>} node:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configurations revision="3">
  <configuration counter="Linux\_block\_rq\_rd" title="Disk I/O" name="Read"/>
  <configuration counter="ARMv7\_Cortex\_A9\_ccnt" event="0xff" title="Clock" name="Cycles"/>
</configurations>
```

To add an event that does not have a counter attribute, for example Cache: Instruction refill, you must choose a counter value from the category for the event. For example, the category for the Cache: Instruction refill event uses \texttt{counter\_set="ARMv7\_Cortex\_A9\_cnt"}. This means you can choose one of the unused \texttt{ARMv7\_Cortex\_A9\_cnt\langle n\rangle} counters to associate with the Cache: Instruction refill event, for example \texttt{ARMv7\_Cortex\_A9\_cnt0}. Your \texttt{configuration.xml} file should look like this:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configurations revision="3">
  <configuration counter="Linux\_block\_rq\_rd" title="Disk I/O" name="Read"/>
  <configuration counter="ARMv7\_Cortex\_A9\_ccnt" event="0xff" title="Clock" name="Cycles"/>
  <configuration counter="ARMv7\_Cortex\_A9\_cnt0" event="0x01" title="Cache" name="Instruction refill"/>
</configurations>
```

---

**Note**

You can only associate each \texttt{\_cnt\langle n\rangle} counter with a single event at the same time, so you cannot include all possible events in this \texttt{configuration.xml} file.

---

These steps should enable you to create a \texttt{configuration.xml} file for a specific target to capture the events you are interested in.
12.2 Capturing energy data locally and importing it to a capture

If you want to run a capture session locally and want the resulting report to include energy data, it is possible to run caiman and gatord simultaneously and then merge the data from the two captures into a single report.

To import energy data into a capture, follow these steps:

Procedure
1. Open the Capture & Analysis Options dialog box.
2. Enable energy capture by selecting ARM Energy Probe or NI DAQ from the Energy Capture drop-down menu, then enter the settings for each channel.
3. Enter all other options as normal.
4. Export a session.xml file using the Export... option.
5. On the host, start a caiman local capture where the channel/resistance pairs match what you configured in the exported session.xml. For example:
   caiman -1 -r 0:20 -r 1:30
6. At the same time, trigger a gator local capture using the session.xml previously exported:
   gatord -s session.xml -o <YourFileName>.apc
7. When ready, stop the gator capture.
8. Immediately stop the caiman capture.
9. On your host machine, create a sub-folder in the .apc folder of the gator capture. Name the new folder energy.
10. Copy the following caiman local capture files to the newly created energy folder: 0000000000, captured.xml, and, if it exists, warnings.xml.
11. In the Streamline Data view, right-click on the merged capture and select Analyze... from the contextual menu.
12. Use the Offset energy chart data buttons in the Chart Configuration panel in the Timeline view to match the energy data with other capture data.
12.3 Using the gator_events_mmapped.c custom counters example

The file `gator_events_mmapped.c` is provided as an example of how to add custom counters to code that uses kernel space gator. It is located in the gator driver source code.

Incorporating the simulated examples from `gator_events_mmapped.c` into gator is a good way to familiarize yourself with the process of adding your own counters.

The gator driver source code is available from either of the following locations:

- `DS_install_directory/sw/streamline/gator/driver/`
- `https://github.com/ARM-software/gator`

To add a custom counter using `gator_events_mmapped.c`, follow these steps:

**Procedure**

1. Open the `gator_events_mmapped.c` example file in the editor of your choice.
2. Copy the XML from the comments section of `gator_events_mmapped.c`.
3. Create an XML file in the same directory as the gator source code, and call it `events-mmap.xml`.
4. Add the copied XML from the comments section of `gator_events_mmapped.c` to `events-mmap.xml`.
5. Remove any * comment markers from the copied XML.
6. Save `events-mmap.xml`.
7. Rebuild gator and copy it to the target.
8. Kill the old gator process if it is already running, then enter `/gator &` on the command line of your target to launch the newly built gator.
9. Open the **Counter Configuration** dialog using the button in the **Streamline Data** view.
   - **Results:** A new category, `mmapped`, appears in the Counter Configuration dialog box with the Sine, Triangle, and PWM simulated counters.
10. Add Sine to list of counters.
11. Run a capture session.
   - **Results:**
     - If successful, the waveform generated by the simulated Sine counter appears in the charts section of the Live and Timeline views.

![Sine counter chart](image)

**Figure 12-1** Sine counter chart

**Related references**

- 12.5 gator_events functions on page 12-218
- 12.7 Attributes for custom, filesystem, and ftrace counters in the events XML file on page 12-220
12.4 Creating custom performance counters with kernel space gator

In addition to the hardware-specific and Linux performance counters that you can configure using the Counter Configuration dialog, the gator daemon and driver provide hooks that enable you to create custom counters.

Note

This topic only applies if you are using kernel space gator. If you are using user space gator, create custom counters using the macros ANNOTATE_ABSOLUTE_COUNTER, ANNOTATE_DELTA_COUNTER, and ANNOTATE_COUNTER_VALUE instead.

Arm Streamline derives its default set of counters from the performance monitoring unit, Linux hooks, and memory-mapped peripherals. You can add your own counters to this list if there is a hardware metric that you want to track which Arm Streamline does not provide by default.

To create your own counters, mimic the methods used in the gator_events_mmapped.c file or any of the other gator_events_x files included with the gator driver source.

The gator driver source code is available from either of the following locations:

- DS_install_directory/sw/streamline/gator/driver/
- https://github.com/ARM-software/gator

Follow these steps to ensure that gatord interacts with your custom source:

**Procedure**

1. Create an empty gator_events_your_custom.c file or duplicate gator_events_mmapped.c.
2. Update the makefile to build the new gator_events_your_custom.c file.
3. Add the preprocessor directive `#include "gator.h"` if you do not use gator_events_mmapped.c as a template.
4. Implement the following functions in your new source file: gator_events_your_custom_init, gator_events_your_custom_interface, gator_events_your_custom_create_files, gator_events_your_custom_start, gator_events_your_custom_read, and gator_events_your_custom_stop.
5. Add gator_events_your_custom_init to GATOR_EVENTS_LIST in gator_main.c.
6. All of your new counters must be added to the events list. To do this:
   a. Create a new XML file in the same directory as the makefile before building gatord. The makefile pulls in any file that begins with events-, so create an XML file called events-YourCustom.xml.
   b. Define how many counters exist in your custom set using the `<counter_set name="counter_name" count="x">` tag. Give your counter a unique, descriptive name and enter how many counters are available for the count attribute. For example: `<counter_set name="ARM_Cortex-A9_cnt" count="6"/>
   c. For each counter set, you must list each of the possible events. Define the event category using the `<category>` tag.
   d. Define the individual events for each event category, including all of the necessary attributes.

A basic event example:

```xml
<event event="0x01" title="Cache" name="Instruction refill"
      description="Instruction fetch that causes a refill of at least the level of instruction or unified cache closest to the processor"/>
```

A fixed event example:

```
<event counter="ARM_Cortex-A9_ccnt" title="Clock" name="Cycles"
      display="hertz" units="Hz" average_selection="yes"
      description="The number of core clock cycles"/>
```
7. Re-build gatord after you create your new XML file.

Related concepts
9.1 Annotate overview on page 9-182

Related references
12.5 gator_events functions on page 12-218
12.7 Attributes for custom, filesystem, and ftrace counters in the events XML file on page 12-220
12.5 gator_events functions

To create your own custom counters, you must add gator functions to your source code.

This table gives a brief description of each of the gator events functions:

<table>
<thead>
<tr>
<th>Gator events function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gator_events_your_custom_init</td>
<td>gator calls this function at startup.</td>
</tr>
<tr>
<td>gator_events_your_custom_interface</td>
<td>Tells gator what triggers calls to your custom events file.</td>
</tr>
<tr>
<td>gator_events_your_custom_create_files</td>
<td>Adds custom directories and enabled, event, and key files to /dev/gator/events.</td>
</tr>
<tr>
<td>gator_events_your_custom_start</td>
<td>gator calls this at the start of execution.</td>
</tr>
<tr>
<td>gator_events_your_custom_read</td>
<td>gator calls this at every sample.</td>
</tr>
<tr>
<td>gator_events_your_custom_stop</td>
<td>gator calls this at the termination of a capture session.</td>
</tr>
</tbody>
</table>

Related tasks

12.3 Using the gator_events_mmapped.c custom counters example on page 12-215
12.4 Creating custom performance counters with kernel space gator on page 12-216
12.6 Creating filesystem and ftrace counters

gator supports reading generic files based on entries in events XML files. This feature allows you to add counters to extract data held in files, for example, /dev, /sys or /proc file entries. You can use a similar technique to add counters for ftrace data.

To add filesystem counters, you first need to create an events-xxx.xml file. Ensure that for each entry in the file, the counter attribute values begin with filesystem_ and are unique. Use path attributes to specify the names and locations of the files to read. An example file called events-Filename.xml containing commented-out entries is provided with the gatord source code.

gator reads the files specified in your events-xxx.xml ten times per second. By default, it interprets each file as an integer, but if you provide a regular expression using the regex attribute, gator applies it and converts the matched entry to an integer.

gator also supports reading ftrace data, using a similar technique. In the events-xxx.xml file, ensure the counter attribute values begin with ftrace_ and are unique. You can specify regular expressions to extract counter values. For an example, see events-ftrace.xml located in the gator daemon source code.

After creating the events XML file, you can either:

• Rebuild gatord. In this case, your events-xxx.xml file must be in the same location as the gatord makefile.
• Restart gatord using the -E command-line option, specifying the location of your events-xxx.xml file. This appends your filesystem and ftrace counters to the existing events that gator supports and avoids the need to rebuild gatord. If you choose this method, you need to specify the XML header and include an events node in your events-xxx.xml file. For example:

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

Note

Some ftrace counters are available by default in the Counter Configuration dialog, if the target supports them.

Related tasks

2.5.2 Building the gator daemon yourself on page 2-40

Related references

12.7 Attributes for custom, filesystem, and ftrace counters in the events XML file on page 12-220
2.11 gatord command-line options on page 2-49
12.7 Attributes for custom, filesystem, and ftrace counters in the events XML file

Events XML files define the counters that you can select in the Counter Configuration dialog box. In order to add custom, filesystem, or ftrace counters to gator, you must define them in your events XML file.

For descriptions of the valid XML nodes and attributes you can use in the events XML file, see the gator protocol documentation, located in DS_install_directory/sw/streamline/protocol/gator/. In addition, the following attributes are specific to Filesystem and Ftrace counters:

**regex**
A POSIX-extended regular expression used to extract a value from a file.

**path**
The absolute path and name of a file, for example /proc/stat.

**enable**
For ftrace counters only, specifies an ftrace event to enable. This attribute is optional.

**tracepoint**
This has the same meaning as enable, but uses perf instead of ftrace when using user space gator. This attribute is optional.

**arg**
Used in conjunction with tracepoint to specify the value to show. This attribute is optional. If not specified, the number of tracepoint events is counted.

**Related tasks**
12.3 Using the gator_events_mmapped.c custom counters example on page 12-215
12.4 Creating custom performance counters with kernel space gator on page 12-216
12.8 Enabling atrace and ttrace annotations

Arm Streamline supports atrace annotations on Android targets that are running Linux kernel versions 3.10 and later, and ttrace annotations on targets that are running Tizen version 2.4.

Arm Streamline converts application-generated atrace macros into either string annotations or counter charts. It also lists any Android ATRACE_TAG_* macros that you enable as available events in an Atrace section in the Counter Configuration dialog. If you expect to see atrace events in this dialog but none are displayed, click on the Warnings tag in the Counter Configuration dialog to see why atrace support is not enabled.

To notify running applications that atrace annotation tags have been enabled, the file notify.dex must be installed on the target in the same directory as gatord. You can install a pre-built version of notify.dex as part of target setup, by clicking the Setup target... button in the Connection Browser dialog. The Java source code for notify.dex is available in the following locations:

- DS_install_directory/sw/streamline/gator/notify/
- https://github.com/ARM-software/gator/tree/master/notify

To enable Tizen ttrace annotations:

- Specify the path to SDB in the ADB Path field in the dialog at Window > Preferences > Streamline > External Tools.
- Select a target Tizen device, with gator running on it, using the Connection Browser dialog.
- Select the ttrace events to capture from the Ttrace section in the Counter Configuration dialog.

Arm Streamline displays the ttrace tags as annotations in the Timeline view and Log view.

Related concepts
5.2 Counter Configuration dialog box structure on page 5-87

Related references
5.5 Counter Configuration dialog box settings on page 5-91
4.1 Capture & Analysis Options dialog box settings on page 4-77
3.2.1 Connection Browser dialog box on page 3-59
12.9 Profiling the Linux kernel

If you do not include the kernel in the images in the Capture & Analysis Options dialog box, the statistics generated by the kernel are not aligned with source code in the Analysis Reports. Before you can include the Linux kernel in the Program Images section of the Capture & Analysis Options dialog, you must build a version of vmlinux with kernel debug information enabled.

To profile the Linux kernel, follow these steps:

**Procedure**

1. Navigate to the kernel build directory.
2. Enter the following command to enable you to change menuconfig options:
   ```
   make ARCH=arm CROSS_COMPILE=<cross_compiler_directory>/bin/arm-linux-gnueabihf-
   menuconfig
   ```
3. In the Kernel Hacking menu, select the Compile the kernel with debug info option. This enables the CONFIG_DEBUG_INFO kernel option.
   The options described in 2.4 Required kernel configuration menu options on page 2-38 must also be enabled.
4. Enter the following command to build the image:
   ```
   make -j5 ARCH=arm CROSS_COMPILE=<cross_compiler_directory>/bin/arm-linux-
   gnueabihf- uImage
   ```

**Results:** This creates a new vmlinux image.

**Note**

You can profile a driver by either statically linking it into the kernel image or by adding the module as an image in the Capture & Analysis Options dialog box.

5. Optional: Enable kernel stack unwinding using either of the following methods:
   - Remove the comment tags that surround GATOR_KERNEL_STACK_UNWINDING in the Makefile and rebuild gator.ko.
   - Run the following command as root on the target after gatord has started:
     ```
     echo 1 > /sys/module/gator/parameters/kernel_stack_unwinding
     ```

**Note**

- This step is only required if you previously built gator.ko with kernel stack unwinding turned off.
- Enabling kernel stack unwinding might trigger errors that appear at millisecond intervals during the capture session. If you experience this behavior, disable kernel stack unwinding.

6. Open the Capture & Analysis Options dialog box.
7. Click the Add ELF image... button in the Program Images section.
8. Navigate to your vmlinux file and select it.
9. Click OK
10. Start a new capture session.

**Related tasks**

2.6 Building the gator module on page 2-42

**Related references**

2.4 Required kernel configuration menu options on page 2-38
12.10 Adding support to gatord for a new CPU or perf PMU

Use either of the following ways to add support for a new CPU or perf PMU to gatord. The information in this topic applies to gator version 23 and later.

--- Note ---
- Perf support in Linux for the PMU is required because gatord uses perf to read the hardware counters.
- Check the perf PMUs supported by your kernel by running

```bash
ls /sys/bus/event_source/devices/
```
If ARMv7_Cortex_Axx, CCI_400, or cnn are listed, then Axx, CCI-400, or CNN respectively are supported.
- Only XML changes are required, no code changes are necessary.
- Rebuilding gatord after the XML changes is recommended but not required because you can pass PMUs and events to gatord on the command line.

Make the following changes, then rebuild gatord:

- Add a line to $DS_install_directory/sw/streamline/gator/daemon/pmus.xml describing the new PMU. For CPUs, the following information is required:
  - The CPU Implementer and Primary part number from the Main ID Register.
  - The number of generic hardware counters that can be selected simultaneously.
  - Optionally, set the perf PMU name of the CPU to ensure correct operation in multi-PMU Arm big.LITTLE configurations.
- Create an events XML file, named events-xxx.xml in the gator daemon source directory that defines the events that the new PMU generates. This file should exclude the XML header and `<events>` element. See the Cortex-A15 events XML file, $DS_install_directory/sw/streamline/gator/daemon/events-Cortex-A15.xml for an example.

Alternatively, to add support without having to rebuild gatord, do the following:

- Create an events XML file that defines the events that the new PMU generates. This file must include the XML header and `<events>` element, for example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<events>
  <category name="Cortex-A9" counter_set="ARMv7_Cortex_A9_cnt" per_cpu="yes" supports_event_based_sampling="yes">
    <event counter="ARMv7_Cortex_A9_ccnt" event="0xff" title="Clock" name="Cycles" display="hertz" units="Hz" average_selection="yes" average_cores="yes" description="The number of core clock cycles"/>
    <event event="0x00" title="Software" name="Increment" description="Incremented only on writes to the Software Increment Register"/>
    <event event="0x01" title="Cache" name="Instruction refill" description="Instruction fetch that causes a refill of at least the level of instruction or unified cache closest to the processor"/>
  </category>
</events>
```

- Create an XML file that defines information about the new PMU. For the required format, see the gator pmus.xml. For example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<pmus>
  <pmu pmnc_name="ARMv7_Cortex_A9" cpuid="0x41c09" core_name="Cortex-A9" pmnc_counters="6"/>
</pmus>
```

- Copy these files to the target and restart gatord using the following flags:
  - `-E` to specify the location of the events XML file. This flag causes the events to be appended to the list of events that gatord supports.
  - `-P` to specify the location of the PMU XML file. This option causes the new PMU to be added to the list of PMUs defined in pmus.xml that gatord has built-in support for.
Related tasks
2.5.2 Building the gator daemon yourself on page 2-40

Related references
2.11 gatord command-line options on page 2-49
12.11 Increasing the memory that is available for Arm Streamline

To increase the amount of memory that is available for Arm Streamline, you must modify the Arm Streamline ini file.

Arm Streamline supports multiple operating systems, so you must select the ini file that is specific to your operating system. To find the correct file, see the following table:

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Ini file location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 64-bit, GUI</td>
<td>DS_install_directory/sw/streamline/Streamline-gui.ini</td>
</tr>
<tr>
<td>Windows 64-bit, command line</td>
<td>DS_install_directory/sw/streamline/Streamline.ini</td>
</tr>
<tr>
<td>Windows 32-bit, GUI</td>
<td>DS_install_directory/sw/streamline/Streamline-32-gui.ini</td>
</tr>
<tr>
<td>Windows 32-bit, command line</td>
<td>DS_install_directory/sw/streamline/Streamline-32.ini</td>
</tr>
<tr>
<td>Linux 64-bit</td>
<td>DS_install_directory/sw/streamline/.Streamline.ini</td>
</tr>
<tr>
<td>Linux 32-bit</td>
<td>DS_install_directory/sw/streamline/.Streamline-32.ini</td>
</tr>
<tr>
<td>Mac OS</td>
<td>DS_install_directory/sw/streamline/Streamline.app/Contents/MacOS/Streamline.ini</td>
</tr>
</tbody>
</table>

**Procedure**

1. Open the correct ini file and edit the `-Xmx` setting. For example, to set aside 4GB of memory for Arm Streamline, change it to:

   ```
   -Xmx4g
   ```

   __Note__

   You might have to change the access permissions for the file before you can edit it.

2. Save and close the file.
3. Restart Arm Streamline.

**Related information**

"JVM terminated" error when launching DS-5 / OutOfMemoryError reported when using DS-5
12.12 Automatically start gator on boot

Run the following commands on the target device to automatically start gator when you turn on the device:

```bash
cd /etc/init.d
cat << EOF > rungator.sh
#!/bin/bash
<path_to_gatord> &
EOF
update-rc.d rungator.sh defaults
```
Chapter 13
Bare-Metal Support

Describes the bare-metal support available within Arm Streamline.

It contains the following sections:
• 13.1 Bare-metal support overview on page 13-228.
• 13.2 Profiling with on-target RAM buffer on page 13-229.
• 13.3 Profiling with System Trace Macrocell on page 13-235.
• 13.4 Profiling with Instrumentation Trace Macrocell on page 13-238.
• 13.5 Profiling with Embedded Trace Macrocell on page 13-241.
• 13.6 Profiling with Instruction Trace on page 13-245.
• 13.7 Custom counters on page 13-250.
• 13.8 Data storage on page 13-252.
• 13.9 Interfacing with barman on page 13-253.
13.1 Bare-metal support overview

Bare-metal support allows Arm Streamline to visualize elements of the system state of a target device that is not running a Linux-based operating system.

Note

See 1.1 Features of Arm Streamline editions on page 1-17 to check if your edition includes Bare-metal support.

Specifically, support is provided for systems without the following:

- A connected trace device.
- A Linux-based operating system for which the gator daemon and kernel module are provided.

Bare-metal support is provided by an agent that is referred to as barman. It consists of two C source files, barman.c and barman.h that you build into the executable that runs on the target device. A configuration and generation utility generates these files.

To use barman, you must modify your existing executable to do the following:

- Initialize barman at runtime.
- Periodically call the data collection routines that barman provides.
- Optionally, stop the capture.
- Optionally, extract the raw data that barman collects and provide it to Arm Streamline for analysis.

Barman has the following features:

- It captures PMU counter values from Cortex-A and Cortex-R class processors.
- It captures sampled PC values.
- It captures custom counters.
- It allows you to control the sample rate.
- It writes the data that it collects to memory.
- It has low data collection overhead.

Barman supports the following Arm architectures:

- Armv7-A
- Armv7-R
- Armv7-M
- Armv8-A, both AArch32 and AArch64.
- Armv8-R
- Armv8-M

See DS_install_directory/sw/streamline/examples/barman for the following Bare-metal examples:

Streamlinebare_metal_ARmv8_AArch64
A demonstration of how to use barman with AArch64, from configuring the bare-metal agent to analyzing the results.

Streamlinebare_metal_Cortex_R5
A demonstration of how to use barman with Arm Cortex-R5, from configuring the bare-metal agent to analyzing the results.

Streamlinebare_metal_M_profile
A demonstration of how to use barman with Armv7-M and Armv8-M, from configuring the bare-metal agent to analyzing the results.

u-boot-instrumentation
An example of how to modify U-Boot to allow it to be profiled using barman.
13.2 Profiling with on-target RAM buffer

This section describes how to collect profiling data using the RAM buffer on the target device.

13.2.1 Configuring barman

You must configure barman with the configuration and generation utility before you compile the binary executable to be analyzed. Barman must then be built into the executable.

The configuration and generation utility is a wizard dialog available from the Streamline menu. The generated header and source files, and the configuration XML file, are then saved into a folder of your choice. The generation mechanism is also accessible from the command line.

Procedure

1. Access this utility from Streamline > Generate Barman Sources.
2. Configure the default configuration options, such as the number of processor elements, whether you intend to supply executable image memory map information, whether you intend to provide process or task level information (for example if you are running an RTOS), and configure data storage mode (linear or circular RAM buffer).

![Barman Generator Wizard](image)

--- Note ---

See 13.3 Profiling with System Trace Macrocell on page 13-235 for information about using the STM Interface data storage backend.

See 13.4 Profiling with Instrumentation Trace Macrocell on page 13-238 for information about using the ITM Interface data storage backend.
3. Select the target processor from the pre-defined list.

![Barman Generator Wizard](image)

4. Select the PMU counters to collect during the capture session by double clicking on them in the Available events list. Alternatively you can drag and drop the events into the Selected events list. To deselect events, drag and drop them back into the Available events list.
5. Add custom counters.

6. Select generator options.
7. Finish.

Your bare metal agent files will be generated into 'C:\'.

- You selected to generate source files, you will find two files named 'barman.h' and 'barman.c' which must be compiled and linked into your program.

- You selected to save the configuration file, you will also find a file named 'barman.xml' in the same location. This file contains the configuration settings specified in this wizard, allowing you to reload, edit or recreate the source files at a later date, or with the command line tool.

By pressing the Finish button you will complete the generation process.

For more information please see the Streamline documentation.
The setup process produces the following output:

- A configuration file, `barman.xml`, which contains the settings that were entered into the configuration wizard, and which can be used to reproduce the same configuration later.
- `barman.c`. You must compile and link this file into the bare-metal executable.
- `barman.h`. You must include this header when calling any of the functions within the agent. It also declares function prototypes for the functions you must implement.
- `barman_in_memory_helpers.py`. You can use this file as a use case script in Arm Development Studio. It helps you dump the contents of the in-memory capture buffer.

You need the compiler flag `-gnucf` for armcc (Arm Compiler 5) to compile `barman.c`.

### Related tasks

15.5 Accessing the Bare-Metal generation mechanism from the command line on page 15-279

### Related references

13.2.3 Barman use case script on page 13-233

### Extracting and importing data

You must extract the data from the RAM buffer when the capture is complete.

For example, you could choose to do one of the following:

- Save the data to the file system of the target device, if one exists.
- Retrieve the data from RAM using JTAG during a debug session.
- Transfer the data over one of the available communication interfaces, for example ethernet or USB.

After extracting the raw data, give the data file a `.raw` extension. You can import this file into Arm Streamline by clicking **Import Capture File(s)**... The imported data is then available for Arm Streamline to analyze.

If you added a custom `pmus.xml` or `events.xml` file during the configuration and generation stage, you must provide a copy of the same file into the `.apc` directory that is created for the imported capture. The files must be named `pmus.xml` and `events.xml` and must be placed in the directory alongside the `barman.raw` file for them to be detected and used.

### Related references

15.6 Importing a trace from the command-line on page 15-280

### Barman use case script

Arm Streamline generates the file `barman_in_memory_helpers.py` with the barman agent sources when you select an in-memory data storage backend. You can use it as a use case script in Arm Development Studio to help you dump the contents of the in-memory capture buffer.

Run the script with the following command:

```
usecase run "barman_in_memory_helpers.py" <usecase_command>
```

Two use case commands are available:

- **get_parameters**
  - Prints the current details of the buffer and information about how to dump it.

- **dump**
  - Dumps the contents of the memory buffer in a file that you specify with the option `--file <PATH>`.

### Examples

The following examples show how to use these use case commands.

- To use the get_parameters use case command, enter:

  ```
  usecase run "barman_in_memory_helpers.py" get_parameters
  ```
Output:

Barman memory buffer details:
- Base address: 0x000000000001580
- Dump length: 1787404
- Bytes written: 1785996 of 67099264 (2.7%)

To dump this buffer use the command:
dump memory <PATH> 0x1580 +1787404
Or use the usecase command 'dump':
usecase run "barman_in_memory_helpers.py" dump --file <PATH>

- To use the dump use case command, enter:
  usecase run "barman_in_memory_helpers.py" dump --file barman.raw

Output:

Executing command:
dump binary memory "barman.raw" 0x1580 +1787404
Memory successfully dumped to file barman.raw

Related tasks
13.2.1 Configuring barman on page 13-229

Related information
Use case scripts
13.3 Profiling with System Trace Macrocell

This section describes the collection of profiling data using System Trace Macrocell (STM).

Further information about STM, including the Technical Reference Manual, can be found on Arm Developer.

13.3.1 STM workflow

The workflow for STM involves a complex series of interactions between the applications involved.

1. Generate barman agent code for STM using the Barman Generator Wizard dialog in Arm Streamline.
   a. Select STM Interface as the data storage backend.
   b. Specify the STM parameters for your project.
Barman reserves the channels following the channel number that you specify. The number of channels reserved is the **Maximum number of CPU cores** specified on the previous page of the wizard.

c. Complete the remainder of the wizard as for a standard bare-metal project.

2. Add the barman agent files that the wizard generates to your project.
3. Instrument your bare-metal application code with barman agent calls (initialization, periodic sampling).
4. Compile and link your project.
5. Connect your target to a DSTREAM device.
6. Configure your target for collecting STM data into its RAM buffer.
7. Run the application on a target.
8. When you want to end the profiling, stop the application.
9. Dump the STM trace from the DSTREAM device into a directory.
10. Let Arm Streamline import the trace file dump. Arm Streamline reformats it and prepares it for analysis.

--- Note ---

• If you are using Arm Development Studio, you can dump the STM trace into a directory using the following command:

```bash
trace dump <directory> STM
```

• If you do not launch your bare-metal application from within Arm Development Studio, you must handle connecting to DSTREAM, obtaining the trace file, and importing it into Arm Streamline.

---

**Related tasks**

13.2.1 Configuring barman on page 13-229

**13.3.2 Importing an STM trace**

Import STM trace files into Arm Streamline for analysis.
Note

You can also import the STM trace from the command-line.

Procedure

1. Click **Import Capture File(s)...** in the Streamline Data view.
2. Select the import file type **STM Trace Files (STPv2)**.

3. Select the trace file to import.
4. Click **Open** and a new dialog box opens.
5. Enter the location of the `barman.xml` file that the **Barman Generator Wizard** produced.
   This file contains information about how to find relevant data in the trace file. For example, the channel numbers used.
6. Click **OK**.

Arm Streamline then reformats the data, and converts the STM trace file into a barman agent raw file.

Related references

15.6 Importing a trace from the command-line on page 15-280
13.4 Profiling with Instrumentation Trace Macrocell

This section describes the collection of profiling data using Instrumentation Trace Macrocell (ITM).

13.4.1 ITM workflow

The workflow for ITM involves a complex series of interactions between the applications involved.

1. Generate barman agent code for ITM using the Barman Generator Wizard dialog in Arm Streamline.
   a. Select ITM Interface as the data storage backend.

   ![Barman Generator Wizard]

   _________ Note _________

   Barman uses ports 16-19 for ITM.

   _________

   b. Complete the remainder of the wizard as for a standard bare-metal project.
   c. If you selected a Cortex-M processor, select the number of cycles for the PC sampling interval.
2. Add the barman agent files that the wizard generates to your project.
3. Instrument your bare-metal application code with barman agent calls (initialization, periodic sampling).
4. Compile and link your project.
5. Connect your target to a DSTREAM device.
6. Configure your target for collecting ITM data into its RAM buffer.
7. Run the application on a target.
8. When you want to end the profiling, stop the application.
9. Dump the ITM trace from the DSTREAM device into a directory.
10. Let Arm Streamline import the trace file dump. Arm Streamline reformats it and prepares it for analysis.

Note

- If you are using Arm Development Studio, you can dump the ITM trace into a directory using the following command:

```
trace dump <directory> ITM
```

- If you do not launch your bare-metal application from within Arm Development Studio, you must handle connecting to DSTREAM, obtaining the trace file, and importing it into Arm Streamline.

---

Related tasks

13.2.1 Configuring barman on page 13-229

13.4.2 Importing an ITM trace

Import ITM trace files into Arm Streamline for analysis.

Note

You can also import the ITM trace from the command-line.

---

Procedure

1. Click Import Capture File(s)… in the Streamline Data view.
2. Select the import file type ITM Trace Files.
3. Select the trace file to import.

4. Click **Open** and a new dialog box opens.

5. Enter the location of the `barman.xml` file that the **Barman Generator Wizard** produced.

   This file contains information about how to find relevant data in the trace file. For example, the channel numbers used.

6. Click **OK**.

Arm Streamline then reformats the data, and converts the ITM trace file into a barman agent raw file.

*Related references*

15.6 Importing a trace from the command-line on page 15-280
13.5 Profiling with Embedded Trace Macrocell

This section describes the collection of profiling data using Embedded Trace Macrocell (ETM).

13.5.1 ETM workflow

The bare-metal agent can use the Data Trace feature that is provided with the ETM for streaming data from an R-class device in Arm Development Studio.

Note

The ETM interface requires ETM 3 or ETM 4 with data address tracing enabled.

To use ETM, generate the bare-metal agent as follows:

1. Open the Barman Generator Wizard by selecting Streamline > Generate Barman sources.
2. Select ETM interface as the data storage backend.
3. Click Finish.

To trace the device in Arm Development Studio, enable data address tracing from within the DTSL settings of your debug connection. For ETM 3, Arm Streamline supports data-only mode. You can enable data value tracing, however it is not required, and it slows execution and greatly increases the trace size.

Figure 13-1  ETM3 configuration

Note

Different devices have different sets of options so this dialog may vary from the image shown.

You must use either DSTREAM or a large in-memory trace buffer for storing the trace. If you select System Memory Trace Buffer (ETR/TMC), configure the in-memory trace buffer in the ETR tab. It is technically possible to use an on chip ETB for storing the trace, but the limited size, often less than 1KB, is not sufficient to store the capture.
Figure 13-2 Select the Trace capture method in the DTSL configuration dialog.

For ETM4, you must enable ETM tracing for all cores you are interested in collecting data from. On this tab, you can also enable data address tracing. If the Enable ETM Timestamps option is available, select it.
You can run the provided python script, `barman_etm_filter_script.py`, as part of the debug configuration. The **Barman Generator Wizard** outputs this script alongside the generated source files. Set it as the debug initialization debugger script in your target debug configuration in Arm Development Studio. In the **Debug Configurations** dialog, click the **Debugger** tab, and enter the location of the script in the **Run debug initialization debugger script** field. This script limits tracing to the part of the bare-metal agent that sends the data. This limitation prevents you from getting the instruction trace, but reducing the amount of trace data in this way leads to smaller captures, faster imports, and the possibility to capture traces for longer. If you use ETM 4, Arm strongly recommends that you run this script. ETM 4 imports are much slower than ETM 3 imports. Limiting the trace data reduces the import time.

Use the following command to dump the ETM trace data:

```
trace dump <OUTPUT> <ETM_SOURCES>
```

Import this trace data into Arm Streamline using the import button, or using the `-import-etm-dt` command-line option, in the same way as for STM and ITM.

**Related references**

15.6 Importing a trace from the command-line on page 15-280

**Related information**

*Arm debug and trace architecture*

*ETM3 Architecture Specification*
ETM4 Architecture Specification
Debug and Trace Services Layer (DTSL)
DTSL Configuration Editor dialog box
13.6 Profiling with Instruction Trace

This section describes how to import an instruction trace, and restrictions related to this task.

Note

See 1.1 Features of Arm Streamline editions on page 1-17 to check if your edition includes support for instruction trace.

13.6.1 Importing instruction trace

Import a trace of the instructions that your application executed for Arm Streamline to analyze. The supported forms of instruction trace are PTM 1.0-1.1, ETM 3.0-3.5, and ETM 4.0-4.2.

Prerequisites

1. Collect the instruction trace for your application using Arm DS Debugger. See the Arm® Arm DS Debugger User Guide for instructions.
2. Export the instruction trace using the following command:

```
trace dump <output> <instruction_trace_sources>
```

Note

<instruction_trace_sources> must only specify valid ETM sources. For example, CSETM_APP_0.

Procedure

1. Click Import Capture File(s)....
2. Navigate to the directory that contains the trace dump.
3. Select any of the files in the directory.
   A wizard opens, enabling you to choose what to do with the import.
4. Select Instruction Trace.
5. Select the sources to import, and specify the timing information for them.
6. Select the executable images that were run.

Arm Streamline uses these files to decode the trace.

7. Configure parameters for the capture that Arm Streamline generates from the trace.
Arm Streamline reads the instruction trace and generates a capture from it. The generated capture contains the following:

- Charts that give approximate information about branching, instructions, load/stores, and exceptions.
- **Call Paths** and **Functions** views that Arm Streamline derives from the instruction trace.
- Optionally, the **Exceptions** view, which shows the exceptions that were taken.

**Related information**

- Configuring trace for bare-metal or Linux kernel targets
- Capturing trace data using the command-line debugger

### 13.6.2 Instruction trace notes and restrictions

There are some restrictions to instruction traces.

- Dynamic compilation/code modification is not supported. If the modification happens before the code is traced, and you supply an ELF image containing the executable code after modification, Arm Streamline can support one-off runtime code modification. You can use the Arm Development Studio memory dump command, specifying the ELF output format.
- You must specify whether the imported files are M-profile, otherwise interrupts are not shown correctly. This does not usually affect the import of the trace, just the output.
- The **Call Paths** view tracks function entry and exit instructions, and attempts to track state across interrupts even when simple context switching is used. This functionality may not be compatible with a more complex OS.

Arm Streamline generates timing information from the trace in one of the following ways:

- Using timestamps, if they are included in all the relevant trace streams.
  - Use this method if it is available, especially if you are importing streams from more than one processing element.
  - Timestamps give a near-accurate representation of the relative times of different events. The accuracy of the timing information depends on the relative frequency of timestamp packets within each trace stream.
— Timestamps allow showing periods of inactivity such as during WFE/WFI.
— Timestamps are usually clocked using a separate clock to the processor. Arm recommends that you enter the duration of the capture rather than the clock frequency for the timestamp clock, unless you know the exact frequency.

• Using cycle counts, if cycle-accurate mode is enabled for all relevant trace streams.
  — Use this mode if timestamps are not available as it gives better relative timing information between different instructions in the trace.
  — Timing information is not synchronized across different processing elements.
  — Unless the cycle counter increments during WFE/WFI instructions, it is not possible to see how long the processor waited for.
  — Systems using dynamic frequency scaling are not clocked correctly because the cycle count for individual instructions does not change with the clock frequency.

• Using the instruction counter.
  — The only mode that is universally available.
  — This mode has all the same limitations as the cycle counter mode.
  — This mode cannot display relative timing information for different instructions.
13.7 Custom counters

You can configure one custom chart, with one or more series, in the configuration wizard.

13.7.1 Configuring custom counters

You can configure chart properties for custom counters. The following chart properties can be configured:

**Name**
Human readable name for the chart.

**Series Composition**
Defines how to arrange series on the chart (stacked, overlay, or logarithmic).

**Rendering Type**
Defines how to render series on the chart (filled, line, or bar).

**Per Processor**
Indicates whether the data in the chart is per processor.

**Average Selection**
Sets whether the Cross Section Marker in Arm Streamline displays average values.

**Average Cores**
Sets whether Arm Streamline averages the values of multiple cores when viewing the aggregate data of a per processor chart.

**Percentage**
Sets whether to display data as a percentage of the maximum value in the chart.

The following series properties can be configured:

**Name**
Human readable name for the series.

**Units**
Defines the unit type to display in Arm Streamline.

**Sampled**
When set to true, the value for this counter is sampled along with the PMU counters. When false, you must call a function to update the counter value.

**Multiplier**
Number to multiply by for fixed-point math. As the data sent from the agent is int64, it must be scaled. For example, the value 1.23 can be represented by the value 123 with a multiplier of 0.01.

**Class**
Specifies the nature of the data that is fed into the chart as follows:

**delta**
Used for values that increment or are accumulated over time, such as hardware performance counters. The exact time when the data occurs is unknown and therefore the data is interpolated between timestamps.

**incident**
The same as delta, except the exact time is known so no interpolation is calculated. Used for counters such as software trace.

**absolute**
Used for singular or impulse values, such as system memory used.
The display value determines how to calculate the data when zooming out for each time bin as follows:

**accumulate**
Sum up the data (valid only for delta and incident class counters).

**hertz**
Does the same as accumulate then normalizes the value to one second (valid only for delta and incident class counters).

**minimum**
Display the smallest value encountered (valid only for absolute class counters).

**maximum**
Display the largest value encountered (valid only for absolute class counters).

**average**
Display the average (valid only for absolute class counters).

**Color**
The color to display the series in. If not set, Arm Streamline selects a color.

**Description**
Human readable description for the series. This description becomes the tooltip when hovering over the series in Arm Streamline.

### 13.7.2 Sampled and nonsampled counters

Sampled counters are polled when the PMU counter values are read.

For each sampled counter, a function prototype is generated of the following form:

```c
extern bm_bool barman_cc_<chart_name>_<series_name>_sample_now(bm_uint64 * value_out);
```

For example:

```c
extern bm_bool barman_cc_interrupts_fiq_sample_now(bm_uint64 * value_out);
```

You must implement this function to set the value of the `uint64` at `*value_out` to the value of the counter, then return `BM_TRUE`. If the counter value cannot be sampled, the function can return `BM_FALSE` and be skipped.

You are responsible for writing nonsampled counters to the capture. For each nonsampled series, the following two functions are declared:

```c
bm_bool barman_cc_<chart_name>_<series_name>_update_value(bm_uint64 timestamp, bm_uint32 core, bm_uint64 value);
bm_bool barman_cc_<chart_name>_<series_name>_update_value_now(bm_uint64 value);
```

For example:

```c
bm_bool barman_cc_interrupts_fiq_update_value(bm_uint64 timestamp, bm_uint32 core, bm_uint64 value);
bm_bool barman_cc_interrupts_fiq_update_value_now(bm_uint64 value);
```

The second function is a shorthand for the first that passes the current timestamp and core number to the appropriate arguments.

When you call these functions, the value for the counter is stored to the capture.
13.8 Data storage

Barman uses a simple abstraction layer for handling the storage of collected data. Typically, the data that barman collects is stored in a RAM buffer on the target.

You can choose from the following data storage modes provided:

**Linear RAM buffer mode**
- Data collection stops when the buffer is full. This mode ensures that no collected data is lost, but no further data can be recorded.

**Circular RAM buffer mode**
- Data collection continues after the buffer is full and the oldest data is lost as it is overwritten by the newest data. This mode gives you control over when the data collection ends.

**STM Interface**
- System Trace Macrocell (STM) data is collected on a DSTREAM device that is connected to the target, or by another similar method. You then dump the STM trace into a directory, which you can import into Arm Streamline for analysis.

**ITM Interface**
- Instrumentation Trace Macrocell (ITM) data is collected on a DSTREAM device that is connected to the target, or by another similar method. You then dump the ITM trace into a directory, which you can import into Arm Streamline for analysis.

For barman to be able to use either of the RAM buffer modes, you must first provide the RAM buffer on the target device. The RAM buffer is a dedicated, contiguous area of RAM that barman can write data to. On multiprocessor systems, the RAM buffer must be at the same address for all processors. It is your responsibility to allocate memory for the RAM buffer, either statically or dynamically.
13.9 Interfacing with barman

When barman is linked into your executable code, the code must call the following functions:

1. `barman_initialize` to initialize barman.
2. `barman_enable_sampling` to enable sampling.
3. The appropriate sample function, `barman_sample_counters` or `barman_sample_counters_with_program_counter`, to periodically collect data.

In a multiprocessor system, a call to one of the sampling functions only reads the counters for the processor element the code is currently executing on.

If you are running a preemptive kernel, RTOS, or similar, you must ensure that the thread running a call to a sampling function is not migrated from one processor element to another during the execution of the call.

In a multiprocessor system, if you are using periodic sampling (for example with a timer interrupt), you must provide a mechanism to call the sampling function for each processor element. In other words, to capture the counters of each processor element, there must be a timer interrupt or thread that is run separately on each processor element.

13.9.1 Configuration #defines

The following defines are configured by the configuration UI and stored in `barman.h`. They can be overridden at compile time as compiler parameters.

<table>
<thead>
<tr>
<th>Define</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM_CONFIG_ENABLE_LOGGING</td>
<td>Enables logging of messages when set to true.</td>
</tr>
<tr>
<td>BM_CONFIG_ENABLE_DEBUG_LOGGING</td>
<td>If <code>BM_CONFIG_ENABLE_LOGGING</code> is true, enables debug messages when set to true.</td>
</tr>
<tr>
<td>BM_CONFIG_ENABLE_BUILTIN_MEMFUNCS</td>
<td>Enables the use of built-in memory functions such as <code>__builtin_memset</code> and <code>__builtin_memcpy</code> when set to true.</td>
</tr>
<tr>
<td>BM_CONFIG_MAX_CORES</td>
<td>The maximum number of processor elements supported.</td>
</tr>
<tr>
<td>BM_CONFIG_MAX_MMAP_LAYOUTS</td>
<td>The maximum number of <code>mmap</code> layout entries to be stored in the data header. Configure to reflect the number of sections to be mapped for any process images.</td>
</tr>
<tr>
<td>BM_CONFIG_MAX_TASK_INFOS</td>
<td>The maximum number of distinct task entries that will be stored in the data. For single-threaded applications, this can be defined as zero to indicate that no information is provided. For multi-threaded applications or RTOS, this value indicates the maximum number of entries to store in the data header for describing processes, threads, and tasks.</td>
</tr>
<tr>
<td>BM_CONFIG_MIN_SAMPLE_PERIOD</td>
<td>The minimum period between samples in nanoseconds. If this value is greater than zero, calls to sampling functions are rate limited to ensure that there is a minimum interval of nanoseconds between samples.</td>
</tr>
<tr>
<td>BARMAN_DISABLED</td>
<td>Disables the barman entry points at compile time when defined to a nonzero value. Use to conditionally disable calls to barman, for example in production code.</td>
</tr>
</tbody>
</table>

13.9.2 Annotation #defines

Color macros to use for annotations.
<table>
<thead>
<tr>
<th>Define</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM_ANNOTATE_COLOR_color name</td>
<td>Named annotation color, where color name is one of the following colors: RED, BLUE, GREEN, PURPLE, YELLOW, CYAN, WHITE, LTGRAY, DKGRAY, BLACK</td>
</tr>
<tr>
<td>BM_ANNOTATE_COLOR_CYCLIC</td>
<td>Annotation color that cycles through a predefined set.</td>
</tr>
<tr>
<td>BM_ANNOTATE_COLOR_RGB(R, G, B)</td>
<td>Create an annotation color from its components, where R, G, and B are defined as follows: R: The red component, where 0 ≤ R ≤ 255. B: The blue component, where 0 ≤ B ≤ 255. G: The green component, where 0 ≤ G ≤ 255.</td>
</tr>
</tbody>
</table>

### 13.9.3 Barman public API

Call the following public API functions to use the bare-metal agent.

#### barman_initialize

The prototype of barman_initialize varies depending on the datastore chosen.

When using the linear or circular RAM buffer:

```c
BM_NONNULL((1, 3, 4))
bool barman_initialize(bm_uint8 * buffer, bm_uintptr buffer_length,
```

When using STM:

```c
BM_NONNULL((2, 3, 4))
bool barman_initialize_with_stm_interface(void * stm_configuration_registers,
```

When using ITM on Arm M-profile architectures:

```c
BM_NONNULL((1, 2))
bool barman_initialize_with_itm_interface(
```

When using ITM on Arm A- or R-profile architectures:

```c
BM_NONNULL((1, 2, 3))
bool barman_initialize_with_itm_interface(void * itm_registers,
```

The remaining parameters for each datastore are the same:

```c
const char * target_name,
const struct bm_protocol_clock_info * clock_info,
#if BM_CONFIG_MAX_TASK_INFOS > 0
bm_uint32 num_task_entries,
const struct bm_protocol_task_info * task_entries,
#endif
#if BM_CONFIG_MAX_MMAP_LAYOUTS > 0
bm_uint32 num_mmap_entries,
const struct bm_protocol_mmap_layout * mmap_entries,
#endif
bm_uint32 timer_sample_rate);
```
Initialize barman.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer</td>
<td>Pointer to in memory buffer.</td>
</tr>
<tr>
<td>buffer_length</td>
<td>Length of the in memory buffer.</td>
</tr>
<tr>
<td>stm_configuration_registers</td>
<td>Base address of the STM configuration registers. This parameter can be NULL if it will be initialized elsewhere, for example by the debugger.</td>
</tr>
<tr>
<td>stm_extended_stimulus_ports</td>
<td>Base address of the STM extended stimulus ports.</td>
</tr>
<tr>
<td>itm_registers</td>
<td>Base address of the ITM registers.</td>
</tr>
<tr>
<td>datastore_config</td>
<td>Pointer to the configuration to pass to barman_ext_datastore_initialize.</td>
</tr>
<tr>
<td>target_name</td>
<td>Name of the target device.</td>
</tr>
<tr>
<td>clock_info</td>
<td>Information about the monotonic clock used for timestamps.</td>
</tr>
<tr>
<td>num_task_entries</td>
<td>Length of the array of task entries in task_entries. If this value is greater than BM_CONFIG_MAX_TASK_INFOS, it is truncated.</td>
</tr>
<tr>
<td>task_entries</td>
<td>The task information descriptors. Can be NULL.</td>
</tr>
<tr>
<td>num_mmap_entries</td>
<td>The length of the array of mmap entries in mmap_entries. If this value is greater than BM_CONFIG_MAX_MMAP_LAYOUT, it is truncated.</td>
</tr>
<tr>
<td>mmap_entries</td>
<td>The mmap image layout descriptors. Can be NULL.</td>
</tr>
<tr>
<td>timer_sample_rate</td>
<td>Timer based sampling rate in Hz. Zero indicates no timer based sampling (assumes max. 4GHz sample rate). This value is informative only, and is used for reporting the timer frequency in the Arm Streamline UI.</td>
</tr>
</tbody>
</table>

### Return value

| BM_TRUE | On success. |
| BM_FALSE | On failure. |

--- Note ---

If BM_CONFIG_MAX_TASK_INFOS ≤ 0, num_task_entries and task_entries are not present.

If BM_CONFIG_MAX_MMAP_LAYOUTS ≤ 0, num_mmap_entries and mmap_entries are not present.

---

**barman_enable_sampling**

```c
void barman_enable_sampling(void);
```

**Description** Enables sampling. Call once all PMUs are enabled and the data store is configured.

**barman_disable_sampling**

```c
void barman_disable_sampling(void);
```

**Description** Disables sampling without reconfiguring the PMU. Sampling can be resumed by a call to barman_enable_sampling.
### Description
Reads the configured PMU counters for the current processing element and inserts them into the data store. Can also insert a program counter record using the return address as the PC sample.

### Parameter
- **sample_return_address**
  - `BM_TRUE` to sample the return address as PC, `BM_FALSE` to ignore.

### Note
- The **Call Paths** view displays the PC values. This view is blank if the application does not call `barman_sample_counters` with `sample_return_address == BM_TRUE`, or `barman_sample_counters_with_program_counter` with `pc != BM_NULL`.
- Application code that is not doing periodic sampling typically calls this function with `sample_return_address == BM_TRUE`.
- This function must be run on the processing element for the PMU that it intends to sample from, and it must not be migrated to another processing element for the duration of the call. This is necessary as it needs to program the per-processing element PMU registers.

### Description
Reads the configured PMU counters for the current processing element and inserts them into the data store.

### Parameter
- **pc**
  - The PC value to record. The PC entry is not inserted if `pc == BM_NULL`.

### Note
- The **Call Paths** view displays the PC values. This view is blank if the application does not call `barman_sample_counters_with_program_counter` with `pc != BM_NULL`, or `barman_sample_counters` with `sample_return_address == BM_TRUE`.
- A periodic interrupt handler typically calls this function, with `pc == <the_exception_return_address>`. This function must be run on the processing element for the PMU that it intends to sample from, and it must not be migrated to another processing element for the duration of the call. This is necessary as it will need to program the per-processing element PMU registers.

The following functions are available if `BM_CONFIG_MAX_TASK_INFOS > 0`:

### barman_add_task_record

```c
bm_bool barman_add_task_record(bm_uint64 timestamp, const struct bm_protocol_task_info *task_entry);
```

<table>
<thead>
<tr>
<th>Description</th>
<th>Adds a new task information record.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameters</strong></td>
<td></td>
</tr>
<tr>
<td><code>timestamp</code></td>
<td>The timestamp at which the record is inserted.</td>
</tr>
<tr>
<td><code>task_entry</code></td>
<td>The new task entry.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td></td>
</tr>
<tr>
<td><code>BM_TRUE</code></td>
<td>On success.</td>
</tr>
<tr>
<td><code>BM_FALSE</code></td>
<td>On failure.</td>
</tr>
</tbody>
</table>
barman_record_task_switch

```c
void barman_record_task_switch(enum bm_task_switch_reason reason);
```

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Records that a task switch has occurred. Call this function after the new task is made the current task, such that a call to <code>barman_ext_get_current_task_id</code> would return the new task ID. For example, insert it into the scheduler of an RTOS just after the new task is selected to record the task switch.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>reason</td>
</tr>
<tr>
<td>Reason for the task switch.</td>
</tr>
</tbody>
</table>

--- Note ---
Call after the task switch has occurred so that `bm_ext_get_current_task` returns the `task_id` of the switched to task.

The following function is available if `BM_CONFIG_MAX_MMAP_LAYOUTS > 0`:

barman_add_mmap_record

```c
bm_bool barman_add_mmap_record(bm_uint64 timestamp, const struct bm_protocol_mmap_layout * mmap_entry);
```

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adds a new mmap information record.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
</tr>
<tr>
<td>The timestamp at which the record is inserted.</td>
</tr>
<tr>
<td>mmap_entry</td>
</tr>
<tr>
<td>The new mmap entry.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM_TRUE</td>
</tr>
<tr>
<td>On success.</td>
</tr>
<tr>
<td>BM_FALSE</td>
</tr>
<tr>
<td>On failure.</td>
</tr>
</tbody>
</table>

Data types associated with the public API functions:

bm_protocol_clock_info

```c
struct bm_protocol_clock_info {
    bm_uint64 timestamp_base;
    bm_uint64 timestamp_multiplier;
    bm_uint64 timestamp_divisor;
    bm_uint64 unix_base_ns;
};
```
**Description**
Defines information about the monotonic clock used in the trace. Timestamp information is stored in arbitrary units within samples. This reduces the overhead of making the trace by removing the need to transform the timestamp into nanoseconds at the point the sample is recorded. The host expects timestamps to be in nanoseconds. The arbitrary timestamp information is transformed to nanoseconds according to the following formula:

\[
bm\text{-}uint64\; nanoseconds = \frac{((timestamp - timestamp\_base) \times timestamp\_multiplier)}{timestamp\_divisor};
\]

Therefore for a clock that already returns time in nanoseconds, `timestamp\_multiplier` and `timestamp\_divisor` should be configured as 1 and 1. If the clock counts in microseconds then the multiplier and divisor should be set to 1000 and 1. If the clock counts at a rate of \(n\) Hz, then the multiplier should be set to 1000000000 and the divisor to \(n\).

**Members**
- **timestamp\_base**
  The base value of the timestamp such that this value is zero in the trace.
- **timestamp\_multiplier**
  The clock rate ratio multiplier.
- **timestamp\_divisor**
  The clock rate ratio divisor
- **unix\_base\_ns**
  The unix timestamp base value, in nanoseconds, such that a `timestamp\_base` maps to a `unix\_base unix` time value.

### bm\_protocol\_task\_info

```c
struct bm_protocol_task_info {
    bm_task_id_t task_id;
    const char * task_name;
};
```

**Description**
A task information record. Describes information about a unique task within the system.

**Members**
- **task_id**
  The task ID.
- **task_name**
  The name of the task.

### bm\_protocol\_mmap\_layout

```c
struct bm_protocol_mmap_layout {
    #if BM_CONFIG_MAX_TASK_INFOS > 0
    bm_task_id_t task_id;
    #endif
    bmuintptr base_address;
    bmuintptr length;
    bmuintptr image_offset;
    const char * image_name;
};
```

**Description**
An MMAP layout record. Describes the position of an executable image (or section thereof) in memory, allowing the host to map PC values to the appropriate executable image.

**Members**
- **task_id**
  The task ID to associate with the map.
- **base_address**
  The base address of the image, or image section.
- **length**
  The length of the image, or image section.
- **image_offset**
  The image section offset.
- **image_name**
  The name of the image.
**bm_task_switch_reason**

```c
enum bm_task_switch_reason {
    BM_TASK_SWITCH_REASON_PREEMPTED = 0,
    BM_TASK_SWITCH_REASON_WAIT = 1
};
```

<table>
<thead>
<tr>
<th>Description</th>
<th>Reason for a task switch.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Members</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BM_TASK_SWITCH_REASON_PREEMPTED</td>
</tr>
<tr>
<td></td>
<td>Thread is preempted.</td>
</tr>
<tr>
<td></td>
<td>BM_TASK_SWITCH_REASON_WAIT</td>
</tr>
<tr>
<td></td>
<td>Thread is blocked waiting, for example on IO.</td>
</tr>
</tbody>
</table>

WFI/WFE event handling functions:

**barman_wfi**

```c
void barman_wfi(void);
```

**Description**
Wraps WFI instruction and sends events before and after the WFI to log the time in WFI. This function is safe to use in place of the usual WFI `asm` instruction, as it degenerates to just a WFI instruction when barman is disabled.

**barman_wfe**

```c
void barman_wfe(void);
```

**Description**
Wraps WFE instruction and sends events before and after the WFE to log the time in WFE. This function is safe to use in place of the usual WFE `asm` instruction as it degenerates to just a WFE instruction when barman is disabled.

**barman_before_idle**

```c
void barman_before_idle(void);
```

**Description**
Call before a WFI/WFE, or other similar halting event, to log entry into the paused state. Can be used in situations where `barman_wfi()`/`barman_wfe()` is not suitable.

--- Note ---
- You must use `barman_before_idle` in a pair with `barman_after_idle()`.
- Using `barman_wfi()`/`barman_wfe()` is preferred in most cases, as it takes care of calling the before and after functions.

**barman_after_idle**

```c
void barman_after_idle(void);
```

**Description**
Call after a WFI/WFE, or other similar halting event, to log exit from the paused state. Can be used in situations where `barman_wfi()`/`barman_wfe()` is not suitable.

--- Note ---
- You must use `barman_after_idle` in a pair with `barman_before_idle()`.
- Using `barman_wfi()`/`barman_wfe()` is preferred in most cases, as it takes care of calling the before and after functions.

Functions for recording textual annotations:

**barman_annotate_channel**

```c
void barman_annotate_channel(bm_uint32 channel, bm_uint32 color, const char * string)
```
<table>
<thead>
<tr>
<th>Description</th>
<th>Adds a string annotation with a display color, and assigns it to a channel.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>channel</td>
</tr>
<tr>
<td></td>
<td>The channel number.</td>
</tr>
<tr>
<td></td>
<td>color</td>
</tr>
<tr>
<td></td>
<td>The annotation color from bm_annotation_colors.</td>
</tr>
<tr>
<td></td>
<td>text</td>
</tr>
<tr>
<td></td>
<td>The annotation text, or null to end the previous annotation.</td>
</tr>
</tbody>
</table>

Note

Annotation channels and groups are used to organize annotations within the threads and processes section of the Timeline view. Each annotation channel appears in its own row under the thread. Channels can also be grouped and displayed under a group name, using the barman_annotate_name_group function.

barman_annotate_name_channel

void barman_annotate_name_channel(bm_uint32 channel, bm_uint32 group, const char * name)

<table>
<thead>
<tr>
<th>Description</th>
<th>Defines a channel and attaches it to an existing group.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>channel</td>
</tr>
<tr>
<td></td>
<td>The channel number.</td>
</tr>
<tr>
<td></td>
<td>group</td>
</tr>
<tr>
<td></td>
<td>The group number.</td>
</tr>
<tr>
<td></td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>The name of the channel.</td>
</tr>
</tbody>
</table>

Note

The channel number must be unique within the task.

barman_annotate_name_group

void barman_annotate_name_group(bm_uint32 group, const char * name)

<table>
<thead>
<tr>
<th>Description</th>
<th>Defines an annotation group.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>group</td>
</tr>
<tr>
<td></td>
<td>The group number.</td>
</tr>
<tr>
<td></td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>The name of the group.</td>
</tr>
</tbody>
</table>

Note

The group identifier, group, must be unique within the task.

barman_annotate_marker

void barman_annotate_marker(bm_uint32 color, const char * text)
Adds a bookmark with a string and a color to the Timeline and Log views. The string is displayed in the Timeline view when you hover over the bookmark, and in the Message column in the Log view.

**Parameters**

- **color**: The marker color from `bm_annotation_colors`.
- **text**: The marker text, or null for no text.

<table>
<thead>
<tr>
<th><strong>bm_annotation_colors</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
</tr>
</tbody>
</table>

### 13.9.4 External functions to implement

You must provide the following external functions.

#### barman_ext_get_timestamp

```c
extern bm_uint64 barman_ext_get_timestamp(void);
```

**Description**
Reads the current sample timestamp value, which must be provided for the time at the point of the call. The timer must provide monotonically incrementing values from an implementation defined start point. The counter must not overflow during the period that it is used. The counter is in arbitrary units. The mechanism for converting those units to nanoseconds is described as part of the protocol data header.

**Return value**
The timestamp value in arbitrary units.

The following functions have weak linkage implementations that can be overridden if necessary:

#### barman_ext_disable_interrupts_local

```c
extern bm_uintptr barman_ext_disable_interrupts_local(void);
```

**Description**
Disables interrupts on the local processor only. Used to allow atomic accesses to certain resources, for example PMU counters.

**Return value**
The current interrupt enablement status value. This value must be preserved and passed to `barman_ext_enable_interrupts_local` to restore the previous state.

--- Note ---
A weak implementation of this function is provided that modifies DAIF on AArch64, or CPSR on AArch32.

#### barman_ext_enable_interrupts_local

```c
extern void barman_ext_enable_interrupts_local(bm_uintptr previous_state);
```

**Description**
Enables interrupts on the local processor only.

**Parameter**

- **previous_state**: The value that was previously returned from `barman_ext_disable_interrupts_local`

--- Note ---
A weak implementation of this function is provided that modifies DAIF on AArch64, or CPSR on AArch32.

The following functions must be defined if `BM_CONFIG_MAX_CORES > 1`:
### Description

Given the MPIDR register, returns a unique processor number. The implementation must return a value between 0 and \( N \), where \( N \) is the maximum number of processors in the system. For any valid permutation of the arguments, a unique value must be returned. This value must not change between successive calls to this function for the same argument values.

```c
bm_uint32 barman_ext_map_multiprocessor_affinity_to_core_no(bm_uintptr mpidr) {
    return (mpidr & 0x03) + ((mpidr >> 6) & 0x4);
}
```

**Parameter**
- `mpidr`: The value of the MPIDR register.

**Return value**
- The processor number.

---

### Note

This function need only be defined when `BM_CONFIG_MAX_CORES > 1`

---

### Description

Given the MPIDR register, return the appropriate cluster number. Cluster IDs should be numbered from 0 to \( N \), where \( N \) is the number of clusters in the system.

```c
bm_uint32 barman_ext_map_multiprocessor_affinity_to_cluster_no(bm_uintptr mpidr) {
    return ((mpidr >> 8) & 0x1);
}
```

**Parameter**
- `mpidr`: The value of the MPIDR register.

**Return value**
- The cluster number.

---

### Note

This function need only be defined when `BM_CONFIG_MAX_CORES > 1`

---

The following function must be defined if `BM_CONFIG_MAX_TASK_INFOS > 0`:

### Description

Returns the current task ID.

```c
extern bm_task_id_t barman_ext_get_current_task_id(void);
```
The following functions must be defined if `BM_CONFIG_ENABLE_LOGGING != 0`:

**barman_ext_log_info**

```c
void barman_ext_log_info(const char * message, ...);
```

<table>
<thead>
<tr>
<th>Description</th>
<th>Prints an info message.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>message</td>
</tr>
</tbody>
</table>

**barman_ext_log_warning**

```c
void barman_ext_log_warning(const char * message, ...);
```

<table>
<thead>
<tr>
<th>Description</th>
<th>Prints a warning message.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>message</td>
</tr>
</tbody>
</table>

**barman_ext_log_error**

```c
void barman_ext_log_error(const char * message, ...);
```

<table>
<thead>
<tr>
<th>Description</th>
<th>Prints an error message.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>message</td>
</tr>
</tbody>
</table>

The following function must be defined if `BM_CONFIG_ENABLE_DEBUG_LOGGING != 0`:

**barman_ext_log_debug**

```c
void barman_ext_log_debug(const char * message, ...);
```

<table>
<thead>
<tr>
<th>Description</th>
<th>Prints a debug message.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>message</td>
</tr>
</tbody>
</table>
Chapter 14
Fastline

Describes the Fastline Wizard, which automates the process of profiling Fast Models with Arm Streamline.

It contains the following sections:
• *14.1 Fastline overview* on page 14-265.
• *14.2 Fastline Wizard* on page 14-266.
14.1 Fastline overview

Fastline is a plug-in for profiling Linux systems and applications, and bare-metal applications, running on Fast Models. The Fastline Wizard in Arm Streamline automates the manual process of the plug-in.

Note

See 1.1 Features of Arm Streamline editions on page 1-17 to check if your edition includes the Fastline Wizard.

The wizard produces one of the following outputs, depending on the options you select:
- An APC capture that Arm Streamline analyzes and opens as a report.
- A launching script and a set of configuration files for the Fastline plug-in that you can run later.

The Fastline wizard is a separate Eclipse plug-in called com.arm.streamline.fastlinewizard. You can install it in Eclipse separately from Arm Streamline without losing any functionality.

Related information

14.2 Fastline Wizard

The following procedure shows you how to complete the Fastline Wizard in Arm Streamline. You choose whether to start Fastline at the end of the wizard, or generate a launching script to run later.

Prerequisites

- Ensure that you have Fast Models version 11.1.
- Select Window > Preferences > Fastline Preferences. Enter the locations of the Executables Directory and the Plugins Directory, or browse for them in your filesystem. The Plugins Directory must contain FastlineTrace.dll and FastlineTrace_counter_defaults.json.
- Ensure that you have the correct licenses for Fast Models and Arm Streamline.
- For Linux systems, you need the following:
  - The Linux kernel sources that are used for Linux image compilation.
  - System.map, which the kernel build produces.
  - The .config file that the kernel build produces.
  - The vmlinux image that the kernel build produces, with the debug information sections.
  - The PAGE_SHIFT parameter that is used.
  - Whether the OS architecture of the compiled kernel is a 32-bit or a 64-bit architecture.
  - The bootloader, RAMDisk, filesystem, device tree, and other binaries for booting Linux on Fast Models.
  - The command-line arguments to make the model load and run Linux.
- For bare-metal applications, you need the compiled ELF image with the application, symbols, and debug information in it.

Procedure

1. Select Streamline > Fastline Wizard to open the Fastline Wizard dialog.

2. Select one model to use. The wizard lists the models in the specified directory, and models provided with Arm Development Studio in sw/models/bin.

Note

If the Fast Models installation path is not set in the Arm Streamline preferences, the wizard requests this information. Enter the locations of your Fast Models installation and plug-ins directories then click Next.
3. Optionally enter extra command-line arguments for the executable in the text field at the bottom of the dialog.

4. Click Next.

5. If the Fast Models executable cannot find the license, the wizard requests this information. Enter your license information.

6. Click Validate.
   
   If your license information is valid, you can progress. Otherwise, the wizard displays an error message and you cannot continue.

7. Click Next.

8. Right-click in the System Type column for each core, and select the corresponding system type from the drop-down list. The possible values are Linux, Bare Metal, and Not Used.

   The wizard supports running the following configurations:
   
   • One Linux system.
   • One bare-metal application.
   • One Linux system and one bare-metal application.

   By default, the wizard selects Linux for A-class processors, and Bare Metal for M- and R-class processors.

9. Click Next.

10. Provide the extra information that is required for the selected system type.

    • For Linux systems, enter the location of the following items:
      
      — vmlinux.o
      — System.map
      — PAGE_SHIFT Value

    You must also select whether the system is 32-bit or 64-bit.

    Enter this information manually or, to automatically complete these fields, click Scan Linux Source Tree…. If you use the autoscan function, enter the location of the Linux source tree when
prompted. The wizard scans this location for the required files and displays warning messages for any it cannot find.

- For bare-metal applications, enter the location of the program image ELF file.

11. Click **Next**.
12. Select counters from the list of data sources for the system. Click Add to add them to the list of counters to be profiled.

The Basic Mode list contains counters that have the complete set of properties that are required for JSON counter configuration.

The Advanced Mode list contains all the counters. You must specify any information missing from the JSON counter configuration to be able to use the counters on this list. A dialog opens when you select an Advanced Mode counter for entering the missing information.

--- Note ---

You can select multiple counters to add in Basic Mode but you can only add one counter at a time in Advanced Mode. This restriction is because the Counter Configuration dialog opens for each counter you select.

13. Click Next.

14. Check the following capture details and adjust as required:
   - Capture output location.
   - Sampling frequency.
   - Capture start time.
   - Capture end time.

![Figure 14-5 Adjust capture details as required.]

15. Click Next.

16. Select whether you want to start profiling now or generate a launch script for profiling later.
17. For Linux applications, specify a working directory to ensure that the command-line options that you provided to boot Linux properly resolve as relative paths. The working directory is usually the directory that contains the binary files.

18. Click **Finish**.

The wizard produces the following output:

- A platform-specific script that starts Fast Models with the Fastline plug-in and all the specified configuration files.
- `fastline.xml`, a file that contains all your inputs to the wizard.
- For Linux systems, the wizard produces the Linux kernel data structure offsets.
- For bare-metal systems, the wizard produces a CSV file that contains the symbol names and addresses that Fastline requires.

If requested, Fast Models launches and Arm Streamline begins profiling. When you close Fast Models, Arm Streamline analyzes the capture data and opens the report.

To process the ELF image and fully populate the **Call Paths** view, re-analyze the capture with the image selected.

**Note**

Fastline creates an APC file when the Fast Models execution ends. This file is created in the default location for Arm Streamline, unless you specify otherwise. If you choose a different location, import the data into Arm Streamline by clicking **Import Capture File(s)**... and navigating to the chosen directory.

---

**Related information**

*Booting Linux on a model*
Chapter 15
Using Arm Streamline on the Command Line

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

It contains the following sections:

• 15.1 Opening Arm Streamline-enabled command prompts or shells on page 15-272.
• 15.2 Arm Streamline command-line options on page 15-273.
• 15.3 Outputting command-line data to a file on page 15-277.
• 15.4 Exporting the Heat Map from the command-line on page 15-278.
• 15.5 Accessing the Bare-Metal generation mechanism from the command line on page 15-279.
• 15.6 Importing a trace from the command-line on page 15-280.
15.1 Opening Arm Streamline-enabled command prompts or shells

Using Arm Streamline outside of the user interface allows you to perform automated captures and generate text-based reports which you can export into a spreadsheet application.

The way that you open Arm Streamline-enabled command prompts or shells depends on your operating system:

- On Windows, select Start > All Programs > Arm DS > Arm DS IDE Command Prompt .
- On Linux, add the $DS_install_directory/bin location to your PATH environment variable then open a Unix bash shell.

Related tasks
15.3 Outputting command-line data to a file on page 15-277

Related references
15.2 Arm Streamline command-line options on page 15-273
15.2 Arm Streamline command-line options

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

Arm Streamline command-line modes

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

```
streamline <mode> [options] <file...>
```

Use either of the following options directly after streamline on the command line:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
</table>
| -capture | This mode initiates a capture session. You must specify a valid `session.xml` file with this option. The `session.xml` file defines your target hardware and the parameters of the capture session. For example:  
```
streamline -capture session.xml
```

To create a `session.xml` file, enter your settings and then use the Export... option in the Capture & Analysis Options dialog box. |
| -report | This mode reads data from a capture and outputs it to your console or to a file. You can use additional options to filter and format the data. You must enter a valid .apc capture after the declaration of report mode. For example:  
```
streamline -report threads_001.apc
```
| -merge | This mode merges multiple .apc capture files. You can optionally specify the timing offset between captures and specify the merged output .apc file name. |

![Figure 15-1  Functions report generated using report mode](image)

Options common to all modes

The following options are available in all modes:
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, -?, -help</td>
<td>Outputs help information to the console, listing each mode and option, and the required syntax.</td>
</tr>
<tr>
<td>-v, -version</td>
<td>Displays the program version information.</td>
</tr>
</tbody>
</table>

**Capture mode options**

The following options are available only in capture mode:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-duration &lt;seconds&gt;</td>
<td>Sets the number of seconds that you want the capture to last. The capture automatically terminates when it reaches that duration. For example, the following command triggers a 60 second capture session: <code>streamline -capture -duration 60 session.xml</code></td>
</tr>
<tr>
<td>-c, -config &lt;configuration.xml&gt;</td>
<td>Specifies the location of the configuration file to send to the target.</td>
</tr>
<tr>
<td>-retrieve-image &lt;regex&gt;</td>
<td>Specifies a regex that is used to match the processes to retrieve from the target.</td>
</tr>
<tr>
<td>-o, -output &lt;filename&gt;</td>
<td>Specifies the name of the .apc capture file to create.</td>
</tr>
<tr>
<td>-username &lt;string&gt;</td>
<td>Specifies the username to use when retrieving images from the target.</td>
</tr>
<tr>
<td>-password &lt;string&gt;</td>
<td>Specifies the password to use when retrieving images from the target.</td>
</tr>
<tr>
<td>-include-libs</td>
<td>Fetches the relevant libraries when retrieving images from the target.</td>
</tr>
<tr>
<td>-p, -pmus &lt;pmus.xml&gt;</td>
<td>Specify the path to your pmus.xml file.</td>
</tr>
<tr>
<td>-e, -events &lt;events.xml&gt;</td>
<td>Specify the path to your events.xml file.</td>
</tr>
</tbody>
</table>

**Report mode options**

The following options are unique to report mode:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-all</td>
<td>Outputs the contents of the Timeline, Call Paths, Functions, and Log views, including bookmarks and custom activity map tables. -all is the default option.</td>
</tr>
<tr>
<td>-callpath</td>
<td>Outputs the contents of the table data of the Call Paths view. Subordinate functions are indented.</td>
</tr>
<tr>
<td>-o, -output &lt;output directory&gt;</td>
<td>Sends all report text into files in the specified directory. If multiple tables are specified for export, each table is written to &lt;output directory&gt; in a new file with an appropriate name. If multiple captures are specified, each capture is output to a subdirectory of &lt;output directory&gt; with a name matching the capture.</td>
</tr>
<tr>
<td>-function</td>
<td>Outputs the contents of the Functions view.</td>
</tr>
<tr>
<td>-log</td>
<td>Outputs the contents of the Log view.</td>
</tr>
<tr>
<td>-timeline</td>
<td>Outputs the contents of the Timeline view.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-cam</td>
<td>Outputs the contents of custom activity map tables.</td>
</tr>
<tr>
<td></td>
<td>Note</td>
</tr>
<tr>
<td></td>
<td>To export custom activity map data, the report must contain custom activity map</td>
</tr>
<tr>
<td></td>
<td>annotations.</td>
</tr>
<tr>
<td>-heatmap &quot;&lt;source&gt;&quot;</td>
<td>Outputs the contents of the Heat Map. <code>&lt;source&gt;</code> must match the name of the activity</td>
</tr>
<tr>
<td></td>
<td>source for the Heat Map in the capture.</td>
</tr>
<tr>
<td>-opencl</td>
<td>Outputs the contents of the OpenCL table.</td>
</tr>
<tr>
<td></td>
<td>Note</td>
</tr>
<tr>
<td></td>
<td>To export OpenCL data, the report must contain OpenCL annotations.</td>
</tr>
<tr>
<td>-bookmarks</td>
<td>Outputs all bookmarks stored in a capture, listing the time index location of the</td>
</tr>
<tr>
<td></td>
<td>bookmark and its text.</td>
</tr>
<tr>
<td>-process &lt;regex&gt;##&lt;PID&gt;</td>
<td>A regex matching the process for which to filter the timeline data and its PID,</td>
</tr>
<tr>
<td></td>
<td>separated by a hash.</td>
</tr>
<tr>
<td>-template &lt;filename&gt;</td>
<td>Use this option with a valid chart configuration template to list data for your</td>
</tr>
<tr>
<td></td>
<td>customized charts on the command line. Use the Switch and manage templates</td>
</tr>
<tr>
<td></td>
<td>button in the Live or Timeline view to create a chart configuration template,</td>
</tr>
<tr>
<td></td>
<td>then use that file with the -template option.</td>
</tr>
<tr>
<td>-per_core</td>
<td>Outputs per-core data when used with the -timeline option.</td>
</tr>
<tr>
<td>-scale &lt;number&gt;</td>
<td>Sets the number of bins per second to use in the report, by default 1000.</td>
</tr>
<tr>
<td>-start &lt;seconds&gt;</td>
<td>Filters output data to start at the specified time within the timeline. For example if you enter 0.005 with this option, all data before the 5 millisecond mark is not included in the output.</td>
</tr>
<tr>
<td>-stop &lt;seconds&gt;</td>
<td>Filters output data to stop at the specified time within the timeline.</td>
</tr>
<tr>
<td>-bstart &lt;name&gt;</td>
<td>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.</td>
</tr>
<tr>
<td>-bstop &lt;name&gt;</td>
<td>Filter the data to end at the first bookmark with the provided name.</td>
</tr>
<tr>
<td>-format &lt;space</td>
<td>tab</td>
</tr>
</tbody>
</table>

You can define multiple report types using these options. For example, to output the Call Paths and Functions data from the thread_001.apc capture, enter:

```
streamline -report -callpath -function thread_001.apc
```
Merge mode options

The following options are unique to merge mode:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-synchronize &lt;timestamp</td>
<td>none</td>
</tr>
<tr>
<td>-o, -output &lt;output&gt;</td>
<td>The name of the .apc capture file to create.</td>
</tr>
</tbody>
</table>

Related tasks

15.1 Opening Arm Streamline-enabled command prompts or shells on page 15-272
15.3 Outputting command-line data to a file on page 15-277
15.3 **Outputting command-line data to a file**

Because the reports generated by Streamline can be very large, it can be useful to send the output to a file. You can then import data from this file to your favorite spreadsheet application.

Specify the output file to the `streamline` command using the `-o` option, for example:

```
streamline -report -timeline capture_001.apc -o output.txt
```

This creates a file with the given name, if it does not already exist, and outputs the data to the new file instead of to the command window. If the file already exists, it is overwritten.

![Timeline view data output to a text file](image)

**Figure 15-2** Timeline view data output to a text file

**Related tasks**

15.1 *Opening Arm Streamline-enabled command prompts or shells* on page 15-272

**Related references**

15.2 *Arm Streamline command-line options* on page 15-273
15.4 Exporting the Heat Map from the command-line

Export the data that is contained in the **Heat Map** mode of the details panel to a text file.

Enter the following at the command-line:

```
./streamline -report <apc-name> -heatmap "<source>"
```

`<source>` is the activity source that is used in the analysis report, for example CPU Activity, or GPU Fragment. This parameter is optional. If you do not provide the activity source, CPU Activity is exported by default.

The following options are also available:

- `-o`, `-output <filename>`
- `-process <regex>#$<PID>`
- `-scale <number>`
- `-start <seconds>`
- `-stop <seconds>`
- `-bstart <name>`
- `-bstop <name>`
- `-format <space|tab|csv>`

**Note**

The **Heat Map** data is not included in `-all`, and therefore not exported when executing the following command:

```
./streamline -report -all
```

**Related concepts**

- 6.6.2 Heat Map mode on page 6-138
- 6.6.5 Selecting the activity source on page 6-142

**Related references**

- 15.2 Arm Streamline command-line options on page 15-273
15.5 Accessing the Bare-Metal generation mechanism from the command line

You can pass the configured, and optionally modified, XML file produced in the Bare-Metal configuration process to the command line. The generator then outputs the source and header files.

Enter `streamline -generate-bare-metal-agent <options>`

The following command-line arguments are available:

- `-c, -config <config.xml>`
  The configuration file to use to generate the bare-metal agent.

- `-p, -pmus <pmus.xml>`
  Specify the path to your `pmus.xml` file.

- `-e, -events <events.xml>`
  Specify the path to your `events.xml` file.

- `-o, -output <output_path>`
  Specify the output path to where the generated files will be written.

Related tasks

13.2.1 Configuring barman on page 13-229
15.6 Importing a trace from the command-line

Import a trace file and create a capture from it.

Enter the following command:

```
./streamline <mode> <mode_options>
```

The following importing modes are available:

- **-import-apc-zip <file>**
  - Import zipped APC files.

- **-import-perf <file>**
  - Import perf data files.

- **-import-barman-raw <file>**
  - Import a bare-metal raw file.

- **-import-stm <file>**
  - Import a bare-metal trace that was captured through STM. Arm Streamline converts the trace file into a bare-metal raw file.

- **-import-itm <file>**
  - Import a bare-metal trace that was captured through ITM. Arm Streamline converts the trace file into a bare-metal raw file.

- **-import-etm-dt <file>**
  - Import a bare-metal trace that was captured through ETM. Arm Streamline converts the trace file into a bare-metal raw file.

- **-import-socd**
  - Import SoC Designer profiles.

The following options are available to importing modes:

- **-o, --output <output_path>**
  - The parent directory where the capture will be created.

- **-barman-xml <path_to_generated_barman_xml_file>**
  - Provide 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.

**Related concepts**

3.8 Importing perf data on page 3-71

**Related references**

13.2 Profiling with on-target RAM buffer on page 13-229
13.3 Profiling with System Trace Macrocell on page 13-235
13.4 Profiling with Instrumentation Trace Macrocell on page 13-238
Chapter 16
JIT and Python Profiling Support

Describes how to profile Just In Time (JIT) and Python execution with Arm Streamline.

It contains the following sections:

• *16.1 JIT profiling support* on page 16-282.
• *16.2 Python profiling support* on page 16-284.
16.1 JIT profiling support

You can profile Just In Time (JIT) compiled language execution with Arm Streamline by importing jitdump files that are generated by JIT engines.

Common supported languages are: Node.js, Chrome V8 Javascript, and Java.

To profile JIT execution, you can trace your program by recording the execution using the `perf record` command on a supported JIT engine that produces `jitdump` data. You can then directly import the `perf.data` file into Arm Streamline.

You can specify the `jitdump` file paths in the Analyze dialog box, JIT Dump Files tab, shown in the following figure.

![Analyze dialog box (JIT Dump Files)](image)

Arm Streamline processes the files to extract the source and line information, and shows the generated machine code in the Arm Streamline code view.

Note

- Only imported `perf` recordings are supported because Node.js, JVM, and other VMs use a CLOCK_MONOTONIC clock by default. However, `gatord` uses a CLOCK_MONOTONIC_RAW
clock. If you use gatord, you must manually modify Node.js or the JVM Tool Interface library, that is shipped with perf, to use CLOCK_MONOTONIC_RAW.

- You must pass -k CLOCK_MONOTONIC to perf record when profiling using libperf-jvmti.so. libperf-jvmti.so uses CLOCK_MONOTONIC_RAW but, by default, perf does not use the same clock.
- JIT profiling is not compatible with gator.ko. You can only use user space capturing.

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=1000000/, 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 ...
```


**JVM**

Use the libperf_jvmti.so agent library which is provided in source form as part of the perf tools sources. See https://patchwork.kernel.org/patch/8236901/.

- 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 concepts**

3.8 Importing perf data on page 3-71

**Related information**

Streamline and JITed code.
16.2 Python profiling support

You can profile Python execution at the module level, or you can add trace functions to your code. Arm Streamline uses the tracing functions that are provided in the Python sys module to profile Python script execution.

Arm Streamline includes a Python module, gator.py to trace Python execution.

Note

gator.py works with Python 2 and Python 3 but is only tested with CPython.

This section contains the following subsections:

• 16.2.1 Profiling a Python module on page 16-284.
• 16.2.2 Profiling individual functions in a Python module on page 16-284.
• 16.2.3 Attaching Python script sources for analysis on page 16-285.
• 16.2.4 Viewing scripts in analysis on page 16-286.

16.2.1 Profiling a Python module

To profile all functions in a Python module, you can run it in the gator.py module.

Procedure

1. Start Arm Streamline to capture trace information.
2. Start your Python module:
   
   Use the -m argument to run gator.py. gator.py accepts your Python module as an argument.
   
   ```
   python -m gator [gator_args...] <user_module> [user_module_args...]
   ```

   gator.py accepts the following arguments:
   
   - **-h**  Prints a description of the options.
   - **-d**  Enable debug messages.
   - **-t**  By default, gator.py uses profiling mode. When this flag is set, trace mode is used which traces line numbers rather than just function call entry and exits.
   
   <user_module>  The Python module to execute followed by its arguments.

16.2.2 Profiling individual functions in a Python module

You can add calls to gator.py profiling functions in your own code.

You can profile specific functions by creating a GatorProfiler object and calling the gator_profiler.runctx() or gator_profiler.call() functions as appropriate:

```python
import gator

g = gator.GatorProfiler()

g.runctx(some_function, args=(arg1, arg2, arg3), kwargs={'kwarg1':kwarg1})
```

Function some_function() is then called and profiled.

When you are tracing individual functions, gator.py runs in the following modes:

Profiling mode

Uses the Python sys.setprofile() function. It provides lower overhead analysis of a python script at the expense of only tracing function calls rather than individual lines of code. When this mode is enabled, you can see which functions were called and how often. The execution of lines within a function, or how often they were executed, cannot be viewed.
Trace mode

Uses the Python `sys.settrace()` function. It provides much more detailed analysis at the expense of a significantly larger profiling overhead. In this mode, individual lines are profiled.

In both modes, a full call-stack is collected.

The `GatorProfiler` constructor is defined as:

```
def __init__(self, trace_mode=False, debug_enabled=False, ignored_paths=[])
```

The arguments are:

- `trace_mode` When FALSE, profiling mode is used. When TRUE, trace mode is used.
- `debug_enabled` When TRUE, debug messages are output to `stderr`.
- `ignored_paths` A list of absolute paths (as strings) to any scripts that `gator.py` must not profile.

Note

You must start the Arm Streamline capture session before running your Python module.

16.2.3 Attaching Python script sources for analysis

When you are analyzing the capture, you can specify script search directories where Arm Streamline looks for Python script sources.

You can use the script search paths tab to specify locations where Python scripts are searched for when Arm Streamline analyses the trace. The search locations must contain directory structures that match the script paths that are returned by `gator.py`. Scripts names are transferred from the target as absolute paths. For example, if you have profiled a script `/home/user/script.py`, you must provide a path to a directory `<search path>` so that `<search path>/home/user/script.py` points to the file that you want to analyze.

![Figure 16-2 Analyze dialog box (Script Search Paths)](image-url)
**Downloading scripts from the target**

When the capture completes, you must select and add the appropriate Python process in the Image Download dialog box. See 6.5 Image download on page 6-135. The download includes both the Python interpreter and the Python scripts traced.

### 16.2.4 Viewing scripts in analysis

Python function names are shown in the call paths view under the associated Python interpreter process. Python source code is shown in the code view.

Where a source file for a script is found, the names of the profiled functions are shown in the call paths view. Where a source file is not found, the entry is shown as: `<unknown code in /path/to/script.py>`.

![Figure 16-3 Python call paths view](image)

Python source files appear in the code view with markers indicating number of samples for each profiled line. If `gator.py` profiling mode is chosen, only lines containing function definition statements indicate the number of times the function was called. The following figure shows code that is executed in `gator.py` trace mode.
16.2 Python profiling support

Figure 16-4 Python code view

```python
import threading

def func(i):
    print("Thread %s %i,")
    for j in range(i, i**2):
        string = "%s" % (j,)
        if '31' in string:
            print(string)

def execute_test():
    print("Creating threads")
    ts = []
    for i in range(1, 20):
        target = func, name="T-%i", args=(i,)
        ts.append(t)
        t.start()
    print("Joining threads")
    for t in ts:
        t.join()
    print("Done")

if __name__ == '__main__':
    print("Starting")
    g = gator.GatorProfiler(False)
    g.call(execute test)
    #execute test()
    print("Finished")
```
Chapter 17
Troubleshooting Common Arm Streamline Issues

Describes how to troubleshoot some common Arm Streamline issues.

It contains the following sections:
- 17.1 Troubleshooting target setup issues on page 17-289.
- 17.2 Troubleshooting target connection issues on page 17-290.
- 17.3 Troubleshooting Android issues on page 17-292.
- 17.4 Troubleshooting gator issues on page 17-293.
- 17.5 Troubleshooting perf import issues on page 17-296.
- 17.6 Troubleshooting Graphics Analyzer Mode issues on page 17-298.
- 17.7 Troubleshooting Energy Probe issues on page 17-300.
- 17.8 Troubleshooting report issues on page 17-301.
17.1 Troubleshooting target setup issues

If you encounter a problem while setting up your target to work with Arm Streamline, consult the following table for solutions.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error message generated: jni/PerfGroup.cpp: In function 'int sys_perf_event_open(perf_event_attr*, pid_t, int, int, long unsigned int)': jni/PerfGroup.cpp:36:17: error: '__NR_perf_event_open' was not declared in this scope</td>
<td>Upgrade to a more recent version of the Android NDK.</td>
</tr>
</tbody>
</table>

Related tasks

2.5.2 Building the gator daemon yourself on page 2-40
## 17.2 Troubleshooting target connection issues

You might have problems when trying to start a capture session, for instance by pressing the **Start capture** button. Use these solutions to solve common target connection issues.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error message generated:</td>
<td>Make sure <code>gatord</code> is running on your target. Enter the following command in the shell of your target:</td>
</tr>
<tr>
<td>Unable to connect to the gator daemon at <code>target_address</code>.</td>
<td>`ps ax</td>
</tr>
<tr>
<td>Please verify that the target is reachable and that you are running</td>
<td>If this command returns no results, <code>gatord</code> is not active. Start it by navigating to the directory that contains <code>gatord</code> and entering the following command:</td>
</tr>
<tr>
<td>gator daemon v17 or later. Installation instructions can be found in:</td>
<td><code>sudo ./gatord &amp;</code></td>
</tr>
<tr>
<td><code>streamline/gator/README.md</code>. If connecting over WiFi, please try again</td>
<td>Try connecting to the target again.</td>
</tr>
<tr>
<td>or use a wired connection.</td>
<td>If <code>gatord</code> is active and you still receive this error message, try disabling any firewalls on your host machine that might be interfering with communication between it and the target.</td>
</tr>
<tr>
<td></td>
<td>In addition, if you are running Android on your target, make sure the ports are accessible by using the <code>adb forward</code> command, for example:</td>
</tr>
<tr>
<td></td>
<td><code>adb forward tcp:8080 tcp:8080</code></td>
</tr>
<tr>
<td>Error message generated:</td>
<td>Make sure that you have correctly entered the name or IP address of the target in <strong>Address</strong> field. If you have entered a name, try an IP address instead.</td>
</tr>
<tr>
<td>Unknown host</td>
<td></td>
</tr>
<tr>
<td>When using event-based sampling, Arm Streamline fails to find the PMU.</td>
<td>The PMU on your hardware might not be correctly configured to allow the processor interrupts necessary for Arm Streamline to use event-based sampling. Test on alternate hardware or disable event-based sampling in the <strong>Counter Configuration</strong> dialog box.</td>
</tr>
<tr>
<td>Problem</td>
<td>Solution</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
</tr>
</tbody>
</table>
| The target is running a firewall, which prevents Arm Streamline from connecting to `gatord`. | There are several possible ways to resolve this issue:  
  - Update the firewall to allow connections to `gatord`, which defaults to using port 8080.  
  - Use local captures.  
  - If the target accepts SSH connections, you can establish an SSH tunnel by using the `ssh` command on the host. For example:  
    ```bash  
    ssh user@target -L 8080:localhost:8080 -N  
    ```  
    In this example, replace `user` with the username to log in as and `target` with the hostname of the target. On the target, use `localhost` as the hostname.  
    __________ Note __________  
    An SSH tunnel requires additional processing on the target.  
    __________  
    - Reverse SSH tunnels are also possible by running `ssh` from the target to the host. For example:  
    ```bash  
    ssh user@host -R 8080:localhost:8080 -N  
    ``` |
| Delay between clicking the **Capture** button and the **Live** view displaying data. | Use the `gator.ko` kernel module.  
Setting `ftrace` to use a different clock can also help. Use one of the following commands:  
**Linux kernel 4.2 or later**  
```bash  
echo mono_raw > /sys/kernel/debug/tracing/trace_clock  
```  
**Older Linux kernel versions**  
```bash  
echo perf > /sys/kernel/debug/tracing/trace_clock  
``` |

**Related references**

- *[17.8 Troubleshooting report issues on page 17-301](#)*
- *[2.9 Capturing data locally on a target on page 2-47](#)*
### 17.3 Troubleshooting Android issues

Android has the following known issues:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>run-as</code> command fails on Android.</td>
<td>Make sure that the application is debuggable.</td>
</tr>
<tr>
<td>The following error is reported:</td>
<td></td>
</tr>
<tr>
<td><code>run-as: Could not set capabilities: Operation not permitted</code></td>
<td>The <code>run-as</code> command is broken. This error is a known issue on the Samsung Galaxy S7 with no known work-around.</td>
</tr>
<tr>
<td>Capture fails on startup, usually showing no events captured.</td>
<td>This problem usually indicates a failure to configure the <code>perf</code> API. Run the following command:</td>
</tr>
<tr>
<td></td>
<td><code>adb shell setprop security.perf_harden 0</code></td>
</tr>
<tr>
<td></td>
<td>Reduce the set of events that are captured or try capturing only a limited set of <code>perf</code> software events. This error can be caused by:</td>
</tr>
<tr>
<td></td>
<td>• Exceeding the limit for the number of open file descriptors.</td>
</tr>
<tr>
<td></td>
<td>• The target device does not have a correctly configured PMU driver.</td>
</tr>
<tr>
<td>Hardware counters read as zero.</td>
<td>This error is usually a sign of misconfigured PMU. It is not usually possible to work around.</td>
</tr>
<tr>
<td>Mali counters are not detected.</td>
<td>Try specifying <code>--mali-type</code> and optionally <code>--mali-device</code> command-line arguments when running <code>gatord</code>.</td>
</tr>
<tr>
<td>When running non-root on Android, <code>gatord</code> exits with the message:</td>
<td></td>
</tr>
<tr>
<td><code>Error creating server TCP socket</code></td>
<td>• Run <code>gatord -p uds ...</code> to enable use of UDS socket instead of TCP socket.</td>
</tr>
<tr>
<td></td>
<td>• Execute <code>adb forward tcp:&lt;some-port&gt; localabstract:streamline-data</code> on your host to configure port forwarding.</td>
</tr>
<tr>
<td></td>
<td>• Set the target address as <code>localhost:&lt;some-port&gt;</code> in the <strong>Target View Address Field</strong> text box.</td>
</tr>
</tbody>
</table>
### 17.4 Troubleshooting gator issues

Consult the following table for solutions to issues related to gator.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel version before 4.6 with CONFIG_CPU_PM enabled produces invalid results. For example counters not showing any data, large spikes, and non-sensible values for counters. This issue is a result of the kernel PMU driver not saving state when the processor is powered down, or not restoring state when it is powered up.</td>
<td>Upgrade to the latest version of the kernel, or apply the patch found at <a href="https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=da4e4f18afe0f3729d68f3785c5802f786d36e34">https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=da4e4f18afe0f3729d68f3785c5802f786d36e34</a>. This patch applies cleanly to version 4.4, and it may also be possible to back port it to other versions. If you apply the patch, you may also need to apply the patch at <a href="https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=cbcc72e037b8a3eb1fad3c1ae22021df21e97a51">https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=cbcc72e037b8a3eb1fad3c1ae22021df21e97a51</a>.</td>
</tr>
<tr>
<td>Incorrect data is displayed, or unexpected behavior is exhibited, for example the target kernel crashes.</td>
<td>Use kernel space gator. User space gator is in beta release with known issues.</td>
</tr>
<tr>
<td>An Oops occurs when a processor is offline (user space gator only).</td>
<td>The fix was merged into mainline in 3.14-rc5, see <a href="http://git.kernel.org/tip/e370f8c6c25c4338c3ad8e68891c2a3731">http://git.kernel.org/tip/e370f8c6c25c4338c3ad8e68891c2a3731</a>, and has been backported to older kernels (3.4.83, 3.10.33, 3.12.14, and 3.13.6).</td>
</tr>
<tr>
<td>dmesg output: CPU PMU: CPUx reading wrong counter -1 (user space gator only)</td>
<td>Update to the latest Linux kernel, or use kernel space gator.</td>
</tr>
<tr>
<td>Scheduler switch resolutions are on exact millisecond boundaries (user space gator only).</td>
<td>Update to the latest Linux kernel, or use kernel space gator.</td>
</tr>
<tr>
<td>perf misidentifies the processor type. Check if you are affected by running <code>ls /sys/bus/event_source/devices/</code> and verifying that the listed processor type is the one expected. For example, an A9 should return the following: <code># ls /sys/bus/event_source/devices/ ARMv7_Cortex_A9 breakpoint software_tracepoint</code></td>
<td>Upgrade to a later version of the kernel, or comment out the following call in <code>gator_events_perf_pmu.c</code>: <code>gator_events_perf_pmu_cpu_init(gator_cpu, type);</code></td>
</tr>
<tr>
<td>Problem</td>
<td>Solution</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Error message generated:</td>
<td>Disable SELinux then mount gatorfs by running <code># setenforce 0</code>. When gator starts, you can reenable SELinux.</td>
</tr>
<tr>
<td>Unable to mount the gator filesystem needed for profiling</td>
<td></td>
</tr>
<tr>
<td>or</td>
<td></td>
</tr>
<tr>
<td>Unable to load (insmod) gator.ko driver</td>
<td></td>
</tr>
<tr>
<td>while using SELinux and with the following dmesg output:</td>
<td></td>
</tr>
<tr>
<td>&lt;7&gt;[ 6745.475110] SELinux: initialized</td>
<td></td>
</tr>
<tr>
<td>(dev gatorfs, type gatorfs), not</td>
<td></td>
</tr>
<tr>
<td>configured for labeling</td>
<td></td>
</tr>
<tr>
<td>&lt;5&gt;[ 6745.477434] type=1400</td>
<td></td>
</tr>
<tr>
<td>audit(1393005053.336:10): avc: denied</td>
<td></td>
</tr>
<tr>
<td>{ mount } for pid=1996 comm=&quot;gatord-main&quot; name=&quot;/&quot; dev=&quot;gatorfs&quot;</td>
<td></td>
</tr>
<tr>
<td>ino=8733 scontext=u:r:shell:s0 tcontext=u:object_r:unlabeled:s0</td>
<td></td>
</tr>
<tr>
<td>tclass=filesystem</td>
<td></td>
</tr>
<tr>
<td>Mali Filmstrip does not work. dmesg output similar to:</td>
<td>Use streamline_annotate.h and streamline_annotate.c from DS-5 v5.20 or later, or disable SELinux by running</td>
</tr>
<tr>
<td>&lt;4&gt;[ 585.367411] type=1400</td>
<td># setenforce 0</td>
</tr>
<tr>
<td>audit(1421862808.850:48): avc: denied</td>
<td></td>
</tr>
<tr>
<td>{ search } for pid=3681 comm=&quot;mali-renderer&quot; name=&quot;/&quot; dev=&quot;gatorfs&quot;</td>
<td></td>
</tr>
<tr>
<td>ino=22378 scontext=u:r:untrusted_app:s0 tcontext=u:object_r:unlabeled:s0</td>
<td></td>
</tr>
<tr>
<td>tclass=dir</td>
<td></td>
</tr>
<tr>
<td>Annotations do not work on Android.</td>
<td>Disable SELinux by running <code># setenforce 0</code>.</td>
</tr>
<tr>
<td><strong>Problem</strong></td>
<td><strong>Solution</strong></td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Target does not correctly emit uevents when processors go online or offline. As a result, CPU Activity with user space gator is 0% or 100% on a given processor. The Heat Map may also display many unresolved processes. To test for this issue, run `# ./gatord -d</td>
<td>grep uevent<code>. When processors go online or offline with user space gator, output similar to the following is emitted: INFO: read(UEvent.cpp:61): uevent: offline@/devices/system/cpu/cpu1 INFO: read(UEvent.cpp:61): uevent: online@/devices/system/cpu/cpu1 Check the processors that are online or offline by running the following command: </code># cat /sys/devices/system/cpu/cpu*/online<code>If the online and offline processors that the</code>cat` command shows change, but no cpu uevent is emitted, this issue affects the target.</td>
</tr>
<tr>
<td>The following issue occurs when starting gatord using ndk-build on an older version of Android: <code># ./gatord</code> [1] + Stopped (signal) ./gatord <code>#</code> [1] Segmentation fault ./gatord <code>#</code></td>
<td>Starting with Android-L, only position independent executables (pie) are supported. However, some older versions of Android do not support them. To avoid this issue, modify Android.mk and remove the references to pie.</td>
</tr>
</tbody>
</table>
### 17.5 Troubleshooting perf import issues

Consult the following tables for potential error messages and warnings related to importing perf data.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error message generated:</td>
<td></td>
</tr>
<tr>
<td>Big-endian traces are currently not supported.</td>
<td>Only import trace files that are made on little-endian targets.</td>
</tr>
<tr>
<td>Error message generated:</td>
<td></td>
</tr>
<tr>
<td>Unable to decode individual event ids from the trace.</td>
<td>Raise a support issue with Arm and provide an example of the perf recording that triggered the error message.</td>
</tr>
<tr>
<td>An unexpected error occured.</td>
<td></td>
</tr>
<tr>
<td>Duplicate event ids found.</td>
<td></td>
</tr>
<tr>
<td>Invalid cluster map data.</td>
<td></td>
</tr>
<tr>
<td>Sample found with id that does not map to any known event.</td>
<td></td>
</tr>
<tr>
<td>Unexpected read_format value.</td>
<td></td>
</tr>
<tr>
<td>Error message generated:</td>
<td></td>
</tr>
<tr>
<td>File not recognized.</td>
<td></td>
</tr>
<tr>
<td>The input file was not a recognized perf recording, or was created with an unsupported version of the tool.</td>
<td>Ensure that the file is in the correct perf format. Update your version of the perf tool if necessary.</td>
</tr>
<tr>
<td>Error message generated:</td>
<td></td>
</tr>
<tr>
<td>Missing timestamp information. Make sure perf record has -T flag set.</td>
<td>Add the -T argument to perf record.</td>
</tr>
<tr>
<td>The file does not contain timestamps in events.</td>
<td></td>
</tr>
<tr>
<td>Error message generated:</td>
<td></td>
</tr>
<tr>
<td>Missing HEADER_EVENT_DESC section.</td>
<td>Upgrade your version of the perf tool.</td>
</tr>
<tr>
<td>Missing HEADER_PMU_MAPPINGS section.</td>
<td></td>
</tr>
<tr>
<td>Missing HEADER_TRACING_DATA section.</td>
<td></td>
</tr>
<tr>
<td>&quot;sample_id_all&quot; flag is not set on at least one attribute.</td>
<td></td>
</tr>
<tr>
<td>The file does not contain sufficient information for it to be successfully converted.</td>
<td></td>
</tr>
</tbody>
</table>
### Problem

Error message generated:
A recorded event is missing
PERF_SAMPLE_READ sample_type, or
PERF_SAMPLE_PERIOD is missing.
An event that is not counted was configured.

### Solution

Add the $ modifier, specify a period greater than zero, or use frequency sampling alongside recording the sample period. Use the -F argument for frequency sampling and the -P argument for the sample period.

### Error message generated:

The input file does not contain any sample records for any recognized counter.

### Solution

Ensure the recording contains at least one sample for a counter that Arm Streamline recognizes.

### Warning

Warning in **Timeline** view:
Some counters did not have samples associated with them and are excluded.

### Explanation

Some of the recorded counters do not have recorded samples, and therefore do not appear in Arm Streamline. The excluded counters are listed.

Warning in **Timeline** view:
Missing cluster map.

### Explanation

Arm Streamline attempts to work out the cluster configuration based on which processor, or subset of processors, each recorded event is associated with. Arm Streamline also attempts to assign appropriate labels, such as A53 and A57, to these clusters.

This warning means that it is not possible to know from the recorded data what the clusters on the device were. As perf record does not explicitly store the names of clusters, heuristics are used to work them out. However, this method can often fail, which means that processor PMU counters are not separated into different clusters. Therefore counter values appear as though there is no clustering.

### Related concepts

- **3.8 Importing perf data** on page 3-71
## 17.6 Troubleshooting Graphics Analyzer Mode issues

If you encounter a problem running Graphics Analyzer, consult the following list of possible errors. These errors are listed in the order in which Streamline detects them.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error message generated:</td>
<td></td>
</tr>
<tr>
<td>No Graphics Analyzer Daemon running on target device.</td>
<td>Launch <code>aga-daemon</code> on the target device.</td>
</tr>
<tr>
<td>Error message generated:</td>
<td></td>
</tr>
<tr>
<td>Timed out connecting to Graphics Analyzer Daemon.</td>
<td>Stop any ongoing tracing operations.</td>
</tr>
<tr>
<td>Arm Streamline detected that <code>aga-daemon</code> is already</td>
<td></td>
</tr>
<tr>
<td>busy tracing and cannot perform a new tracing request.</td>
<td></td>
</tr>
<tr>
<td>Error message generated:</td>
<td></td>
</tr>
<tr>
<td>Unsupported version of Graphics Analyzer Daemon is running.</td>
<td>Install a newer version of <code>aga-daemon</code> which supports the Graphics</td>
</tr>
<tr>
<td>Analyzer mode feature.</td>
<td></td>
</tr>
<tr>
<td>Error message generated:</td>
<td></td>
</tr>
<tr>
<td>No Graphics Analyzer executable found.</td>
<td>Open Window &gt; Preferences &gt; Streamline &gt; External Tools and set the</td>
</tr>
<tr>
<td>Please check the location specified in Preferences-&gt;External Tools.</td>
<td>path to the Graphics Analyzer executable in the Graphics Analyzer</td>
</tr>
<tr>
<td>Error message generated:</td>
<td></td>
</tr>
<tr>
<td>Unsupported version of installed Graphics Analyzer application found.</td>
<td>Upgrade to Graphics Analyzer version 4.0.0.</td>
</tr>
<tr>
<td>Version x.x.x found, but at least version 4.0.0 required.</td>
<td></td>
</tr>
<tr>
<td>Error message generated:</td>
<td></td>
</tr>
<tr>
<td>Mali Graphics Debugger x.x.x reported an invalid protocol version</td>
<td>Upgrade to Graphics Analyzer version 4.0.0.</td>
</tr>
<tr>
<td>number. Please make sure the version of Graphics Analyzer you are</td>
<td></td>
</tr>
<tr>
<td>using is compatible with this version of Streamline.</td>
<td></td>
</tr>
<tr>
<td>Error message generated:</td>
<td></td>
</tr>
<tr>
<td>Unable to detect Graphics Analyzer application version.</td>
<td>Ensure the path in the dialog at Window &gt; Preferences &gt; Streamline &gt;</td>
</tr>
<tr>
<td>Please ensure the executable path is valid.</td>
<td>External Tools points to a valid installation of Graphics Analyzer.</td>
</tr>
<tr>
<td>Problem</td>
<td>Solution</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Error message generated:</td>
<td>Ensure that the connection with the target device is working correctly, and that the correct version of <code>aga-daemon</code> is installed and running on the device. Also ensure that <code>aga-daemon</code> is contactable through port 5002 and is not, for example, blocked by a firewall.</td>
</tr>
<tr>
<td>An error occurred when attempting to detect Graphics Analyzer Daemon version.</td>
<td></td>
</tr>
<tr>
<td>Error message generated:</td>
<td>For adb connections, ensure that the path to the adb executable is set in the dialog at <strong>Window &gt; Preferences &gt; Streamline &gt; External Tools</strong>. Also ensure that there are no conflicting port forwarding rules which are already configured with adb.</td>
</tr>
<tr>
<td>Could not determine hostname for target device, or 'adb forward' failed.</td>
<td>For non-adb connections, treat this error as a bug.</td>
</tr>
<tr>
<td>If Arm Streamline cannot determine the hostname for the target device, it is not possible for it to work out the IP address where <code>aga-daemon</code> is running.</td>
<td></td>
</tr>
<tr>
<td>Alternatively for adb targets, the command <code>adb forward</code> failed and Arm Streamline was not able to forward the <code>aga-daemon</code> port so that it could communicate with <code>aga-daemon</code>.</td>
<td></td>
</tr>
</tbody>
</table>

**Related concepts**

6.7 Graphics Analyzer Mode in **Live view** on page 6-148
## Troubleshooting Energy Probe issues

If you encounter a problem using the Energy Probe, consult the following list of potential problems and error messages.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
</table>
| Difficulty running caiman from Arm Streamline. | Run `caiman` on the command line in local mode. For example:  
```
    DS_install_directory/bin/caiman -l -r 0:20
```
  
  Information to assist with debugging is displayed. If no messages are printed after a few seconds, you can kill `caiman`.  
  
  If there are no problems, the `0000000000` file is non-empty. |
| Error message generated: Unable to detect the energy probe | Disconnect and reconnect the Energy Probe, and ensure the Energy Probe properly enumerates with the OS. This problem can occur on some operating systems that do not properly re-enumerate the Energy Probe device after rebooting, or going into sleep or hibernate. |
| Error message generated: Unable to set `'/dev/ttyACM0' to raw mode, please verify the device exists` | Check the permissions of `/dev/ttyACM0`. You may need to add your username to a group or modify the permissions of `/dev/ttyACM0`. If the problem persists, a different program may be using the device. Use `lsof` to debug further. On Ubuntu, after plugging in the Energy Probe, the modem-manager opens the device for a while. |
| `/dev/ttyACM0` does not exist after plugging in the energy probe. | Upgrade to Linux version 2.6.36 or later. |
| Error message generated: `bash: ./caiman: No such file or directory` | Ensure `/lib/ld-linux.so.2` is installed. It is in the `libc6-i386` package on Ubuntu. This problem can occur when running the 32-bit version of `caiman` on x86_64. |
| The power values in Arm Streamline look incorrect. | Ensure that the shunt resistor values are correct in the Energy Capture section of the Capture & Analysis Options dialog box. |
| Missing channels. | Ensure either power, voltage, or current is checked in the Energy Capture section of the Capture & Analysis Options dialog box. |
| Data is reporting zero or close to zero values. | Check the connections. If you are using the Energy Probe, ensure that the green LED is on. |
| Error message generated: `caiman-src/EnergyProbe.cpp:27:21: error: libudev.h: No such file or directory` | Your platform does not support udev or is missing the udev headers. Either install `libudev-dev`, or equivalent, or disable udev support in `CMakeLists.txt` by setting `SUPPORT_UDEV` to 0. |
# 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.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arm Streamline does not show any source code in the Code view.</td>
<td>Make sure that you use the <code>-g</code> option during compilation. Arm Streamline must have debug symbols turned on in order to match instructions to source code. If necessary, ensure that the path prefix substitutions are set correctly. See the Path Prefix Substitutions dialog box in the Code view.</td>
</tr>
<tr>
<td>Arm Streamline does not show source code for shared libraries.</td>
<td>Add the libraries using the Capture &amp; Analysis Options dialog box. Click Add ELF Image... in the Program Images section, navigate to your shared library, and then add it.</td>
</tr>
</tbody>
</table>
| The data in the Call Paths view is flat. The presented table is a list rather than a hierarchy. | Use the GCC options `-fno-omit-frame-pointer` and `-marm` during compilation. `-marm` is only required for Armv7 and earlier architectures. Also, check the Call Stack Unwinding option in the Capture & Analysis Options dialog box. If frame pointers are being set correctly, the disassembly of the code should contain instructions in the form `add fp,sp,#<n>` at the start of functions. To generate a disassembly, use the following command:  
```bash
arm-linux-gnueabihf-objdump -d foo.so
```
| Note | By default, Arm Streamline does not walk the stack for kernels or loadable kernel modules. These generate flat data in the Call Paths view unless you built gator.ko to perform kernel stack unwinding.  
• Arm Streamline does not walk the stack for statically-linked drivers.  
• User space gator does not support call stack unwinding. |
| Functions that you know are highly used are missing from the reports. Other functions might seem artificially large. | This can be because of code inlining done by the compiler. To turn inlining off, add `-fno-inline` as an option during compilation. |
| A newly-generated capture has no data. | If you experience this and the profiling session had event-based sampling enabled, the PMU on your target might not have triggered the interrupts correctly. Test on alternate hardware or disable event-based sampling in the Counter Configuration dialog box. |

## Related tasks
- 12.9 Profiling the Linux kernel on page 12-222
- 8.3 Path prefix substitution in the Code view on page 8-176

## Related references
- 17.2 Troubleshooting target connection issues on page 17-290